Performance Analysis of an Adaptive User Interface System Based on Mobile Agents

Nikola Mitrović, Jose A. Royo, and Eduardo Mena

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

Abstract. Adapting graphical user interfaces for various user devices is one of the most interesting topics in today's mobile computation. In this paper we present a system based on mobile agents that transparently adapts user interface specifications to the user device' capabilities and monitors user interaction. Specialized agents manage GUI specification according to the specific context and user preferences. We show how the user behavior can be monitored at run-time in a transparent way and how learning methods are applied to anticipate future user actions and to adapt the user interface accordingly. The feasibility and performance of our approach are shown by applying our approach to a non-trivial application and by performing tests with real users.

1 Introduction

Adapting graphical user interfaces (GUIs) to different devices and user preferences is one of the most challenging questions in mobile computing and GUI design. User devices have different capabilities, from small text-based screens and limited processing capabilities to laptops and high-end workstations. Another important challenge is to adapt user interfaces to user preferences, context, and GUI actions to be performed. Some of these parameters, user preferences, depends on the specific user while others, user's context or actions, do not. However all these parameters vary over time which makes them more difficult to manage.

Mobile environments are particularly challenging: mobile devices require applications with small footprints, written for specific proprietary platform that can execute on devices with very limited capabilities and resources. Mobile devices connect to other devices by using wireless networks which are more expensive[1], unreliable, and slower, than their wired counterparts. Handling these problems is very difficult and applications are frequently written to accommodate specific devices and environment. Developing such applications requires a significant effort and expertise therefore portability across different user devices is a must.

To create user interfaces that can adapt to different devices and situations researchers use *abstract user interface definition languages* as a common ground. The

[1] In the case of wireless WAN's.

abstract definition (usually specified in XML-based notation) is later rendered into a concrete (physical) user interface. Many abstract GUI definition languages exist: XUL [30], UIML [1], XIML [34], XForms [32], usiXML [31], just to name few. To adapt an abstract GUI definition to a real GUI researchers use client-server architectures [8], specialized tools to create separate GUIs for different platforms [22], and other take advantage of agent technology [18, 14].

Current GUI design methods lead to the re-design and re-implementation of applications for different devices. In addition, direct generation of user interfaces do not allow the system to monitor the user interaction which can be useful for adaptive systems. Our proposal to generate and manage adaptive GUIs is *ADU*S *(ADaptiv*e *Use*r *Interfac*e *System*) [18] which is based on an abstract graphical user interface definition language and a mobile agent architecture. Thus, while abstract a GUI definition language gives flexibility when describing a user interface, mobile agents allow flexible rendering of such a GUI definition and provide abstraction from other application layers (e.g., platform, connectivity problems, etc). Thus we adopt this approach as it enables the creation of flexible user interfaces that are able to adapt and move through the network. The ADUS system also enables adaptation to user preferences, context, and actions by monitoring and analyzing the user behavior [21]; such a collected knowledge is reused in future program executions to anticipate the user's actions.

In this paper we present the advantages of using ADUS in mobile computing applications, specifically, we show how learning from user actions on the generated GUI improves the performance of the system. For this task, we describe how ADUS has been used in a software retrieval service and the results of testing both versions (with and without ADUS) with real users.

The rest of this paper is as follows. In Section 2 we describe the main features of ADUS. Section 3 describes how ADUS learns from the user behavior and anticipates future user actions. In Section 4 we apply ADUS to a non-trivial sample application. Performance and usability evaluations of such a system are presented in Section 5. Section 6 gives an overview of the state of the art and the related work. Finally, conclusions and future work are presented in Section 7.

2 ADUS: Adaptive User Interface System

The ADaptive User interface System (ADUS) is an approach based on mobile agents that generates user interfaces adapted for different devices at run-time [18]. To provide this functionality, agents manage abstract descriptions of graphical user interfaces to be deployed. While abstract UI definition languages give flexibility in describing user interface, mobile agents allow flexible rendering of the UI definition and provide abstraction of other application layers (e.g., platform, connectivity problems, etc). We adopt this approach as it enables the creation of a flexible user interface capable of adapting and moving through the network. ADUS is part of the ANTARCTICA system [15] that provides users with different wireless data services aiming to enhance the capabilities of their mobile devices.

As GUI definition language we use XUL (*eXtensibl*e *Use*r *interfac*e *definition Languag*e) [30]. The GUI is specified in XUL and then transformed on the fly by mobile agents to a concrete user interface. Some of the GUI properties, such as window size, colors, and widgets used, are adapted on the fly. In addition, GUI sections and elements can be modified by mobile agents at the run time (see Section 3.4). The developed prototype can adapt such user interface descriptions to Java AWT, Java Swing, HTML, and WML clients, and supports limited plasticity [29]. GUI widgets are mapped to the concrete UI using CC/PP [4] and different transformation engines; further plasticity improvements are planned as future work.

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 [16, 17]. Mobile agents can bring computation wherever needed and minimize the network traffic, especially in wireless networks (expensive, slow, and unstable), without decreasing the performance of the system [33]. In our context, mobile agents are able to arrive at the user device and show their GUIs to the user in order to interact with her/him [18]. The deployment of mobile agents is automatic and has little performance overheads [33]. In our prototype we use the mobile agent system Voyager [9]; however any other mobile agent system could be used to implement our approach.

Our system uses indirect user interface generation [21] which is a method where several agents collaborate in order to transparently produce user interfaces adapted to users and devices. The main steps are (see Figure 1):

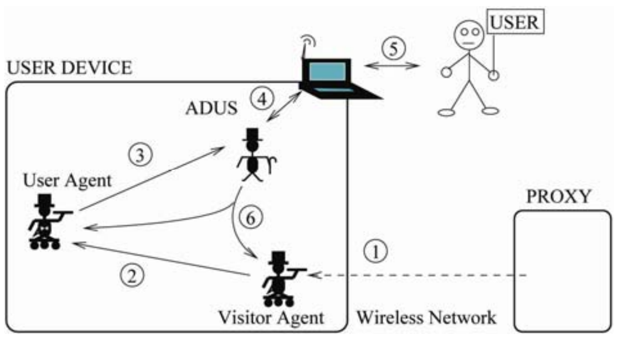


Fig. 1. Indirect generation of GUIs

1. A visitor agent arrives at the user device to interact with the user.
2. The visitor agent, instead of generating a GUI directly, generates a XUL [30] specification of the needed GUI, which is sent to the user agent who applies user-specific information to the GUI specification. This modification is based on user's preferences, context, or collected knowledge. For example, the user agent could use data from previous executions to automatically assign the values that were entered by the user to past visitor agents requesting the same information [15, 21]
3. The user agent creates an ADUS agent initialized with the new GUI specification.
4. The ADUS agent generates the GUI which will include the specific features for that user and for that user device.

5. The user interacts with the GUI.
6. The ADUS agent handles and propagates the GUI events to 1) the visitor agent, who should react to such events, and 2) the user agent, which in this way monitors and learns from such user actions.

The additional benefit of such a transparent user interface generation is the simplicity of software development – using our approach only one version of user interface and application code is developed (in XUL) but the corresponding GUIs are automatically generated for very different user devices without user or software developer intervention.

3 User Interaction Monitoring and Application: The Learning Process

One of the key features of our prototype is the ability to monitor and collect user interaction information at the run time [21]. The prototype monitors both GUI interaction and interaction between the visitor agent and the user using the indirect user interface generation model, as explained before. Such data can be used to examine user's behavior and apply the collected knowledge on the subsequently generated user interfaces. The monitoring mechanism does not depend on the type of application or platform. It is important to notice that, as the monitoring mechanism is based on mobile agents, it is distributed, mobile, and can be extended with security frameworks for mobile agents [20].

Our prototype uses data mining techniques to anticipate user's actions. In addition, our prototype utilizes task models as training data for data mining techniques. In the following paragraphs we present the techniques used in our prototype.

3.1 Predicting User Behavior

Predicting the user behavior is a difficult task: a common methodology to predict users' behavior is predictive statistical models. These models are based on linear models, TFIDF (Term Frequency Inverse Document Frequency), Markov Models, Neural Methods, Classification, Rule Induction, or Bayesian Networks [35]. Evaluation of predictive statistical models is difficult -some perform better than other in specific contexts but are weaker in other contexts [35].

We advocate using Markov-based models as they behave better for our goal while retain satisfying prediction rates [24, 6, 19]. Specifically, in our prototype we use the Longest Repeating Subsequence (LRS) method [24]. A longest repeating subsequence is the longest repeating sequence of items (e.g. user tasks) where the number of consecutive items repeats more than some threshold T (T usually equals one).

3.2 Task Models

Statistical models such as LRS can be beneficial for predicting user actions. However, there are two major drawbacks to such models: 1) in order to predict next actions, training data must be supplied before the first use, and 2) poor quality training data can potentially divert users from using preferred application paths.

Contrary to statistical models which are created at run-time, task models are created during the design phase of an application. Task models are often defined as a description of an interactive task to be performed by the user of an application through the user interface of the application [13]. A task model represents the static information on users and application tasks and their relationships.

Many different approaches to defining task models have been developed [13]: Hierarchical Task Analysis (HTA) [26], ConcurTaskTrees (CTT) [23], Diane+ [2], MUSE [12], to name few. We use CTT, developed by Patterno [23], as it provides well developed tools for defining concurrent task trees.

Task models successfully describe static, pre-designed interaction with the users. However, it is very difficult (if not impossible) to describe with sufficient accuracy (for user behavior predictions) user-application interaction in case application tasks change dynamically. For example, if the application changes its tasks dynamically based on the information downloaded from the Internet, the task model of such an application would be a high-level description; task models would not be able to model precisely the dynamic tasks created as per downloaded information. This is because information used to create tasks from the Internet is not known to the software developer at the design time, and some generic task or interaction description would have to be used in the task model.

In our prototype we use specially crafted CTT models as pre-loaded training data to statistical learning modules. CTT models used are very basic and do not follow closely CTT standard notation; models are specifically customized for our use.

3.3 Learning Models in ADUS

Behavior analysis and learning in our system are provided by two separate knowledge modules. The first module treats user preferences and simple patterns (e.g. modifying the menus or font size). The second module is specialized in LRS-based behavior analysis. Both knowledge modules help the user agent make the necessary decisions that are later reflected on the user interface [21].

To improve LRS predictions we have developed a specialized converter utility that can convert specifically crafted CTT definition into LRS paths database. The converter utility is very basic – the CTT diagrams must be specifically prepared to accommodate our converter tool which involves supplying object tags as per our specification and designing trees with LRS in mind. In the current version of the prototype CTT diagrams are very basic and do not follow closely CTT task types. Previously prepared information from CTT can be then loaded into the LRS module as the default knowledge with a configurable weight (i.e. path preference). This has been designed to: 1) ensure existence of the initial training data (before the first use), and 2) to ensure that the paths supplied by the GUI designer have certain initial priority (weight) over dynamically collected paths. Such measures could improve overall user experience and could improve quality of dynamically collected data.

However, the learning mechanism implemented in ADUS is agnostic - different learning techniques can be implemented at the same time. Learning process is not limited to tasks, but can be extended (with different learning techniques) to any other type of learning.

3.4 Applications of Learning Features to the User Interface

Gathered knowledge (e.g., default values, color preferences, or previous actions and selections) is applied by the user agent to the GUI specification. The LRS method is more closely linked to tasks and user interaction paths and has been visually implemented as a *predictive toolbar* (see Section 4.3 and Figure 4). The user agent automatically inserts this toolbar in the application window (unless otherwise specified) and it shows a configurable number of next-most-probable actions [19].

In cases when software developers anticipate that predictive toolbar would not be useful for the user (e.g. applications where the toolbar would not be visible, or where tasks are not executed through buttons), the LRS module could be used by the visitor agent through the user agent. Section 4.3 presents in detail usage modalities of the LRS module.

4 Using ADUS in a Sample Application

To show the benefits of learning techniques to GUI and complex GUI transformations we have applied the ADUS approach to a multi-agent application –the *Software Retrieval Service (SRS)* [15]. The Software Retrieval Service tries to solve one of the most frequent tasks for an average computer user: to search, download, and install new software.

In the following we briefly introduce the agents that participate in the SRS and then we describe how the ADUS approach is applied. The resulting system is tested by real users in Section 5.

4.1 The Software Retrieval Service (SRS)

The Software Retrieval Service [15] is an application that helps naive users to find, download, and install new software on their devices. The SRS is distributed between the user's device (also known as *user place*) and a proxy location (known as *software place*), as illustrated in Figure 2.

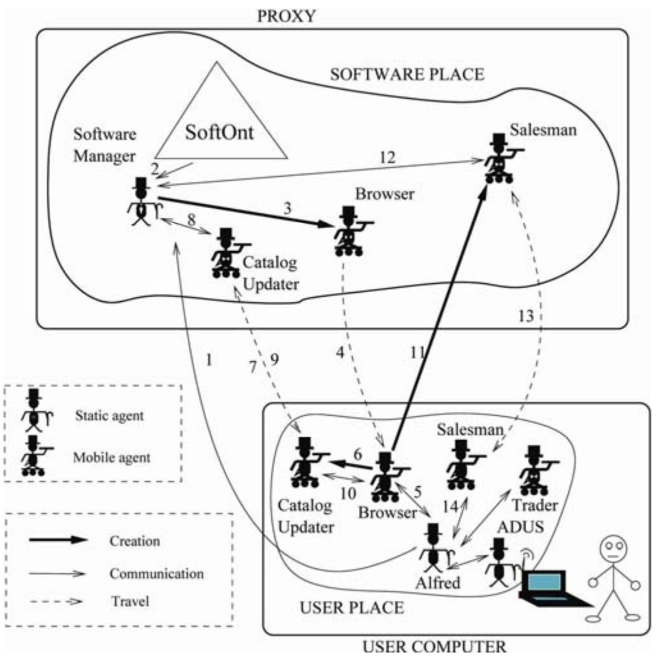


Fig. 2. Main architecture for the Software Retrieval Service

In the following paragraphs we briefly describe the main agents of the SRS (more details about this system can be found in [15]):

1. *The Alfred agent.* It is a user agent that serves the user and is in charge of storing as much information about the user equipment, preferences, and context as possible. Mobile agent technology allows that mobile agents can learn (e.g. using information from the Web) about previously unknown contexts.
2. *The Software Manager agent.* It creates and provides the Browser agent with a catalog of the available software, according to the requirements supplied by Alfred (on behalf of the user, step 1 in Figure 2), i.e., it is capable to obtain customized metadata about the underlying software.
3. *The Browser agent.* It travels to the user device (step 4) with aim to interact with the user (see Figure 3) in order to help her/him browse the software catalog (step 5).

Working in this way – without ADUS – the Browser agent directly generates its GUI on the user device without knowing user preferences and user device capabilities.

4.2 Using ADUS with the Software Retrieval Service

When applying the ADUS approach to the SRS application, Alfred plays the role of user agent and the Browser agent behaves as a visitor agent that arrives to the user device with the purpose of creating a GUI. An ADUS agent will be required to facilitate indirect user interface generation. The ADUS agent interacts with the SRS agents as follows:

1. The Browser agent (as depicted in Figure 2) sends the XUL specification of the GUI to Alfred.
2. Alfred amends the XUL specification according to the user preferences, context, and device capabilities. In this example, size and location of “split panes” are set by Alfred.
3. Alfred delegates the generation of the GUI to an ADUS agent, who renders the GUI, interacts with the user, and feeds interaction data to Alfred (the user agent) and the Browser (the visitor agent). Figure 3 shows the Java GUI generated by the ADUS agent for a Pocket PC PDA.

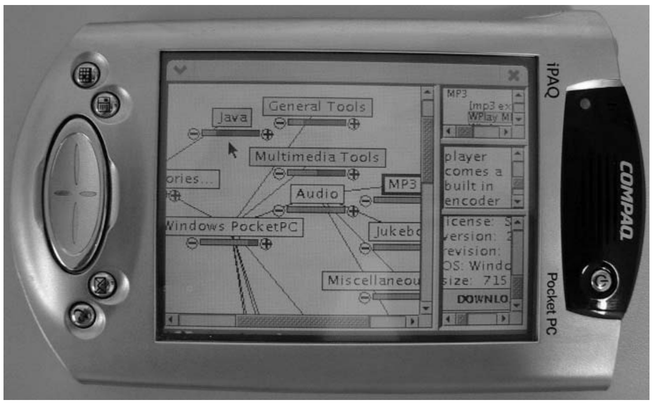


Fig. 3. Java Swing Browser GUI created indirectly on a PDA

4. GUI events and data received by the ADUS agent are communicated to Alfred and the Browser agent for further processing. Alfred stores and analyses such data to predict future user actions, and the Browser agent reacts to the selections or data entered by the user by generating new or updating the existing GUI.

The above process is repeated until the Browser (the visitor agent) finishes its tasks on the user device.

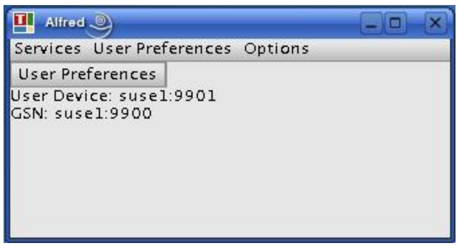
4.3 The Learning Process in the SRS

As described earlier, behavior analysis and learning are provided by the user agent (Alfred in the case of the SRS), which treats user preferences and predicts the user behavior following the stored patterns.

Once users start using the application, Alfred collects the necessary data by monitoring user-executed actions in an effort to predict the next task. In the current version of our prototype, the user agent Alfred monitors task execution only through button widgets. As the SRS Browser agent uses a customized interaction model, the visitor agent (the Browser agent in the example) can use the LRS module via the user agent (Alfred) to benefit from the learning features of the system (as described in Section 3.4).

The Browser agent uses the LRS module described earlier via Alfred to automatically expand or collapse browsing nodes (see Figure 3). The user agent will then expand the nodes that are identified as the next most probable nodes to be opened by the user[2].

In addition to the SRS Browser agent GUI, Alfred has its own GUI that is designed for configuration of user preferences, service options, and execution of other services. This GUI features the predictive toolbar automatically generated by Alfred as described in Section 3.4 and depicted in Figure 4. To improve the quality of training data, and to provide initial training data to the LRS module in Alfred's GUI, we have developed a CTT task model (see Figure 5). The task paths are extracted from the model using a converter utility and path weight is assigned to the paths.



User Device: susel:9901
GSN: susel:9900

Fig. 4. Alfred's GUI – predictive toolbar

5 Performance Evaluation

In this Section we present results of the performance tests and analyze differences in performance between using SRS with and without ADUS approach.

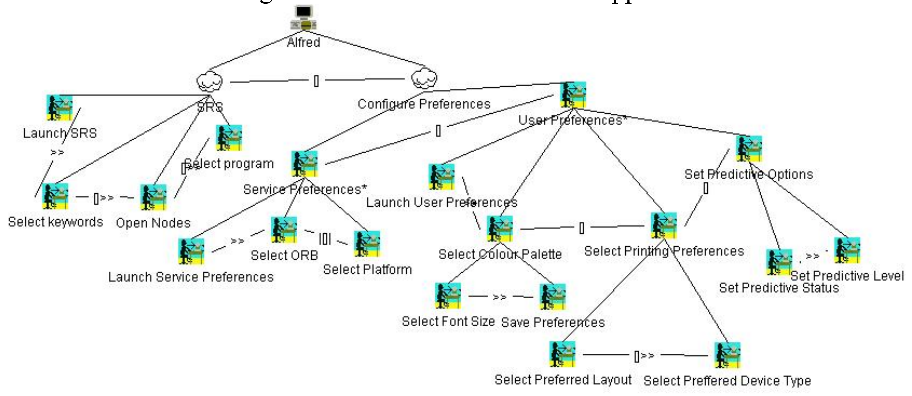


Fig. 5. CTT model for Alfred's GUI

[2] The main task of the Browser agent is to help the user the user to browse a software catalogue to find a certain software.

In our test, users[3] were asked to retrieve several pieces of software using the SRS application. The first half of the participating users used the SRS application without the ADUS architecture (direct GUI generation). The second half used the SRS application with ADUS (indirect generation of GUIs). 50 users with mixed levels of skill participated in this test.

In the first test we compare how the learning features of ADUS improve the system from the point of view of time-consuming tasks. Measured times have been divided into three categories:

– *Data transfer*: this is the time spent by the system 1) to send the different software catalogs to the user device, 2) to move an agent across the network, and 3) to invoke remote procedure calls[4] .
– *Reading catalog*: this category represents the time spent by the user to read/browse the software catalog shown on the device screen; this time includes to open/close a catalog node to read its information.
– *UI operations*: This measure quantifies the time spent by the system on GUI generation (and monitoring, when ADUS is used).

In [21] we showed that just using ADUS (without any prediction) improved the performance of the SRS despite the small overhead due to the indirect GUI generation and monitoring. From Figures 7 and 8 we can observe that the use of the LRS method reduce the total time spent by users to find the software and even the time spent by the system to generate GUIs: when estimations of user behavior are correct, users save several GUI interactions (and the system saves the corresponding (indirect) GUI generations). Figure 6 depicts times spent on the SRS application without ADUS.

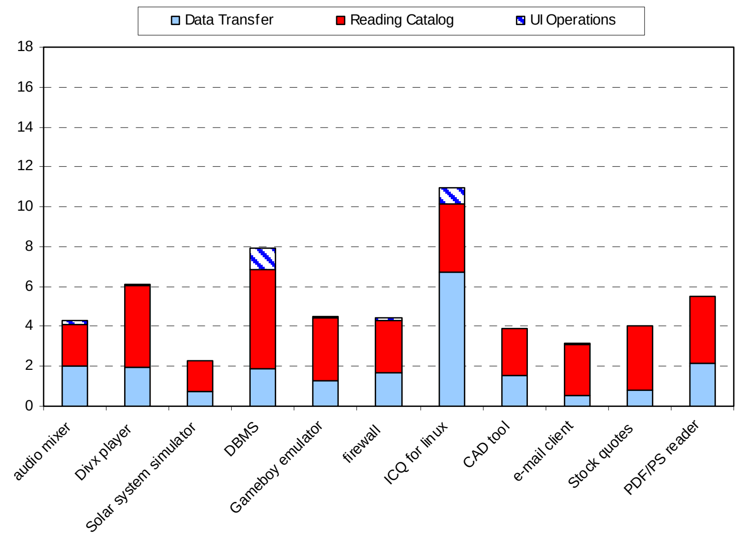


Fig. 6. Time-consuming tasks for SRS without ADUS

[3] The authors would like to express their gratitude to all persons participating in this study.

[4] Intelligent (mobile) agents in the SRS decide between whether to use remote procedure call or movement approach depending on the execution environment.

When the predictive features are used ADUS utilizes the data obtained from monitoring interaction between the user and the Browser agent to predict the users' next most probable action (see Section 3). The SRS application then expands and collapses browsing nodes according to the next most probable action. This way, the user interface is generated fewer times: multiple nodes are expanded or collapsed at the same time with only one processing of UI. In the previous version, without predictive features, nodes are expanded by the user manually which triggered additional UI operations.

The second test gives indication of whether predictive features were used and if they were useful. In Figure 9 we present usage of predictive features and the ratio of correct predictions. "Right" represents the percentage of correct predictions that have been followed by users. "Wrong" represents misleading predictions that have not been followed by users. "Ignored" represents percentage of correct predictions that were ignored by the users (they follow a non-optimal path).

Figure 9 shows that the predictive features had a good ratio of successful predictions (on average 90.25%). The average percentage of wrong predictions was 9.74%. 69.74% (on average) of requests followed the correct prediction which implies that predictive features have been seen as useful by most of the users. A certain percentage of requests (20.51%) however did not see the features as useful or felt that the predictions are erroneous.

In the next test we can observe that due to the predictive features the SRS Browser agent loads a better sample of data leading to lower network utilization (cost saving if wireless networks are used) which also results in better processing of the information from the network as more relevant data are downloaded.

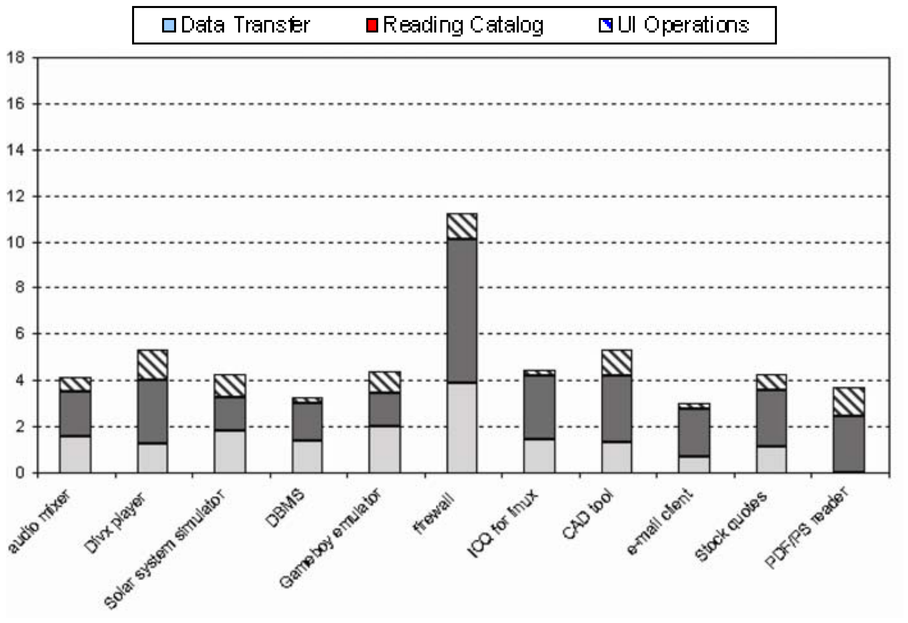


Fig. 7. Time-consuming tasks for SRS + ADUS without predictive features

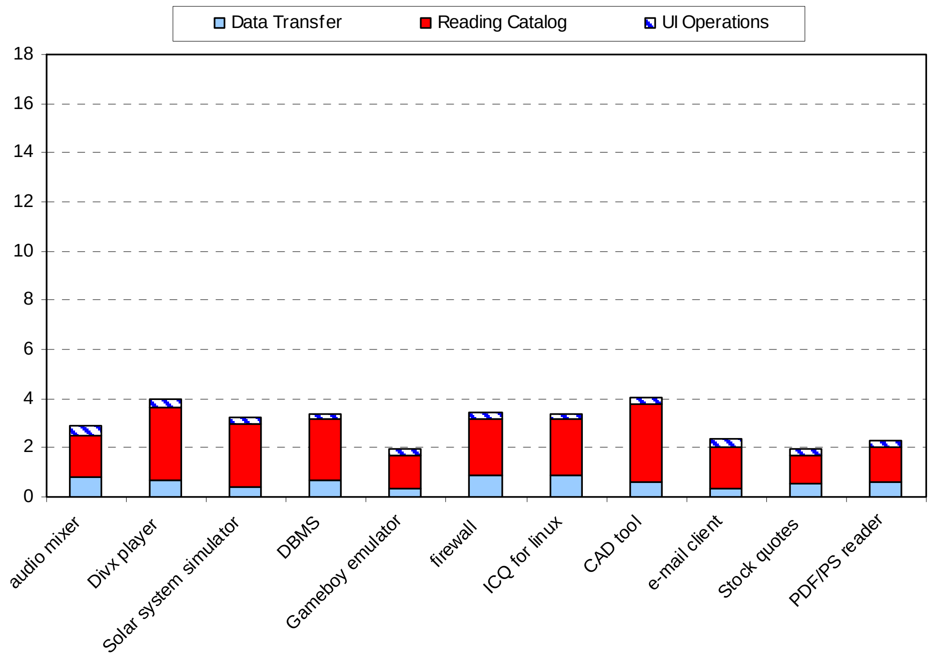


Fig. 8. Time-consuming tasks for SRS + ADUS with predictive features

This measurement is defined as *Browser (agent) intelligence* [15] and represents efficiency in refining software catalogs shown to the user.

Figure 10 shows a comparison among two versions of the Browser agent intelligence; the higher percentage, the better network and processing usage. On average the improvement due to ADUS with predictive features ranged from -2% to 23% (average of averages was 7%). To conclude, time to find the requested application using the SRS application with ADUS and predictive features has been improved through lower UI operations, network consumption and information processing due to correct predictions made by the system.

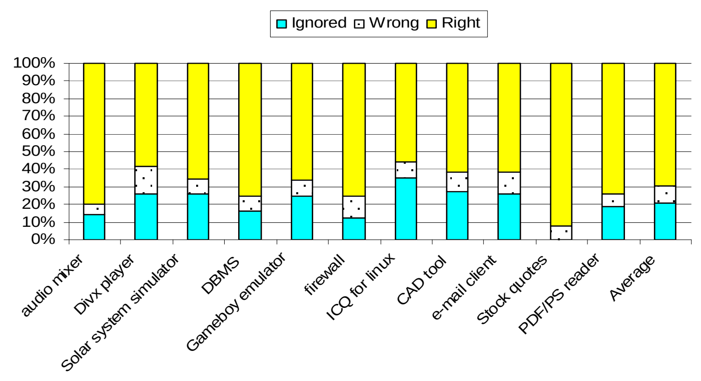


Fig. 9. Usage of Predictive Features

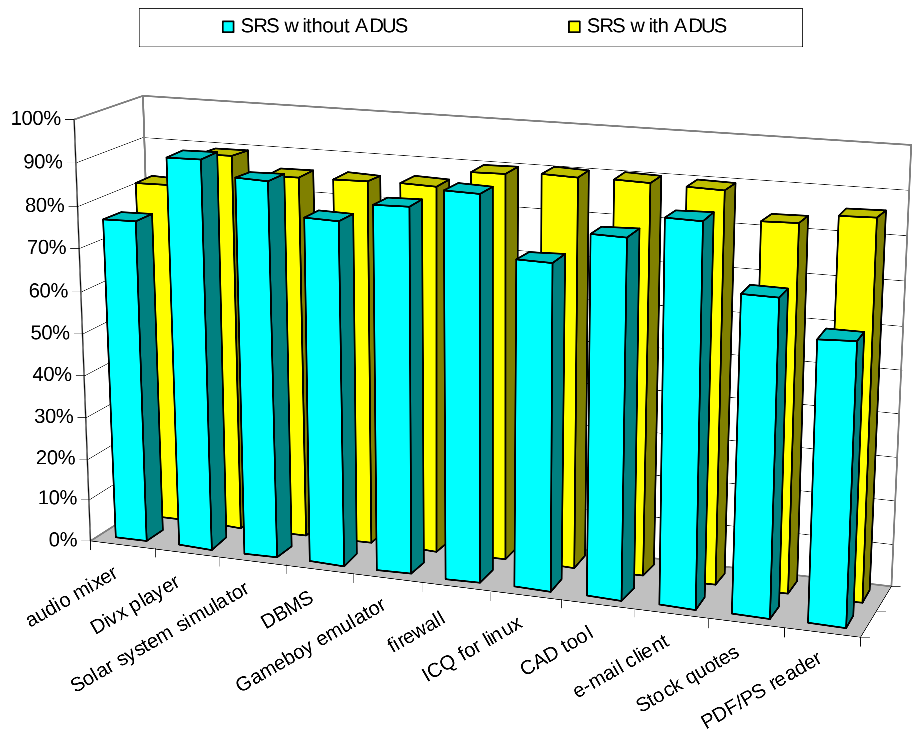


Fig. 10. Browser (agent) intelligence with and without predictive features

In addition to the measurable indicators we asked users to express the usefulness of the predictive features in the SRS application. Usability was measured in a relative way; users were asked to compare the SRS application without ADUS to the SRS application with ADUS with predictive features and the usability of predictive features in comparison with the original SRS without ADUS: scores range from 0 (not useful) to 10 (very useful). The score above 5 signifies that the ADUS versions of program are more preferred. Figure 11 shows the usability of 1) SRS with and without ADUS predictive features and 2) usability of predictive features alone in SRS with ADUS in comparison to the SRS without ADUS. The usability rating was surveyed for every task in order to understand better usability of predictive features relating to a particular task.

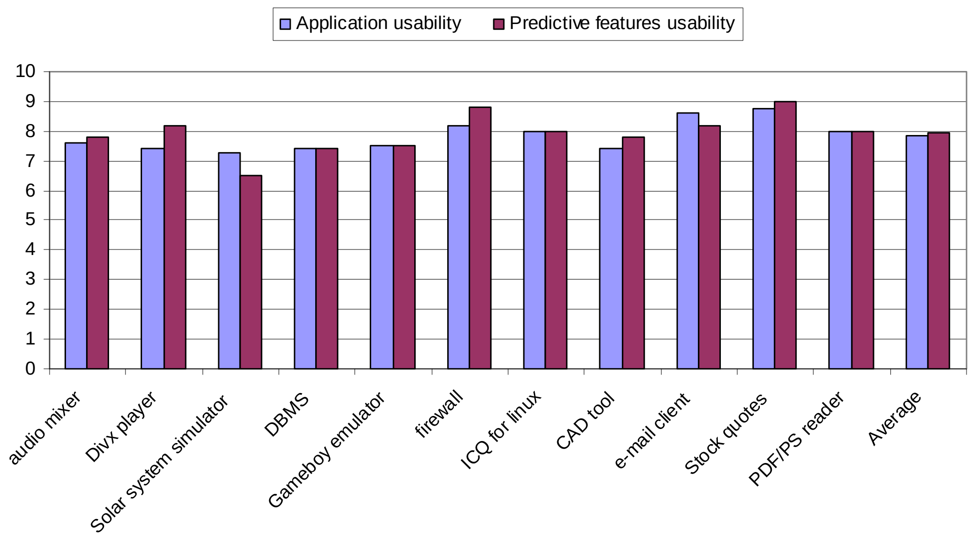


Fig. 11. Average usability of two SRS versions and predictive features

On average, the SRS version with ADUS and predictive features was seen as more usable than the version of SRS without ADUS. Similar results were obtained for the usability of predictive features. However, in some cases usability of predictive features has a much lower score than the application usability – this was typically a result of an erroneous prediction that confused users. In total, both the improved application and predictive features scored almost 3 points above the old system versions which shows that the improvements to the system have been seen as usable.

Results Summary

Tests were conducted with 50 users to demonstrate quantifiable difference between two versions of the SRS application: without and with ADUS and predictive features. It has been demonstrated that, although general GUI processing is increased when following ADUS approach, *the actual processing time decreases due to the application of predictive features*. In addition, information processing and network operations are reduced, which lowers the operational and usage cost of mobile applications on wireless networks.

Tests were also designed to measure usability of the system improvements through time to download, usage ratio of predictive features and number of correct predictions by the system. All tests concluded that improvements to the original application were made; a good percentage of predictions were correct and the predictive features have been used by the testers.

Furthermore we have examined some subjective factors: relative usability of two applications and relative usability of predictive features. The survey showed that both the improved application and predictive features were seen more usable than the original versions.

6 State of the Art and Related Work

In this section we present several approaches related to the work presented in this paper. Various approaches to adapting user interfaces to different devices are present. The approaches can be grouped into two categories: web applications and classic applications. While the first category [5, 8] 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 [1, 27, 11, 32, 22] —or middleware— on various platforms. Solutions are usually designed as client-server and are developed for specific platforms.

Some researchers use software agents (or software entities) [14, 7, 25] which should not be confused for mobile agents. Software agents are software programs that rarely offer any interoperability or mobility and are frequently specifically written for a particular case or application. Lumiere [7] system gives user behavior anticipation through the use of Bayesian models but does not offer any mobility and can be used only in Microsoft Office applications and with use of user profiles. Seo et al. [25] investigate software entities that are standalone, desktop applications. Such entities monitor use of the particular web browser application and provide some anticipation of interaction. The Eager system [28] anticipates user actions but does not offer any mobility and is written for specific operating system/application set. Execution of such system relies on generation of macro scripts within the used application set.

Improving user interface usability is a complex area and many approaches to improving usability exist. We will focus on three main approaches to improve user interface usability: user interface metrics, data mining – user behavior prediction, and task models. The basic concept is to collect user interface metrics for a web site [10]. Usually, collected data are used to perform traffic-based analysis (e.g., pages-per-visitor, visitors-per-page), time-based analysis (e.g., page view durations, click paths) or number of links and graphics on the web pages. These methods fail to give prediction of user behavior, and results can be influenced by many factors. In addition, such analysis is usually used during the UI design (and not in run-time) to improve existing or create new interfaces.

Many models that treat to predict user behavior are based on Markov chains [6]. Predictions are made based on the data from usage logs. More advanced models, like Longest Repeating Subsequence (LRS) [24] or Information Scent [3] perform data mining seeking to analyze navigation path based on server logs, similarity of pages, linking structure and user goals. These models incorporate parts of Markov models in order to give better results. Our prototype uses LRS model as described in Section 3.

Task models are often defined as a description of an interactive task to be performed by the user of an application through the application's user interface [13]. Task model is defined during the application design and gives information on user and application tasks and their relationships. Many different approaches to defining task models have been developed [13]: Hierarchical Task Analysis (HTA) [26], ConcurTaskTrees (CTT) [23], Diane+ [2], MUSE [12], to name few. Task models are typically used to help define and design user interface, and sometimes also to help create user interfaces during the design. In our prototype we use task models as source of training information for user interaction analysis.

7 Conclusions and Future Work

This paper presents results of performance and usability studies on ADUS, our proposal for adaptive user interface generation, which is based on mobile agents. In addition, it allows the user behavior monitoring due to its indirect user interface generation method. As summary, the main advantages of our approach are:

– Transparent adaptation of abstract user interface definition to concrete platforms, in an indirect way. GUIs supplied by visitor agents are generated correctly (according to the user preferences and device capabilities) if they are specified in XUL by visitor agents.
– Visitor agents do not need to know how to generate GUIs in different devices. Also the direct generation of GUIs by visitor agents can be easily avoided; direct GUI generation could undermine platform's efforts to improve user's experience and allow uncontrolled malicious behaviors such as phishing.
– User interfaces are adapted to meet the specific user's context and preferences without user or developer intervention.
– Any user interaction can be monitored by the system in order to help the user to interact with future invocations of services.
– The system learns from the user behavior to anticipate future user actions, with the goal of improving the performance and usability. The user behavior is analyzed and next most probable action is advertised. The prediction rate of the proposed algorithm used in our prototype is satisfactory. However, any other predictive algorithm or model could be used in ADUS.

Finally we have presented some performance and usability tests of the system. The performance results demonstrate that there are no significant processing overheads of the proposed architecture and that some performance benefits could be drawn by reducing GUI, network, and information processing operations through predicting future states of user interaction. The results of the usability survey show that users perceive a system more useful when it follows the ADUS architecture.

As future work we are considering some options for improving the exploitation of user interaction data stored by user agents and expanding user agents' ability to automatically recognize tasks from a wider range of GUI widgets.

Acknowledgements

This work was supported by the CICYT project TIN2004-07999-C02-02.

References

1. M. Abrams, C. Phanouriou, A.L. Batongbacal, S.M. William, and J.E. Shuster. Uiml: An appliance-independent xml user interface language. In *WWW8 / Computer Networks 31(11-16): 1695-1708*, 1999.
2. M.F. Barthet and J.C. Tarby. The diane+ method. In *Computer-aided design of user interfaces (pp. 95120). Namur, Belgium*, 1996.
3. E.H. 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.
4. WWW Consortium. http://www.w3.org/Mobile/CCPP/.
5. Microsoft Corp. Creating mobile web applications with mobile web forms in visual studio .net, 2001. http://msdn.microsoft.com/vstudio/technical/articles/mobilewebforms.asp.
6. M. Deshpande and G. Karypis. Selective markov models for predicting web-page accesses. Technical report, University of Minnesota Tech. Report 00-056, 2000.
7. E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 256–265, Madison, WI, July 1998.
8. IBM. Ibm websphere transcoding publisher, 2001. http://www3.ibm.com/software/webservers/transcoding/.
9. Recursion Software Inc., 2006. http://www.recursionsw.com/voyager.htm.
10. M.Y. Ivory, R.R. Sinha, , and M.A. Hearst. Empirically validated web page design metrics. In *SIGCHI*, 2001.
11. K. Coninx K. Lyten. An xml runtime user interface description language for mobile computing devices. In *8th DSV-IS Workshop*. Springer Verlag, 2001.
12. K.Y. Lim and J. Long. The muse method for usability engineering. In *Cambridge University Press, Cambridge*, 1994.
13. Q. Limbourg and J. Vanderdonckt. *Comparing Task Models for User Interface Design*. Lawrence Erlbaum Associates, 2003.
14. H. Liu, H. Lieberman, and T. Selker. A model of textual affect sensing using real-world knowledge. In 2003 *Int. Conference on Intelligent UIs*, January 2003.
15. E. Mena, A. Illarramendi, J.A. Royo, and A. Goni. A software retrieval service based on adaptive knowledge-driven agents for wireless environments. *Transactions on Autonomous and Adaptive Systems (TAAS)*, 1(1), September 2006.
16. D. Milojicic. Mobile agent applications. *IEEE Concurrency*, 7(3):80–90, 1999.
17. D. Milojicic, M. Breugst, I. Busse,J.Campbell, S. Covaci,B.Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. MASIF, the OMG mobile agent system interoperability facility. In *Proceedings of Mobile Agents*, volume 1477. LNAI, September 1998.

18.N. Mitrovic and E. Mena. Adaptive user interface for mobile devices. In *Interactive Systems. Design, Specification, and Verification. 9th International Workshop DSVIS 2002, Rostock (Germany)*. Springer Verlag LNCS, June 2002.
19.N. Mitrovic and E. Mena. Improving user interface usability using mobile agents. In *10th DSV-IS Workshop, Funchal, Madeira Island (Portugal)*. Springer Verlag LNCS 2844, June 2003.
20.N. Mitrovic, J.A. Royo, and E. Mena. Adus: Indirect generation of user interfaces on wireless devices. In *15th Int. Workshop on Database and Expert Systems Applications (DEXA'2004), 7th Int. Workshop Mobility in Databases and Distributed Systems (MDDS'2004)*. IEEE Computer Society, September 2004.
21.N. Mitrovic, J.A. Royo, and E. Mena. Adaptive user interfaces based on mobile agents: Monitoring the behavior of users in a wireless environment. In *I Symposium on Ubiquitous Computing and Ambient Intelligence, Spain*. Thomson-Paraninfo, 2005.
22.J.P. Molina, S. Melia, and O. Pastor. Just-ui: A user interface specification model. In *4th International Conference on Computer-Aided Design of User Interfaces CADUI 2002*. Kluwer, 2002.
23.F. Paterno and C. Santoro. One model, many interfaces. In *Fourth International Conference on Computer-Aided Design of User Interfaces (CADUI 2002)*.Kluwer Academics, 2002.
24.J. Pitkow and P. Pirolli. Mining longest repeatable subsequences to predict world wide web surfing. In *2nd Usenix Symposium on Internet Technologies and Systems (USITS)*, 1999.
25.Young-Woo Seo and Byoung-Tak Zhang. Learning user's preferences by analyzing web-browsing behaviors, *Int. Conf. on Autonomous Agents 2000*, 2000.
26.A. Shepherd. D. diaper (ed.) analysis and training in information technology tasks, chicester. In *Task analysis for human-computer interaction*, 1989.
27.H. Stottner. A platform-independent user interface description language, Technical Report 16, Institute for Practical Computer Science, Johannes Kepler University Linz,, 2001.
28.Eager system, 1993. http://www.acypher.com/wwid/Chapters/09Eager.html.
29.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.
30.XUL Tutorial, 2002. http://www.xulplanet.com/tutorials/xultu/.
31.usiXML, 2004. http://www.usixml.org/.
32.W3C. Xforms, 2000. http://www.xforms.org/.
33.A. Wang, C.-F. Srensen, and E. Indal. A mobile agent architecture for heterogeneous devices. In *Proc. of the Third IASTED International Conference on Wireless and Optical Communications (WOC 2003)*, 2003.
34.XIML, November 1999. http://www.ximl.org/.
35.Ingrid Zukerman and David Albrecht. Predictive statistical models for user modeling. In ed. Alfred Kobsa, editor, *User Modeling and User Adapted Interaction (UMUAI) -The Journal of Personalization Research*, volume Ten Aniversary Special Issue. Kluwer Academic Publishers, 2000.

Questions

Jose Campos:

Question: When you change from laptop to PDA you might need to change dialogue control, not only the screen layout. Are your agents capable of this?

Answer: This is an open problem and future work.