Improving User Interface Usability Using Mobile Agents*

Nikola Mitrović[1] and Eduardo Mena[2]

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

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

Abstract. Adapting graphical user interfaces (GUI) to meet higher level of usability for users is one of the most interesting questions of today's mobile computation. Users should have GUI constructed to meet their needs, habits and expectations. In this paper, we discuss existing solutions and present a solution based on mobile agents. Mobile agents 'learn' users' habits, cooperate with other agents and construct the GUI in order to meet the users' expectations. Mobile agents move from host to host and are able to 'learn' about GUI usability by observing multiple users using the GUI. In addition, mobile agents cooperate with *Personal Agents* in order to apply personalized changes to the GUI. The result is an adaptable GUI platform that strives to predict user behaviour and to be more usable. We show the application of this approach by implementing a simple business application.

1 Introduction

Adapting graphical user interfaces (GUIs) to meet usability is one of the most challenging questions in the user interfaces area. Main problems are raised from the fact that the usability is hard to measure and analyse, and that measured data are often not available to multiple instances of the program.

Solutions in this area mainly focus on web-site usage metrics [11], user behaviour prediction and simulation [2], or usability patterns [5], [3]. Collected metrics are often hard to analyse and implement. Collected data could be easily misinterpreted due to varied levels of user expertise or external factors that influence users (fatigue, distraction, etc.). Some solutions aim to collect metrics on web GUI usability [11] so the data could be used to analyse the usability, and more advanced approaches [2] try to predict user behaviour and to propose measures

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

that could increase GUI's usability. Usability patterns try to create general recommendations on how to construct GUIs to achieve better usability. All these solutions aim to provide an off-line analysis and not a run-time solution.

The idea of this work is to transparently predict user behaviour and to adapt accordingly graphical user interface by using mobile agent systems [12]. Agents are highly mobile, autonomous and intelligent. They can cooperate with other intelligent agents in order to exchange information and maximise performance.

In our previous work [1] we demonstrated the capability of mobile agents to autonomously adapt user interfaces to various resources and to collect various types of data. Agents can also collect usability metrics and autonomously decide how to adapt user interface for each user in order to improve usability. In contrast, solutions not using mobile agents often find run-time user interface adaptation and collection of usage metrics of multiple program instances difficult. With classic approach, new user interface (or program) updates lead to reinstalling client programs on every user device, which does not happen when using mobile agents.

Our prototype adapts user interface using mobile agents [12] that process user interface definition described in Extensible User-interface Language (XUL) [1], [15]. XUL interpretation to a standard Swing interface is done by jXUL platform [9]. Agents automatically adapt the interface definition to the clients' interface, making user interface dynamic and multiple interface implementations unnecessary.

The rest of this paper is as follows. Section 2 gives an overview of state of the art and the related work. In section 3 we give brief overview of Markov models, longest repeating subsequence, hybrid solutions and their evaluation. Section 4 introduces mobile agents and extensible user interface language (XUL) and gives an overview of user interface generation. In section 5 we introduce our motivating example and explain bound between mobile agents, GUI and prediction models. Section 6 describes sample scenario that shows the presented technique. Section 7 explains in detail the learning process. Section 8 concludes the paper and discusses the future work.

2 State of the Art and Related Work

Measuring user interfaces and predicting user behaviour is based on several concepts. We will focus on three main approaches: user interface metrics, data mining – user behaviour prediction and the usability patterns.

2.1 User Interface Metrics

The basic concept is to collect user interface metrics for a web site [11]. Usually, collected data are used to perform traffic-based analysis [14] (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. Similarly, the web pages are frequently checked against predefined guidance, e.g. if the images have

ALT tags or if the pages have titles. Some approaches [11] tend to empirically validate metrics against expert ratings (e.g. PC Magazine TOP 100 web sites).

From these methods one can get some of the web site properties: the usual time that users spend on the page, how many users navigated to some page, how many pages were visited by a single user. Furthermore, one can learn if the site is built properly, does it have ALT tags, page titles and if there are broken links. However, these methods fail to give prediction of user behaviour, and results that they give can be influenced by many factors.

2.2 Data Mining and User Behaviour Prediction

The following concepts provide concrete methods to predict and simulate user behaviour in order to test different designs.

Many models that treat to predict user behaviour are based on Markov chains [6], [7]. Predictions are made basing on the data from the usage logs. Generally, more complex, high-order Markov models give better predictions than simpler first-order models. This is because first-order models do not look enough into the history. Sometimes, even higher-order models do not provide good predictions. We will describe Markov models in more detail in section 3.1.

More advanced models, like Longest Repeating Subsequence (LRS) [2] or Information Scent [4] perform data mining seeking to analyse 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.

Presented models help in improving design and usability. But still, there is an evident lack of run-time support for systems that are being analysed, and authors aim to create design-time analysis tools like WUFIS [4] or WebCriteria SiteProfile [13] for web sites.

2.3 Usability Patterns

Usability patterns describe successful solutions to recurring design problems. In HCI community, usability patterns [3] are relatively new concept, although it is successfully implemented in Object-Oriented design and architecture. The idea is to document and share successful solutions that improve usability between designers. Usability patterns define common design patterns such as 'Web Toolbar', 'Contact Us', 'Site Map', etc. Implementing such patterns improves usability of user interfaces.

Porting usability patterns to different platforms and resources is not an easy task [3], documenting patterns [5] still remains an open issue.

3 Longest Repeating Subsequence Method and its modifications

Longest Repeating Subsequence (LRS) is as we described earlier, one of the methods that try to predict user behaviour. This method was developed by

Pitkow and Pirolli, and published in [2]. The method can be seen as evolution of Markov methods for predicting user behaviour [6], [7] and the Path Profiles method developed by Schechter, Krishnan and Smith [25].

We have decided to use this method because in comparison with Markov models it endorses simplicity while retaining prediction rates of Markov models.

In the following paragraphs, we will describe briefly this methodology.

3.1 Markov models

Markov models have been used for studying and understanding stochastic processes, and therefore for predicting users' behaviour while browsing web sites [6]. Sequences obtained from web site logs are used to predict what page is most likely to be accessed next by the user.

The simplest Markov model is *first-order* Markov model [7]. This model predicts next user action by only looking to the last performed action. Evolution of this model leads to looking the last two performed actions – *second-order* model, and that leads to generalised form: K^{th}*-order* model [6].

3.2 Longest Repeating Subsequence - LRS

A longest repeating subsequence [2], [29] is a longest repeating sequence of items where the number of consecutive items repeats more than some threshold T (T usually equals one). To help illustrate, suppose that we have a web site containing pages A, B, C and D, where A contains link to B, and B contains links both to C and D (Fig. 1).

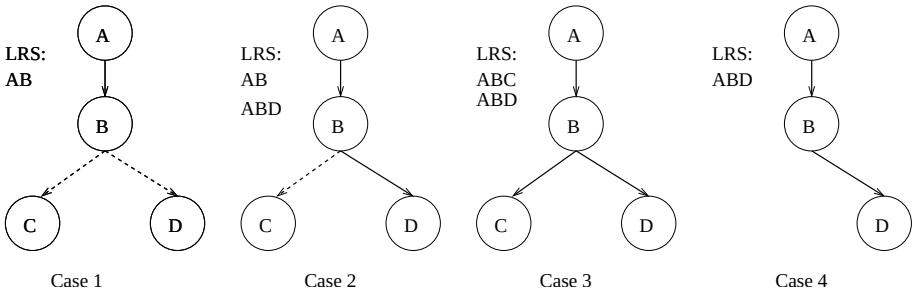

Fig. 1. Formation of longest repeating subsequence (LRS) example

If we suppose that all users go from page A to page B, and one user navigates to C and one to D (as in Case 1, Fig. 1), the longest repeating subsequence (LRS) will be AB. If more than one user navigates to page D, then the longest repeating subsequence will be AB and ABD, as in Case 2. In Case 3, both ABC and ABD are LRS, since both page C and D were visited more than once – AB is no longer LRS. Similarly in Case 4, only the page D is visited.

LRS has several interesting properties. First, low probability transitions are automatically excluded which in some cases will result in prediction not being made. For example, in Case 1, with threshold T=1, the penultimate match for LRS AB is A, and prediction after pages A and B will not be possible [2]. In addition, case of any single page-to-page transition is always repeated as part of a larger sequence is not included in LRS.

3.3 Hybrid LRS-Markov models

Pitkow et al propose two hybrid LRS-Markov models [2]. First, authors propose *one-hop LRS* model that consists of: extraction of LRS from the sample data, and then using these for estimating first-order Markov models. That is creation of one-hop n-grams [2] out of LRS, for example if the LRS is ABCD, the result would be: AB, BC, CD.

The second method is *All-K^{th}-Order LRS* model. This model decompresses extracted LRS subsequences to all possible n-grams. With this model we can predict for all orders of k.

Empirical evaluation of hybrid models and Markov models [2] shows that one-hop LRS model reduces the space necessary for storing models thus satisfying the complexity reduction principle. The prediction rate in Pitkow's experiments were very close which means that one-hop LRS model preserves predicting ability while reducing complexity. Comparison of All-K^{th} models gives similar results.

We have chosen to implement hybrid LRS-Markov models because we found it more suitable for our prototype. The prediction rates of hybrid models are sufficiently good while the complexity is reduced.

4 Generating User Interfaces with Mobile Agents

In our prototype we use eXtensible User Interface Language (XUL) and Mobile Agents in order to create user interface. We have chosen the same technology as we used in our previous work [1] since it enables us to create flexible user interface that is able to adapt and move trough network.

4.1 Extensible User-interface Language - XUL

Extensible User interface Language [1], [15], [9] is designed for cross-platform user interface definition. This language is incorporated in Mozilla project [16], acting as an user interface definition language. Being part of Mozilla project, XUL is open and connectable to other Mozilla projects. The format is organized with modern user interface definition in mind, supporting variety of available controls.

XUL lacks the abstraction layer of interface definition, and is restricted to window-based user interface. It is capable of referencing Cascading Style Sheets (CSS) [17] to define the layout of elements. User actions, property access and

functionality can be stored in JavaScript (ECMAscript) [24] files. Similar approaches include XIML [27] or UIML [22]. However, we found XUL to be suitable open source solution for our purpose.

4.2 Mobile Agents and Agent Platforms

A mobile agent [12], [10], [23] is a program that executes autonomously on a set of network hosts on behalf of an individual or organization. The agent visits the network hosts to execute parts of its program and may interact with other agents residing on that host or elsewhere [8], while working towards a goal. During their lifetime agents travel to different hosts that can have distinct user interface possibilities. Agents typically possess several (or all) of the following characteristics; they are:

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

They do not, by themselves, constitute a complete application. Instead, they form one by working in conjunction with an agent host and other agents. Many agents are meant to be used as intelligent electronic gophers – automated errand boys. Tell them what you want them to do – search the Internet for information on a topic, or assemble and order a computer according to your desired specifications – and they will do it and let you know when they have finished. Some agents are used as *Personal Agents* that store user preferences, certificates, policies or perform actions on the behalf of the user (e.g. enforcing security policies [8]). Mobile Agent Systems (MAS) are the middleware that allows creating and executing mobile agents. For this project, we choose Grasshopper [18] as the most intuitive and stable mobile agent platform, which supports standards such as FIPA [19], CORBA [20] and RMI [21]. In addition, the Grasshopper's feature Webhopper [18] that enables mobile agents for web is a significant plus comparing with other platforms, like Voyager and Aglets [26].

5 Using Mobile Agents to Improve Usability

In our previous work [1], we demonstrated possibilities and benefits of adapting user interfaces to various user devices using mobile agents. The idea of this paper

was to extend our previous work and to build the prototype that adapts user interface aiming to improved usability.

Other approaches (see [4], [14], [2], [6]) try to gather user interfaces metrics, and then to analyse these data in order to redesign user interface to meet higher usability. Although tools that help analysing these data exist (see [4], [14]), this process is manual and performed on off-line data. After applying new design to user interface, applications should be deployed again in order to have new version of user interface. Our leading idea was to design and build a system that will enable real-time analysis of such data and that will try to improve usability by trying to predict the next most probable user action. The system should be able to 'push' new versions of user interface to users as well.

Mobile Agents are particularly suitable for adapting user interfaces and learning [1]. Agents are autonomous, communicative, they work towards their goal, and can decide of their actions based on the environment and external factors [10]. Mobile agents endorse 'push' technology – agents can travel to any host or user without prior invitation. They can provide transparent resolution of many environment errors (e.g. network errors).

In our prototype we use mobile agents to create Swing interfaces, and we plan to use this technology to improve user interface usability for various user devices (e.g. HTML clients, WAP clients, etc.). We created specialised mobile agents that learn user behaviour. These agents examine usage data in order to predict next probable user action. They exchange data, learn from user actions and keep in mind user's preferences. Furthermore, re-designed user interface could be pushed to users (using mobile agents) at any time since GUI designers can also learn from the usage data.

We present a sample application for invoice composition and manipulation. The application has basic options, such as opening, saving, closing and printing an invoice, adding items and taxes and selecting a customer. Sample application is mobile and it communicates with other agents and application instances. User interface is adapted to user's preferences and the application has interactive help that guides users to achieve goals. We will describe the sample application in detail in section 6.

5.1 Implemented Technology

We built a prototype that adapts user interfaces to user needs in transparent manner to both designer and user. We extended eXtensible User Interface language to support demarcation of design patterns. Using extended tags we are able to determine content and position of different design patterns within the user interface definition and therefore to adapt user interface according to user's preferences.

In order to enhance their applications with the functionality of the prototyped agents, the developers should only extend the appropriate class, connect the classes and user interface definition files should be created as described previously in [1].

In addition, we implemented All-K^{th}-order LRS model and adjusted it for run-time use with applications based on mobile agents. We use more complex hybrid LRS-Markov model in order to give better predictions of user behaviour.

5.2 Specialised Agents

Applications based on mobile agents typically consist of several agents that perform different tasks. These agents are specialised to perform these tasks, and contain expert knowledge on how to achieve their goals. Agents learn and react to their environment and autonomously provide functionality to the system, application or other agents.

We have created several specialised agents that work together and help the adaptation and learning process:

- *User Interface Agent*: this agent was developed in our previous work [1], and it serves as a bridge between user and mobile agents. This agent is capable of transforming user interfaces to meet capabilities of various user devices, and it uses XUL as user interface definition language. This agent is fully extendible and connectable to other agents.
- *Helper Agent*: using this agent, our prototype is able to learn. Helper analyses data that is being collected from all users and suggests the next most probable action.
- *Wanderer Agent*: this agent is a specialised agent that wanders trough all clients and collects usage data. Its goal is to exchange data between all Helper agents.
- *Personal Agent*: users can store their preferences (e.g. font sizes, colours, etc.) with this agent. Additionally, this agent can store other relevant data: accessibility preferences, preferences on usability patterns or user certificates.

In Fig. 2 we can observe the composition of the sample application and its network topology. While User Interface, Personal and Helper Agent are mainly static, Wanderer agent travels through network and distributes usage data and updates. Functionality of these agents will be described in following sections.

6 Sample Scenario

In our sample scenario, Invoicing application is an application based on mobile agents and is retrieved from network, from the nearest host. In Fig. 3 we can see that sample application has several windows:

- Main window: from this window, user can execute some of the options, e.g. 'new invoice', 'open invoice', etc.
- Invoice window: whether invoice is new or loaded from the database, this window fits into the main window and gives additional options, e.g. 'save invoice', 'print invoice', 'edit items', etc.

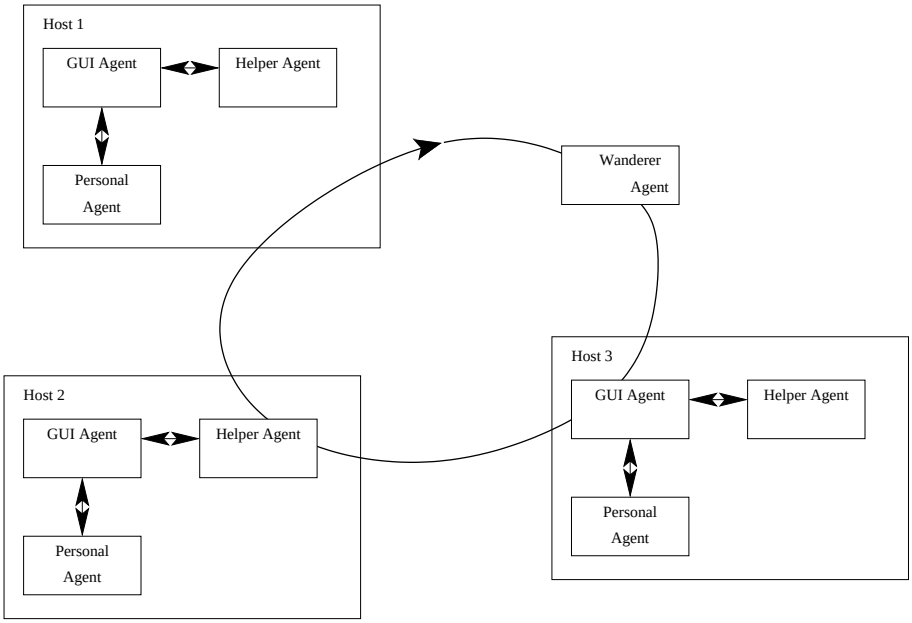

Fig. 2. Sample application structure

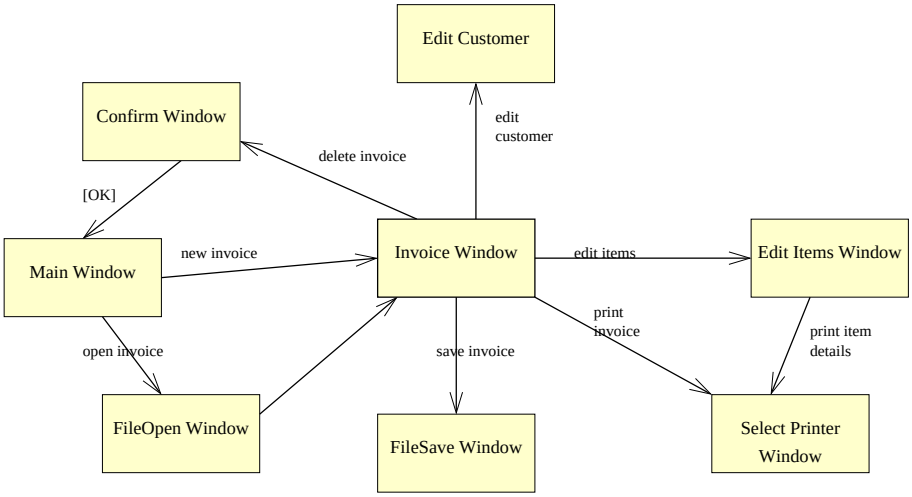

Fig. 3. Sample application structure

– Edit items window: this window enables user to add or remove items from invoice or to print item details

Mobile agents (in our case, specialised agents) autonomously collect usage data from all users and 'learn' user habits using All-K^{th}-order LRS model.

Helper agents then predict the next action to be performed by the user, and display available actions in order of probability in the specialised toolbar (see Fig. 4). Additionally, agents cooperate and adjust user interface to suite user's preference.

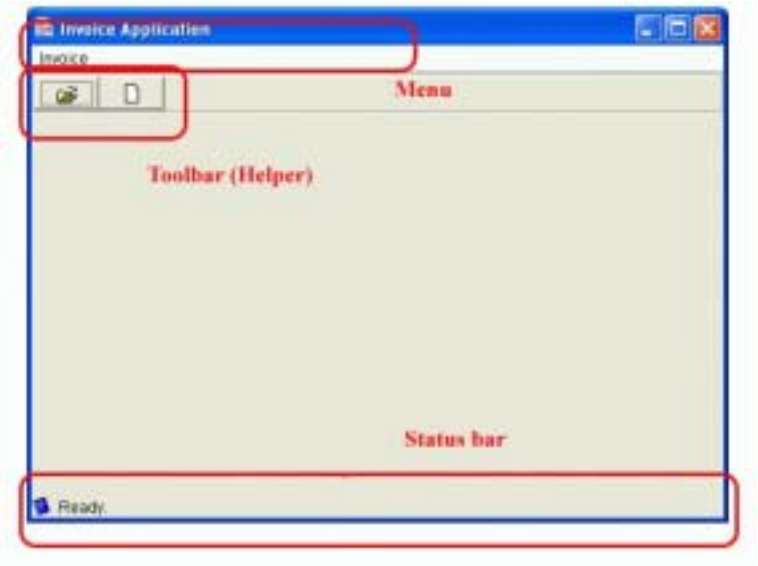

Fig. 4. Main window: basic layout and patterns.

In Fig. 4 we can observe that user interface consists of several design patterns: Menu, Status bar and Toolbar. As we mentioned earlier, this toolbar (*Helper toolbar*) serves as front-end of the Helper Agent to users, giving a hint of the next most probable action. Action that is predicted as the most probable next action will be the first action in the Helper toolbar, and the last one will be the action with least chances to be selected by the user. User can click the icon that appears on the Helper toolbar to perform the action. We decided to use Helper toolbar to display prediction instead of modifying the user interface itself. Modifying the user interface in run-time could be very confusing for the users.

7 The Learning Process

Learning is implemented in specialised Helper Agent. This agent relies on All-K^{th}-order LRS model. If the application did not have training, prediction cannot be made. Once users start using the application, the Helper Agent collects the necessary data to try to predict the next hop. However, learning is not limited just to data collected by the Helper agent from a single user. The Wanderer Agent collects the usage data from all Helper agents, and creates one unique

usage log that is distributed to all Helper agents for analysis. Wanderer acts as a push service, and it is completely autonomous. This means that Wanderer agents chooses what should be the next host to visit, what data are relevant or not and who should receive updated data and when.

This process could lead to less efficient predictions, because of different user experiences and expertise. However, this could also help in increasing overall user expertise, since users with more expertise could influence predictions by supplying better sequences. Therefore, the Helper Agent would provide better predictions for users with less expertise and show them how to use application more efficiently.

In Fig. 3 we can observe that Select Printer window can be reached both from Edit Items window (in order to print item details) and Invoice window (in order to print invoice). In the Invoicing application, Select Printer window serves for generating requested report and sending it to the desired printing device. Therefore, if we offer to user more than next-hop prediction, we could have a case in which on the Helper toolbar user is suggested to go from FileOpen window directly to Select Printer window. This would lead to application crash – because there is no sufficient input data to perform code in Select Printer window (i.e. what report to prepare and for what invoice or item).

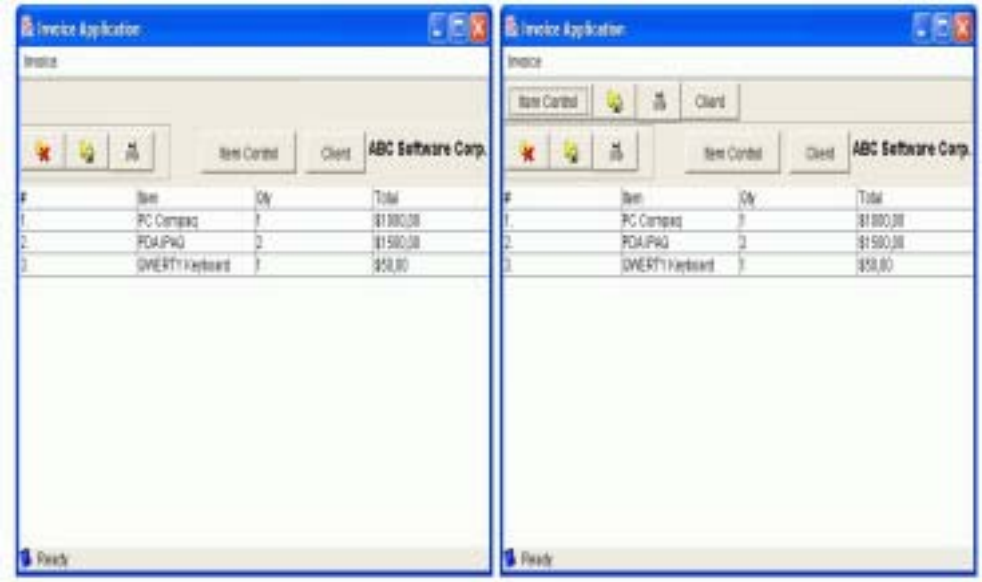

Fig. 5. Helper Agent predicts next action

7.1 Interaction between specialised agents

Here is how the training process goes. When the application is created, it has no usage data. Helper agent's repository and Wanderer's repository are empty. In Fig. 6 you can see interaction and processes map of the application. In this case (Fig. 5, left window) the Helper toolbar should be empty, as no prediction can be made. It is expected that the application designer will provide some basic training to Helper agent so the application could have some initial predictions.

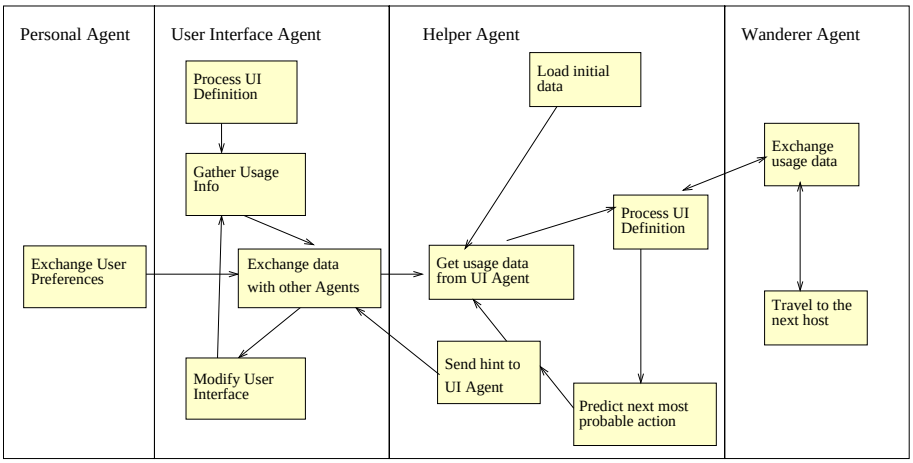

Fig. 6. Agent processes and interaction

When the application is requested from one of the hosts, the Wanderer agent is informed that one of the instances of application is about to depart. Wanderer registers this instance (so it can be visited or updated later), and loads the initial snapshot of usage data to Helper Agent. After being loaded with data, application moves to the destination host, and starts.

While in use, application collects usage data and learns. In Fig. 5 (on the right) we can see that the Helper toolbar displays most probable predictions. Occasionally, the application is visited by the Wanderer agent. The Helper agent and Wanderer agent exchange usage data, and the Helper agent's experience gets richer. The Helper toolbar is dynamically updated with new predictions. Classic desktop applications would probably base its learning on the current user's data only: we use the benefit of mobile agent's push feature to expand our training data. Wanderer agent is pushed to the user and not requested by the user or application. In addition, Wanderer agent can also update the user interface definition if the user interface has been redesigned to improve usability. After the Helper agent was updated with the latest data snapshot, it continues to operate normally – fresh usage data are collected and mixed with the snapshot of all users' data. Learning process continues.

We can also observe from the Fig. 5 that one of the options ('delete invoice') will not be suggested, since it has never been used by the users in our example. Other actions have been ordered by probability of occurrence: item control, save invoice, print invoice and client control. Training period of Helper Agents never stops.

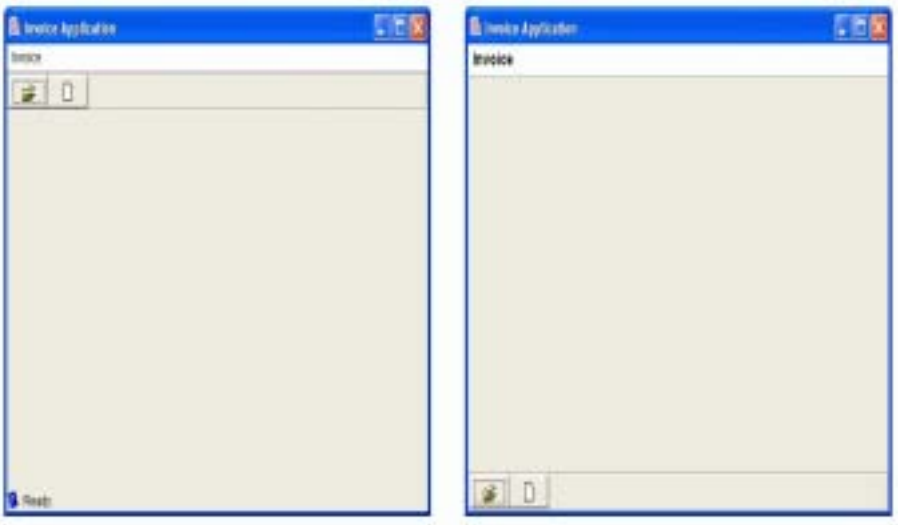

Fig. 7. Helper Agent adapts user interface to meet user's preferences

7.2 Adapting to personal preferences with Personal Agents

One of the features of Helper Agent is to communicate with user's Personal Agent (see Fig. 6). Personal Agent stores user's preferences and supplies information about user's habits to the Helper Agent. We can see in Fig. 7 how can user interface change according to user's preferences. This type of adaptation focuses on usability patterns and user interface accessibility problems. In Fig. 7 we can see that one of the users has following preferences: toolbars on the bottom of the windows, slightly larger fonts and no status bar.

Helper Agent has adapted transparently user interface to the user's preferences and increased usability for that particular user. This enables users to use the application more efficiently as the user saves time in customising and using the application.

8 Conclusions and Future Work

In this paper we have presented autonomous and intelligent system based on mobile agents that transparently adapts user interface. We have constructed

specialised agents that predict user behaviour and suggest actions to users in real-time.

The main features of this approach are:

- We use and extend our previous work, a system for adapting user interfaces to various resources by using mobile agents.
- Specialised agents have been built:
  - Helper Agent: predicts the next action using All-K^{th}-order LRS model and use predictions to improve application usability
  - Wanderer Agent: exchanges data between application users, implements *push* service and is capable of pushing the re-designed user interface to all users.
  - Personal Agent: stores all user's preferences and cooperates with other agents in order to apply these preferences.
- User interface is modified at run-time.
- Usage data are collected on all users, and there is no training time limit.

Our future work will be focused on:

- Adapting of this concept to various resources (HTML, WAP, etc.). The potential problems, as we discussed, are in adapting usability patterns to various resources, and achieving satisfactory plasticity [28].
- Using task models in order to improve hybrid LRS-Markov model.
- Reduction of the space necessary for storing sequences.
- Definition of in-window tasks so the system could predict tasks within one window.
- Measuring usability using users of various expertise levels.

References

1. Mitrović, N., Mena, E.: "Adaptive User Interface for Mobile Devices", 9th International Workshop on Design, Specification and Verification (DSV-IS) 2002, Springer Verlag Lecture Notes in Computer Science vol. 2545, pages 29-44.
2. Pitkow, J., Pirolli, P.: "Mining Longest Repeatable Subsequences to Predict World Wide Web surfing", 2nd Usenix Symposium on Internet Technologies and Systems (USITS), 1999.
3. Seffah, A., Forbrig, P.: "Multiple User Interfaces: Towards a Task-Driven and Patterns-Oriented Design Model", 9th International Workshop on Design, Specification and Verification (DSV-IS) 2002, Springer Verlag Lecture Notes in Computer Science vol. 2545, pages 118-133.
4. Chi, E.H., Pirolli, P., Pitkow, J.: The scent of a site: "A system for analyzing and predicting information scent, usage, and usability of a web site", ACM CHI 00 Conference on Human Factors in Computing Systems, 2000.
5. Seffah, A., Javahery, H.: "On the Usability of Usability Patterns - What can make patterns usable and accessible for common developers", Workshop on Patterns in Practice, ACM CHI Conference, Mineapolis, Minnesota, April 2002.
6. Deshpande, M., Karypis, G.: "Selective Markov Models for Predicting Web-Page Accesses", University of Minnesota Technical Report 00-056, 2000.

7. Griffioen, J., Appleton, R.: "Reducing file system latency using predictive approach", USENIX Technical Conference Cambridge, 1994.
8. Mitrović, N., Arronategui, U.: "Mobile Agent security using Proxy-agents and Trusted Domains", 2nd International Workshop of Security of Multiagent Systems (SEMAS '02) at 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 02), DFKI Research Report, 2002, pages 81-84
9. jXUL, http://jxul.sourceforge.net
10. Etzioni, O., Weld, D.: "Intelligent Agents on the Internet: Fact, Fiction, and Forecast", IEEE Expert vol. 10, 1995, pages 44-49
11. Ivory, M.Y., Sinha, R.R., Hearst, M.A.: "Empirically Validated Web Page Design Metrics", SIGCHI 2001.
12. Distributed Objects & Components: Mobile Agents, http://www.cetus-links.org/oo_mobile_agents.html
13. WebCriteria. Max, and the objective measurement of web sites. http://www.webcriteria.com, 1999.
14. Tiedke, T., Martin, C., Gerth, N.: "AWUSA – A Tool for Automated Website Usability Analysis", in proceedings of 9th International Workshop on Design, Specification and Verification DSV-IS, 2002.
15. Cheng, T.: XUL - Creating Localizable XML GUI, Fifteenth Unicode Conference, 1999. http://www.mozilla.org/projects/intl/iuc15/paper/iuc15xul.html
16. Mozilla project, http://www.mozilla.org
17. Meyer, E. A.:"Cascading Style Sheets: The Definitive Guide", O'Reilly and Associates, 2000.
18. Grasshopper, IKV, http://www.grasshopper.de/
19. Foundation for Intelligent Physical Agents, http://www.fipa.org
20. Pope, A.: "The CORBA Reference Guide: Understanding the Common Object Request Broker Architecture", Addison-Wesley Pub Co, 1998.
21. Java Remote Method Invocation, http://java.sun.com/products/jdk/rmi/
22. Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S.M., Shuster, J.E.: "UIML: An Appliance-Independent XML User Interface Language", WWW8 / Computer Networks 31(11-16): 1695-1708 (1999)
23. Milojicic, D.S.: "Trend Wars: Mobile agent applications", IEEE Concurrency 7(3), 1999, pages 80-90
24. ECMAScript Language Specification, 3rd Edition, December 1999, ECMA, http://www.ecma.ch/ecma1/stand/ecma-262.htm
25. Schechter, S., Krishnan, M., Smith, M.D.: "Using path profiles to predict HTTP requests", Seventh International World Wide Web Conference, 1998.
26. The Mobile Agent List, University of Stuttgart, http://mole.informatik.uni-stuttgart.de/mal/mal.html
27. XIML (eXtensible Interface Markup Language), http://www.ximl.org/
28. Thevenin, D. and Coutaz, J.: "Plasticity of User Interfaces: Frame-work and Research Agenda", Proc of IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'99, Edinburgh, August 1999, IOS Press, 1999.
29. Crow, D., Smith, B.: "DB Habits: Comparing minimal knowledge and knowledge-based approaches to pattern recognition in the domain of user-computer interactions", Neural networks and pattern recognition in human-computer interaction, 1992, pages 39-63