Mobile Access to Real-Time Information – The Case of Autonomous Stock Brokering

Stina Nylander, Markus Bylund, Magnus Boman
Swedish Institute of Computer Science
16 October 2003
E-mail: {stina.nylander, markus.bylund, magnus.boman}@sics.se
SICS Technical Report T2003:18
ISSN 1100-3154
ISRN:SICS-T--2003/18-SE

Keywords: real-time information, mobile services, information push, the Ubiquitous Interactor, device independence.

Abstract

If services providing real-time information are accessible from mobile devices, functionality is often restricted and no adaptation of the user interface to the mobile device is attempted. Mobile access to real-time information requires designs for multi-device access and automated facilities for adaptation of user interfaces. We present TapBroker, a push update service that provides mobile and stationary access to information on autonomous agents trading stocks. TapBroker is developed for the Ubiquitous Interactor system and is accessible from Java Swing user interfaces and Web user interfaces on desktop computers, and from a Java Awt user interface on mobile phones. New user interfaces can easily be added without changes in the service logic.

1 Introduction

Users of real-time services often encounter problems with persistent control when they leave their desktop computer. For instance, professional stock traders would like to bring their desktop environment on a laptop, PDA, or cellular phone to meetings as well as to the coffee room (Blomberg, 2001). Non-professional investors, i.e. people that actively manage their savings, would also welcome services for being

kept up to date outside the home. Unfortunately, this is seldom possible. In many cases, mobile users lose the service altogether, and in the few cases when it is possible to access the service from a mobile device, the functionality is restricted, or the adaptation to the device is insufficient. The prevailing method for making services available from various devices is by making different versions, as in the case of Web services for WAP. This quickly becomes infeasible as the range of devices grows wider, or when extensive personal customization is necessary. Development and maintenance work becomes difficult, and version differences increase the risk of error and inconsistencies.

To accommodate real-time mobile access, services have to be designed for multi-device access, and automated facilities for adaptation of user interfaces to different devices should be provided. Our case in point is *TapBroker*, a mobile and stationary push update service, aimed at providing users with notifications on autonomous agents (Maes, 1994), trading on a financial exchange. Trading agents code the preferences of their owners, in our case limited to stock portfolios. While agent trading in theory relaxes the agent owner from monitoring market data, in practice there will be long periods of intense trading, radical market changes, preference changes, and other factors that will contribute to the need for full control. Agent owners will carry with them persistent trading services, and will expect at all times at least one channel of swift and consistent interaction means. In cooperation with OM, the world's largest provider of software for financial exchanges (Sales, 2001), we have implemented a so-called Agent Trade Server (ATS) (Lybäck and Boman, 2003). In addition to our proof-of-concept implementation, we have implemented agents of varying levels of sophistication, and also various services for agent interaction with their owners.

It might be argued that the best and easiest way to create services with multiple user interfaces is to use Web user interfaces. Most devices are capable of running a WWW browser and could thus support WWW interaction. However, Web interaction has several drawbacks. It is user-driven, and even if there are ways to get around that, it is unsuitable for services relying on push data, like TapBroker. It is also difficult to control how Web user interfaces are presented to the end-user, cf. (Esler et al., 1999).

We will use the Ubiquitous Interactor (Nylander and Bylund, 2002), which supports information push and the development of services that present themselves differently on different devices. This is done by

separating user-service interaction from presentation. User-service interaction is kept the same, and information about how a user interface should be generated on different devices is provided separately. Services can be created once and presentation information for new devices can be specified at any time without causing any changes in the service logic. The possibility to push information from the service to the user interface allows for service-driven interaction, which is useful in other mobile and context-aware applications (Cheverst et al., 2002).

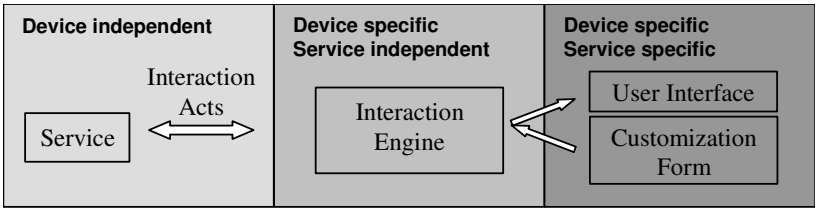

Figure 1: The three layers of specification in the Ubiquitous Interactor. Services and interaction acts are device independent, interaction engines are service independent and device or user interface specific. Customzation forms and generated user interfaces are device and service specific.

The following section briefly describes the Ubiquitous Interactor. Section 3 details the TapBroker service, the user interface aspects of which are discussed in Section 4.

2 The Ubiquitous Interactor

The Ubiquitous Interactor (UBI) (Nylander and Bylund, 2002) is a system for developing device independent services. It has three main parts (see Figure 1): interaction acts, customization forms, and interaction engines. In brief, services offer functionality to users encoded in interaction acts, customization forms contain information about how a given service should be presented on a given device, and finally interaction engines generate user interfaces based on the interaction acts and customization forms. This architecture makes it possible to develop a service for an open set of devices, and to add customization forms for new devices whenever desired.

The issue of device independence has been addressed during different periods of computer history. During the early eighties, attempts were made to overcome differences in hardware standard (Olsen, 1987, Wiecha et al., 1990). More recently, the issue has been approached in

both the mobile research community (Olsen et al., 2000, Abrams et al., 1999) and in that of universal access (Stephanidis, 2001). However, none of the mentioned systems has been able to address the three main problems device independence, control of presentation, and information push in a satisfying way.

2.1 Interaction Acts

An interaction act is an abstract unit of user-service interaction that contains no information about presentation at all. The UBI concept builds on the assumption that user-service interaction for a wide range of services and devices can be captured with a small set of interaction acts in different combinations. The UBI supports eight different interaction acts: `start`, `stop`, `create`, `destroy`, `input`, `output`, `select`, and `modify`. `start` and `stop` refer to the starting and stopping of services. `create` and `destroy` refer to creation and deletion of service specific objects, `input` is input to the service, `output` is output to the user, `select` is selection from a set of alternatives, and `modify` is modification of service specific objects. `input` is mainly used for data not stored in the service, such as data for navigation operations. `select` is used for similar cases where the range of input is limited to a few alternatives. `create`, `destroy`, and `modify` are mainly used for application-specific data that users can manipulate, for example meetings in a calendar, or avatars in a game.

2.2 Customization Forms

Customization forms provide means for service providers to specify how a service should be presented to end-users. Control of presentation is an important issue in commercial development (Myers et al., 2000) since it is used, for example, for branding. By providing a detailed customization form, service providers have full control over how the user interface will be generated. Customization forms are optional. If there is no customization form, or if the form is not exhaustive, defaults are used to generate the user interface. The main forms of presentation information in a customization form are mappings and media resources. Mappings are links between interaction acts and widgets or other user interface components. Media resources are links to pictures, sounds, or other resources that a particular user interface might need to present an interaction act.

2.3 Interaction Engines

Interaction engines are specific to a device, to a family of devices, and to a type of user interface, but they are service independent. For example, an interaction engine for HTML user interfaces could be used on both desktop and laptop computers, while handheld computers would need a special engine. Each device used for accessing a UBI service needs an interaction engine installed. In the ideal case, devices would be delivered with interaction engines pre-installed. Devices that can handle several types of user interfaces can have several interaction engines installed. For example, a desktop computer can have interaction engines for both Java Swing user interfaces and Web user interfaces. During user-service interaction, interaction engines interpret interaction acts and customization forms (when they are available) and generate user interfaces for services. Interaction engines are also responsible for interpreting user actions and for sending them back to services and update user interfaces. User interfaces can be updated both on initiative from services and as a result of user action.

3 Implementation

The Ubiquitous Interactor is a working prototype with several interaction engines that handles the full set of interaction acts. Interaction acts are in turn encoded using the Interaction Specification Language (ISL) (Nylander and Bylund, 2002) , which is XML compliant. Each interaction act has a unique id, a symbolic name, a life cycle value, a modality, an information holder, and a possibility to carry metadata. Customization forms are also encoded in XML. DTDs for ISL and customization forms can be found in appendix A-C.

Each interaction engine contains modules for parsing ISL and customization forms, as well as generating responses to services from user actions. In an earlier project (Nylander and Bylund, 2002), we have implemented interaction engines for Java Swing, HTML, and Tcl/Tk. All three engines can create user interfaces for desktop or laptop computers. However, the default renderings for the Tcl/Tk interaction engine are most suitable for PDA user interfaces. We have also implemented a calendar service as a sample service (Nylander and Bylund, 2002). It provides basic calendar functions as entering, editing and deleting information, navigating the information, and displaying different views of the information. The calendar service has customization forms for all three interaction engines. Both interaction engines as well as the calendar

service are implemented as services in the sView system (Bylund, 2001, Bylund and Espinoza, 2000), to take advantage of the user interface handling features in sView. An sView service is a collection of java class definitions and resources packaged in a standard jar-file that can be loaded and executed in sView (cf. `sview.sics.se`).

4 The Tapbroker Service

The TapBroker notification service gives users feedback on the actions of their autonomous trading agents running on the Agent Trade Server (ATS). Due to security constraints, agents cannot be accessed from outside the ATS during run-time. The TapBroker has access to the XML-format agent logs and can thus provide information about the actions of agents. Users register their agents with TapBroker, which connects to the ATS and subscribes to the relevant log data. The primary source of input data to the TapBroker is thus the agent log. The ATS is pushing log data to the TapBroker, which in turn updates the user interface. This means that most of the changes in the user interface will not be user-driven but service-driven. This is important for a service like TapBroker. Agents buy and sell stocks with users' money, and it is very important that users get updated information every time they access the service. This cannot depend on users refreshing the user interface.

TapBroker gives feedback on the actions of the agents and also some information about their state. The service shows the transactions performed by agents, the content of agents' stock portfolios, the amount of money that agents have available, and the value of their portfolios. The current intentions of agents are shown through the active buy or sell orders. It also shows the agent state (active or shut down) and computes an activity level based on the number of transactions performed during the last 30 minutes. Users can switch between agents, and delete and register new agents to the service. They can also choose the amount of transactions to be shown: those made in the last day, the last week, the last month, or a complete transaction list. Since agents currently do not log the reasons or motivations for their actions, TapBroker cannot give any explanations to agent transactions.

The service can handle multiple agents and can be accessed from three different types of user interfaces: an HTML user interface via a Web browser, a Java Swing user interface from desktop or laptop computer, and a Java Awt user interface from an Ericsson P800 mobile phone. Besides server access, all one needs to run TapBroker is sView and the

Ubiquitous Interactor prototype, both publicly available (see `sview.sics.se`).

5 User Interface Design

The different user interfaces to the TapBroker service have been designed in several cycles, in which both stock traders and researchers were consulted. First, unstructured interviews were conducted with a small number of non-professional traders. They were asked to think about what feedback they would be interested in if they had agents trading on their behalf, in what way they wanted it presented, what devices they would be interested in using to access the information, and whether or not they were interested in mobile access to the information. They were also asked which kind of information that was necessary and which was optional. All traders agreed that the transactions were the most important information, whereas active orders, portfolio content, available amount of money, and portfolio value can be sacrificed in a user interface where screen space is scarce. We also conducted an interview with a professional trader, in order to informally investigate if there were any obvious differences in requirements, which indeed there were. In a professional setting, a motivation for each transaction of an agent, and an indicator of the level of performance for the agent were ranked as important. All traders agreed that mobile access was necessary. A first set of user interface sketches for desktop computer and PDA was created based on these interviews.

Second, the traders were asked to comment the first sketch of the user interfaces and suggest improvements. Preferences again differed between the professional trader and the non-professional traders. The professional traders preferred the sketches that showed the most information, and did not see window size as a problem. The non-professional traders preferred a small window that can be visible on the screen all the time, even if less information was presented. The most probable reason for this difference in preferences is that to professional traders, trading is a main activity and they perform many transactions every day. To non-professional traders, trading is a side activity and the number of performed transactions is low. In a sense, this difference will not persist in the case of trading agents. An agent will trade according to its strategy and the market conditions; regardless of how much attention it gets from its owner.

Third, two researchers in HCI were asked to comment the improved sketches of the user interface. They suggested a state indicator (showing

if the agent is active or shut down), and an indicator of activity level (showing the number of transactions per 30 minutes). Three of the resulting user interfaces can be viewed in figures 2 and 3.

The TapBroker service has customization forms for three different user interfaces: Java Swing, HTML, and Java Awt.

5.1 The Java Swing Customization Form

The Java Swing user interface provides all the available information from the agent. The main part of the user interface is devoted to performed transactions and portfolio content, including additional information about the stocks in the portfolio such as the current buy and sell price. The user interface also shows the agent's id, state and activity level, account information, and information about submitted orders. The state of the agent is showed as a play/paused icon, and the activity level as a progress bar. Users can switch between agents in the pull down menu "agent", and add or remove agents in the pull down menu "options", see Figure 2.

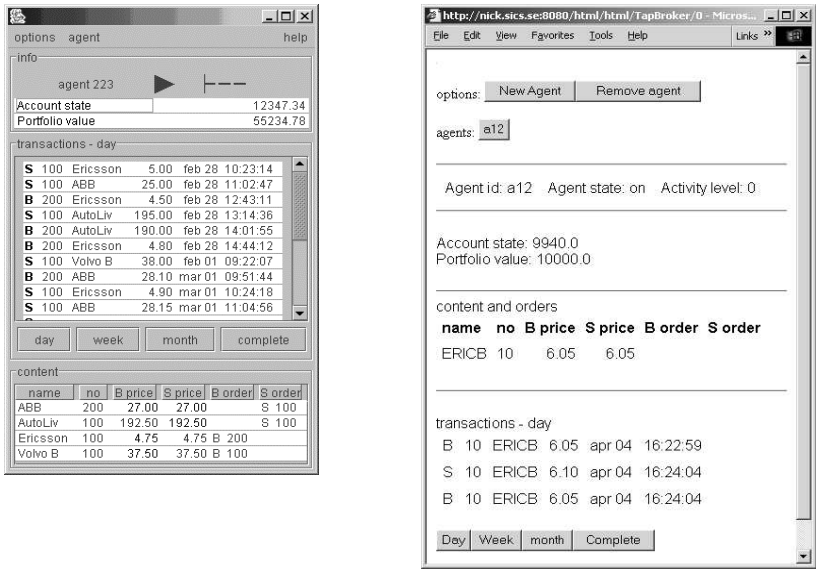

Figure 2: The Java Swing user interface (left) and the HTML user interface of the TapBroker service.

5.2 The HTML Customization Form

The HTML user interface provides the same information as the Java Swing user interface but presented in a slightly different way. The pull down menus for options and switching agents are presented as sets of buttons. Each operation is presented as a button instead of a menu option. The agent state and the activity level are displayed as text strings. The portfolio content is presented before the transactions to avoid excessive scrolling since the number of performed transactions is likely to be much larger than the number of different stocks in the portfolio, see Figure 2.

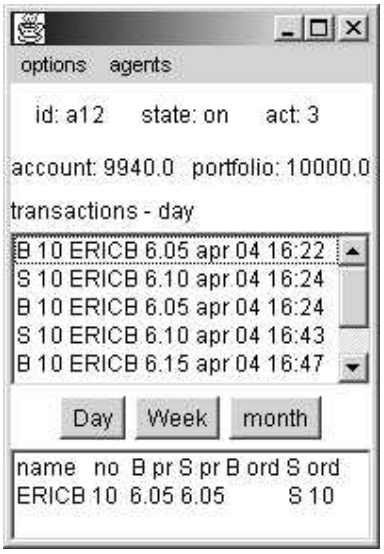

Figure 3: The Java Awt user interface of the TapBroker service.

5.3 The Java Awt Customization Form

The Java Awt user interface is designed to fit a small screen and is thus using shorter names for the different values presented, and sometimes also a shorter format on the value itself (for example no seconds shown in the time stamp of the transactions). The layout of the user interface is also more compact than in the other two, see Figure 3.

6 Conclusions

We have described TapBroker, an implemented service accommodating real-time mobile access, and facilitating automated adaptation of user interfaces to different devices. It provides continuously updated trading agent information by pushing it to the user. The service is available from different devices and presents itself with device specific user interfaces, using the push and the device independence features of the Ubiquitous Interactor system. TapBroker shows that it is possible to create device independent services that rely on information push, which is an important fact for example for the development of mobile context-aware applications. The TapBroker service will next be made available to the group that are currently developing agents for Agent Trade Server execution, see (Lybäck and Boman, 2003), to allow for further usability testing.

7 Acknowledgement

This work has been funded by the Swedish Agency of Innovation Systems through the projects TAP and ADAPT.

8 References

Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M. and Shuster, J. E. (1999) UIML - an appliance-independent XML user interface language, *Computer Networks,* **31**, 1695-1708.

Blomberg, J. (2001) *Narratives and Performance - the Case of Stock-broking*, SSE/EFI Working Papers series in Business Administration, 2001:2, Stockholm School of Economics.

Bylund, M. (2001) *Personal Service Environments - Openness and User Control in User-Service Interaction*, Licentiate thesis, Department of Information Technology, Uppsala University.

Bylund, M. and Espinoza, F. (2000) sView - Personal Service Interaction, in Proceedings of 5th International Conference on The Practical Applications of Intelligent Agents and Multi-Agent Technology.

Cheverst, K., Mitchell, K. and Davies, N. (2002) Exploring Context-aware Information Push, *Personal and Ubiquitous Computing,* **6,** (4), 276-281.

Esler, M., Hightower, J., Anderson, T. and Borriello, G. (1999) Next Century Challenges: Data-Centric Networking for Invisible Computing. The Portolano Project at the University of Washington, in Proceedings of The Fifth ACM International Conference on Mobile Computing and Networking, MobiCom 1999.

Lybäck, D. and Boman, M. (2003) Agent trade servers in financial exchange systems, *ACM Transactions on Internet Technology***,** (In press.).

Maes, P. (1994) Agents that Reduce Work and Information Overload, *Communications of the ACM,* **37,** (7), 31-40.

Myers, B. A., Hudson, S. E. and Pausch, R. (2000) Past, Present and Future of User Interface Software Tools, *ACM Transactions on Computer-Human Interaction,* **7,** (1), 3-28.

Nylander, S. and Bylund, M. (2002) Providing Device Independence to Mobile Services, in Proceedings of 7th ERCIM Workshop User Interfaces for All.

Olsen, D. J. (1987) MIKE: The Menu Interaction Kontrol Environment, *ACM Transactions on Graphics,* **5,** (4), 318-344.

Olsen, D. J., Jefferies, S., Nielsen, T., Moyes, W. and Fredrickson, P. (2000) Cross-modal Interaction using XWeb, in Proceedings of Symposium on User Interface Software and Technology, UIST 2000, 191-200.

Sales, R. (2001) First Europe, now the world, *Wall Street and Technology online*.

Stephanidis, C. (2001) The Concept of Unified User Interfaces, In *User Interfaces for All - Concepts, Methods, and Tools* (Ed, Stephanidis, C.) Lawrence Erlbaum Associates, pp. 371-388.

Wiecha, C., Bennett, W., Boies, S., Gould, J. and Greene, S. (1990) ITS: a Tool for Rapidly Developing Interactive Applications, *ACM Transactions on Information Systems,* **8,** (3), 204-236.