User Interface Design for Ubiquitous Computing W@PNotes, an Example

Andreas Zeidler
Databases and Distributed Systems Research Group
Wilhelminenstr. .7
64283 Darmstadt
+49 6151 16 6233
az@ubicomp.de

ABSTRACT
User interface (UI) design for ubiquitous computing is different from UI design for classical desktop-oriented applications. Interaction of a user with the system and applications takes place in a highly distributed fashion. Input- and output functionality is distributed over space, as well as time. Human-computer interaction (HCI) has to take place wherever needed and whenever needed. Due to the highly dynamic pattern of use, "classical" UI design for a single screen is poorly suited.

This paper presents an architecture for designing applications bringing different input- and output channels to small devices, like mobile phones, as well as taking advantage from the benefits of a desktop computer. The W@PNotes system serves as a proof of concept for the successful use of this architecture.

Keywords
User interface design, multiple input/output channels, function shipping, ubiquitous computing, Wireless Application Protocol.

INTRODUCTION

Obviously, in the vision of ubiquitous computing [1] "distribution"' of human-computer interaction is crucial. The user and her need to interact with her applications has to be in the center of UI design. Given this and the need for ubiquity, the term "distributed systems" must be defined for the distribution of system functionality as well as for the distribution of UI functionality. Apparently, the distribution of UI functionality needs new paradigms of design. UIs have to become "smart" and "intelligent". From a top-level point of view, two distribution schemes for UI functionality to actual devices seem to be appealing: (1) Use the input- and output devices available "here and now", including public displays and/or terminals, (2) try to take advantage of devices with some input-/output capabilities carried around by the user anyway, like mobile phones or personal digital assistants (PDA).

The W@PNotes-system presented here serves as an example for the second approach, although it incorporates some aspects of the first design dimension, too. The main goal of the system is (1) the design of smart and distributed user interfaces and (2) the underlying architecture, which is able to (a) separate and distribute system's functionality over several computers by means of service-oriented design using Jini [3] as an underlying distribution infrastructure and (b) bring advanced functionality to a very limited device, such as a mobile phone with an integrated browser for the Wireless Application Protocol (WAP) [6].

In the next section we will present the research goals of the W@PNotes project. After that, the architecture is described on a more technical level. Then, we will give some discussion of the achieved goals. Concluding, we will give a summary and an outlook.

RESEARCH GOALS

The W@PNotes project is an ongoing research project. The research goals of this project can be summarised as follows:

- Design and architecture of APIs for ubiquitous accessible applications.
- Design of UIs for ubiquitous accessible applications.
- Distribution of application functionality among various components and separation of concerns by using service oriented programming.
- "Good" composition of concerns, on the other hand.
- Distribution of customised HCI functionality among various UIs.
- Exploration of UI limitations with respect to device capabilities

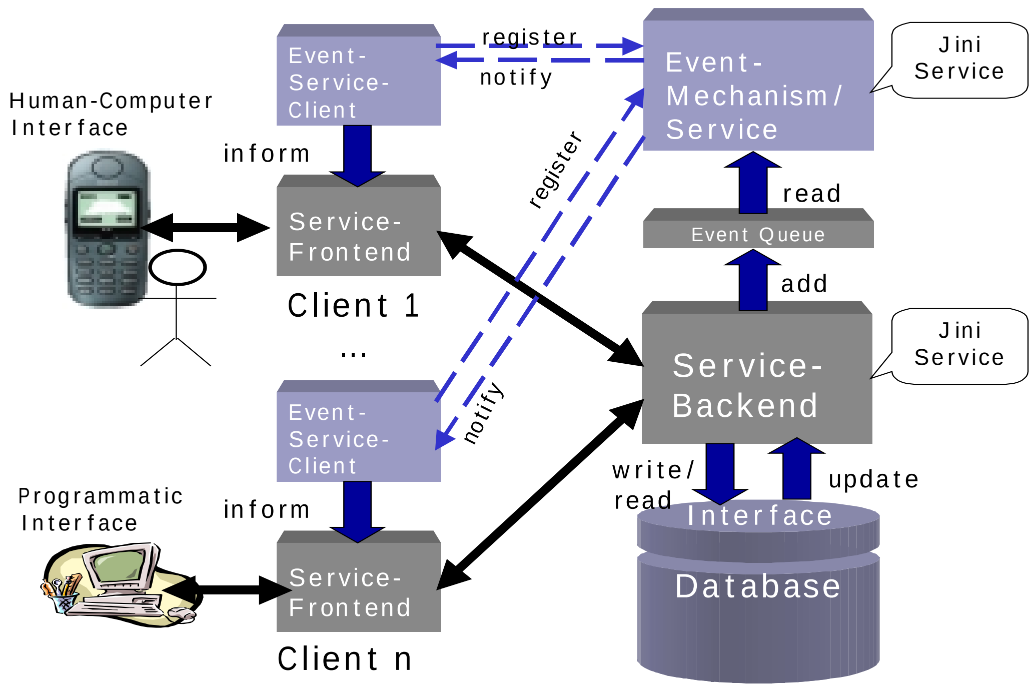

Figure 1. Top Level Architecture

The main research focus lies on the design patterns for designing distributed applications in an ubiquitous computing infrastructure. This includes the "how" of distribution by means of separation of concerns between different components (services) of this environment. On the other hand "composition of concerns", by means of a supporting framework for combining all those autonomous parts of a system in a reliable way, is as demanding.

Part of almost every application is a variety of user interfaces. Traditional application design interweaves the functional part of the application with the UI. Part of our research in this project is to separate application functionality completely from the input-/output channels. Moreover, UIs must be (1) distributed to the user, and (2) must match the capabilities of the device they are running on. It should be clear that not every UI is suitable for every device. We believe that the functionality offered to a user by the UI has to be adaptive with respect to the user and the device it is displayed on.

For example, a small device, like a mobile phone, typically has a very small display (5-6 lines of text, optionally b/w-graphics) and not much bandwidth to connect to a network. Therefore, it does not make much sense to display large amounts of information or a complex menu structure to the user. We feel that a configurable and reduced amount of output and a selected set of input options is more productive in this scenario. The combination of several adaptive UIs and a reliable framework for synchronising the user interfaces provides the full functionality.

For example: There is no convenient way to edit items on a list using a mobile phone and WAP. But, there is a rich variety of editing options using the Java Swing Toolkit on a desktop computer.

When using the mobile phone only for adding a reminder of what to edit should be enough to remind you to edit the list, once sitting in front of your desktop computer where the UI offers the complete functionality and was automatically updated by the system to reflect the changes made on the mobile phone.

The choice of a mobile phone with integrated WAP browser as a device in the `W@PNotes` project has three reasons: (1) It is the most ubiquitous device available today; (2) it has built-in access to the world-wide web via WAP; (3) it a very limited device.

Throughout the next section we describe the realisation of this research goals in the W@PNotes system.

W@PNotes -- Architecture and Design Goals

Roughly, this section is separated into three parts: (1) A short description of the functional part of the W@PNotes system; (2) a description of the backend part of the service; (3) the mechanisms used to transport a specific front-end dynamically to a client.

The functional part of the W@PNotes-system is a distributed and multi-user enabled PostIt-like application, which is neither new, nor exciting. The application itself serves only as a demonstration and proof of concept for the underlying architecture for composing application out of separated services and bringing application functionality over a network to all sorts of devices. The functionality is limited to some of the features which can be expected within a PostIt-like application: Addition and naming of a note, deletion, altering of items in a note. Moreover, some convenience functions were defined.

The PostIt-like application consists of a single backend database, storing and maintaining the actual notes for each user of the system (see figure 1). Through a well defined interface other components of the application gain access to the stored messages. A user-authorisation mechanism is under development in order to protect the privacy of the messages stored in the system.

Separated, except for a common message-queue, an event mechanism can be coupled to the database. This mechanism, following the observer pattern, can communicate changes in the database to interested listeners via the remote event mechanism defined by Jini. We use remote events to communicate changes made on one UI to all other UIs. Every UI has to register with the W@PNotes event service after initialisation and to update the UI according to the type of event received after a change in one of the active UIs was made.
One architectural goal at this point of design was, to explore to which degree separation of concerns can be achieved by using Jini services as components of a system and therefore, how and whether systems can be composed out of autonomous entities only and which impact such a paradigm has on the design process. This is what we call service-oriented design. Both, backend database and event mechanism, are accessible as separate services. Both services are registered as fully qualified services in the Jini registry, called Lookup Service (LUS) [5], and can be found by the standard lookup mechanism [4] provided by the Jini infrastructure. These components are autonomous, functionally separated, and can communicate only through well defined interfaces, defining their functionality on a syntactical and type-safe level. The overall functionality of the application is achieved by combining appropriate services at runtime, using the infrastructure provided by Jini and the framework defined by the W@PNotes system. Figure 2 illustrates the different components and protocol steps at runtime.

Three major players can be identified: (1) The W@PNotes-service, consisting of event mechanism and service

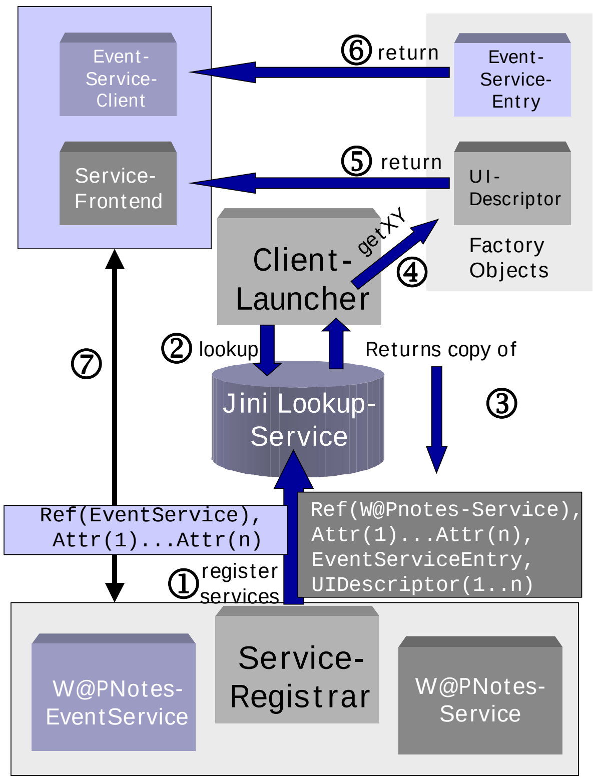

Figure 1. Using Mobile Code for Transportation

functionality; (2) the client, consisting of some initial bootstrapping mechanism (client-launcher) and dynamically downloaded and initialised code, used for event delivery and service interaction; (3) the Jini infrastructure.

At runtime the behaviour is as follows:

- Once initialised, the services will register with the Jini Lookup Service (refer to (1) in figure 2). The registration consists of (a) a reference to the actual service and (b) a set of Java objects, called attributes or entries, which by type and behaviour define descriptive information about the service. Note that Jini relies on code mobility offered by the Remote Method Invocation mechanism (RMI) of Java, which means that attributes are transported to a client and might become active there. This feature is used for client initialisation.
- Due to the overall architecture there is no explicit need for a special client application knowing about the W@PNotes service[1]. The architecture chosen, makes use of a framework defined in the "Jini ServiceUI" project (see [2]) which defines a standardised way for describing, generating and

[1] The prototype uses a generic Jini desktop as client launcher, developed as part of another project and which has no *a priori* idea of the W@PNotes system.

initialising user interfaces based on a role model. In this framework, factory objects are used to generate the appropriate UI at runtime. Those factories are encapsulated and transported to a potential client as Jini attributes. The transportation takes place during the lookup process (refer to (2), (3), and (4)).

A similar approach was developed to describe, find, and initialize various event sources for state-change propagation. As a result, the whole client part of the `W@PNotes` service is transported or generated on demand at runtime (refer to (5) and (6)) on the platform running the launching application and is initialised by information, either contained in the attributes, or obtained dynamically by a sequence of lookup-operations on the LUS. Using context from different sources than the Jini infrastructure is scheduled for future research.

The architecture is easily extensible. As both services, `W@PNotes` service and event service, are decomposed into their functional part and their UI or client, respectively, new UIs and event clients can be "plugged in" at runtime by re-registering the service with a new set of attributes. The new attributes encapsulate factories for the new UIs and event clients, together with a new description of their role. By the time of this writing a Java Swing user interface with full functionality, in the role of a MainUI, and a programmatic user interface, in the role of a WapUI, exist. One might think of a Java AWT (Abstract Window Toolkit) user interface, as well as of a Java Applet.

The WapUI programmatic interface is transported and initialised as described above. Once initialised the WapUI has the functionality of a `W@PNotes`-to-WML (Wireless Markup Language [6]) gateway. On the host running the gateway a server-socket is opened and is listening for incoming HTTP-requests. This requests are mapped to appropriate method-calls on the `W@PNotes` service. Any return-value is used to generate an appropriate WML document which is delivered via HTTP to the WAP-Gateway the user uses for access from her mobile phone (see [6] for the details on WAP). Part of the WML-document are the command options the user has at this stage, encoded as links in the document. In short, the content and command options are dynamically translated to a virtual WML document tree.

Part of the WML documents are input fields for the user (e.g., adding a new item to a note is translated to an input field). The input typed in here is translated by the WapUI to an appropriate method call on the W@PNotes service (i.e., *addItem(<UniqueNameofNote>, <itemtoadd>)*). The `W@PNotes` architecture is then responsible for delivering the change made on the mobile phone to all other user interfaces currently registered with the service. One major drawback of using WAP is visible here: The synchronisation mechanism is not able to *push* changes made on other UIs to the mobile phone. The change is visible only if the user is requesting the document on which the change is visible. This drawback is getting importing when thinking of allowing other users of the system to add notes for a user to the system.

Summary and Discussion

Various research goals have been tackled in the architecture presented here. We feel, distributed user interfaces and HCI are touching many layers of a distributed system. Especially, separation of concerns, like, separation of function and UI is very important. Otherwise adaptivity and extensibility are hard to guarantee. Also, on lower layers of an infrastructure for ubiquitous computing the choice of services as separated components seems to be a feasible approach. Here, asynchronous delivery and encapsulation of state changes into discrete events seems to be a good approach. But, as mentioned, not every protocol is well suited for delivering changes to a device. WAP as an example is not able to *push* content to a device like a mobile phone. Here, the WapUI is the terminal point of the architecture. Also, the limitations of a mobile device imposed by the small display and the keypad as input device are painful. This limitation might get weaker in the near future with the advent of new protocols and devices for the consumer market. To summarise, we think that the approach presented in this paper, as an example of ongoing research in the field of UI design for ubiquitous computing, is both: Promising and far from being finished.

REFERENCES

1. Mark Weiser. The Computer for the Twenty-First Century. http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html
2. ServiceUI Project. http://www.artima.com/jini/serviceui/.
3. Sun Microsystems Inc. Jini Architecure Specification– Revision 1.1, 2000.
4. Sun Microsystems Inc. Jini Discovery and Join Specification– Revision 1.1, 2000.
5. Sun Microsystems Inc. Jini Lookup Service Specification– Revision 1.1, 2000.
6. The Wireless Application Protocol Forum. http://www.wapforum.org/.