

# Providing Device Independence to Mobile Services

Stina Nylander and Markus Bylund

SICS, Lägerhyddsvägen 18, SE-752 37 Uppsala, SWEDEN  
{stina.nylander, markus.bylund}@sics.se

**Abstract.** As electronic services are spreading in our society, they will need to be able to adapt to different users and different usage contexts. Different user interfaces will be needed for different devices and different contexts. We envision a way of developing services where the ability to adapt is included from the start. We use a set of *interaction acts* combined with customization information to create tailored user interfaces. A calendar service has been implemented with user interfaces for Java Swing, HTML and std I/O.

## Introduction

Electronic services become more and more common in our society, and will continue to spread to new areas of our life. Soon we will be accustomed to use services in our everyday life ranging from very simple ones that for example let us control doors or lamps, to more complex ones providing teaching or entertainment.

Developing services in this context will create new requirements on service providers. Until now, the focus on service research and development has been access. When electronic services become more complex and a ubiquitous part of society, access will not be enough. Services must be capable of providing good user interaction with user interfaces that are tailored to the situation of use. This means adapting user interfaces to different devices, different contexts and different users.

In our vision, services are designed from the start to allow this. User interfaces should be created to be customizable for different kinds of use, different usage situations and different devices. The user interface of a certain application could e.g. be customized for a handheld device to be used during travel, or to be accessed from a speech user interface to accommodate users with repetitive strain injuries.

To realize this vision, we are experimenting with a set of *interaction acts* to describe the user-service interaction in a general way, without any device specific presentation information. The interaction acts are complemented with device and service specific presentation information. This way, no part of the work with development and maintenance will be done more than once, and the control of user interface presentations is kept with the service provider.

## Background

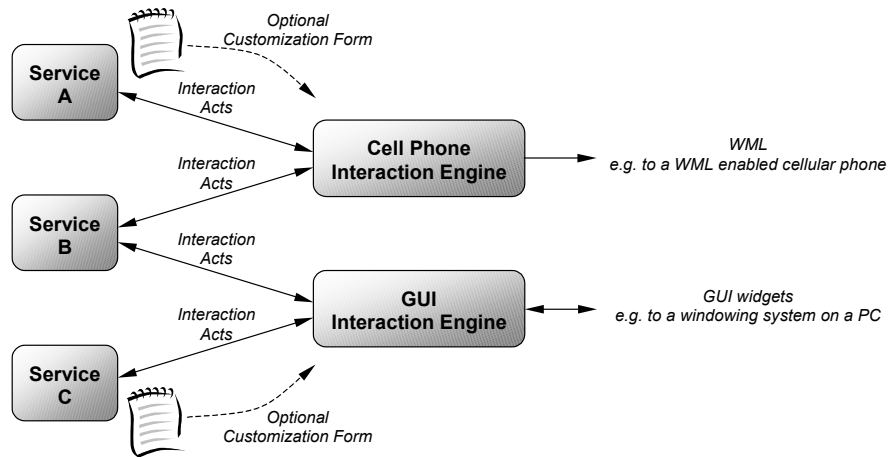
In the last decade, the dominating ways to achieve services with user interfaces that are tailored to different usage situations have been to either implement a new version of the service for each device, or to find a common ground between devices and settle for a single minimal implementation for all of them. Neither of these solutions are satisfactory. Implementing a new version of a service for each device that will be used to access it, or each situation it will be used in, makes both development work and maintenance cumbersome. Different implementations will be made by different people at different times, which will make checks necessary to keep the user interface consistent over different platforms [2]. Using a basic ground between devices to make a service accessible from different devices makes it difficult to take advantage of device specific features like external buttons or scroll wheels. It also limits the user interface since it cannot exceed the capabilities of the thinnest client.

Model-based programming was another attempt to create services with user interfaces tailored to different devices, where the user interfaces were generated from a declarative model. This approach never caught on since the generated user interfaces were unpredictable and the models were difficult to work with [1, 5].

One way to make services accessible for people with special needs have been through assistive technology. Even if better design can improve the possibilities for users with special needs to use mainstream products, assistive technology will still be the best solution in many cases, and in some cases the only one [10]. However, assistive technology has some problems. It is often more expensive than mainstream products, and it seldom manages to keep up with new versions of applications [10]. In our vision, accessing services from assistive technology will be treated the same way as any other device. Thus, we hope to make it easier to connect assistive technology to electronic services.

## Requirements and Limitations

To avoid multiplying development and maintenance work for services with different user interfaces, development methods need to be able to express service interactivity on a level of abstraction that is independent of device or situation, application, and type of user interface. They must not restrict services to certain types of interfaces, e.g. by excluding voice based user interfaces, or by relying on certain types of user-service interaction such as user-driven interaction. For example, HTML is device and application independent, but can only provide user-driven and page-based interaction. The abstract description of the user-service interaction must work on different platforms and different devices, and the abstract units of interaction must be useful for many different applications and different types of interaction.



**Fig. 1.** A design overview of the system for device independent access to mobile services. A number of different services (A through C) executing on a server specify their interaction with interaction acts. Two different interaction engines generate user interfaces for two different types of devices based on the interaction acts of the services. Service A and C have tailored the generation of their interfaces by implementing customization forms. The cell phone interaction engine is running on a server, while the GUI interaction engine is running on the PC

It should be possible to develop a service for an open set of devices and user interfaces. Making a service available from a new device or a new user interface should not create a need to modify the existing application.

Development methods also need to give service providers all possibilities to control the presentation of services to end-users, the “look and feel” of the product [3]. Branding and look and feel are commercially important, and a method that supports this is more attractive than others.

Our approach fulfills the above requirements by providing a description of the user-service interaction that is independent of type of application, type of device and type of user interface; allowing new interface types or devices to be added at any time without changing the service; and keeping the control of the presentation of the user interface with the service provider.

Although we are aiming for general solutions, which cover interaction with many sorts of applications via a large range of interface types, we realize that it might be difficult and in some cases not even desirable to develop services using interaction acts. Some services might be too complex, while others might be too device dependent to benefit from this approach (for example, a high-end multi-player game might not be interesting to play from other platforms than the one it was developed from). For the time being, we are therefore limiting our vision to a few interface types, (mainly windows-based GUIs, command-line interfaces, and speech interfaces), and more simple services (e.g. information services). Our long term-goal is, however, to achieve as wide and general solutions as possible.

## Design

We propose a solution in which the user-service interaction is the level of abstraction. This interaction can be broken down to a small set of interaction acts, which in different combinations allow the user to accomplish different tasks. For example, the act of making a choice from a set of alternatives is the same independently of system and modality, while the means of presenting the alternatives and performing the choice may change, e.g. between a pull down menu, a menu in a speech user interface or physical buttons on an ATM machine. Using interaction acts, services can offer users all kinds of interaction without assuming how the final user interface will look, thus creating great flexibility. The interaction acts can at run-time be mapped to any kind of rendering technique to create a device and service specific user interface. A schematic picture of the system can be seen in figure 1.

### Interaction Acts

An interaction act is an abstract unit of user-service interaction, which is stable over different types of user interfaces as well as different types of applications. No presentation information is included in the interaction act.

We have established a set of four basic interaction acts: `input`, `output`, `selection` and `modification`, where `input` is input to the system, `output` is output from the system that cannot accept user operations in return, `selection` is a choice between at least one alternative, and `modification` is an invitation to modify existing data (for example a calendar entry). These interaction acts can be grouped and groups can be nested to provide different interaction possibilities. All interaction acts or groups of interaction acts can be named. When running, services present hierarchically grouped sets of interaction acts from which user interfaces will be generated, and all actions performed by the users are sent back to the services encoded as interaction acts. The content of the set of interaction acts can be based either on user actions, (e.g. a response to a choice) or on system initiatives (e.g. a reminder).

With this approach, interactions of services can be generally specified once and for all. Since the user-service interaction is general, services never need to keep track of which device is currently used for access, and the same implementation can serve all devices. A service can be developed for an open set of devices and new interfaces can be added without changes in the service. This also allows for simple maintenance. With one single implementation of a service serving many different user interfaces, there will only be one version of the service to maintain.

### Customization Forms

The general interaction descriptions can be complemented with optional customization forms that contain information about how the user interface should be rendered on a given device and user interface. Customization forms map single or groups of interaction acts to behavior. They can for example contain GUI widget

templates for generation of dialog boxes based on selection interaction acts. They can also include media resources (such as images and sounds), or links to media resource databases. Mappings can be based on the type of interaction act, or on the type and the name of an interaction act in combination. As such, customization forms are both device and service specific. If customization forms are not provided, user interfaces can be rendered with default settings for look and feel. We have seen from earlier attempts that it is important that service providers can control the way services are presented to their end-users. The model-based systems for example, suffered from not being able to offer that, and many of the HTML plug-ins stem from the same need [3]. By providing detailed customization forms, application providers get full control of how every part of their user interfaces are generated for particular devices.

This approach will not only facilitate service development, it will also benefit the users. Users will get interfaces tailored to the devices that they use to access services. With interaction acts and customization forms combined it will be possible to use device specific features like scroll wheels and other external controls, designed to facilitate use in certain situations. It also ensures that user interfaces do not include components that the designated devices cannot handle, for example sound on a device without a speaker.

In a commercial setting, the customization form for a service is created by the service provider for the devices or user groups that the service targets. In those cases where the user-service interaction description is open, a customization form could be provided by other parties than the service provider.

## Implementation

We have implemented the design described above. It is composed of two different parts: a device specific *interaction engine*, and an optional service specific *customization form* (see Figure 1). The user-service interaction is expressed in interaction acts, and interpreted by the interaction engine that renders the user interface. If an interaction customization form is provided the user interface is rendered according to that, otherwise default renderings are used. Additional modules for generating interaction acts, parsing interaction acts, and communication between interaction engines and services have been implemented.

To test the system, we have developed a simple calendar service.

### Interaction Engine

Interaction engines are specific to both user interface type and device but independent of services. Their task is to interpret interaction acts presented by services and, using a customization form if there is one, generate a user interface of a certain type for a specific device. They are also responsible for interpreting user acts, which are encoded and returned to the service as interaction acts. Since an interaction engine is user interface and device specific, the interface that is generated is always adapted to the presentation and interaction capabilities of each device type. E.g. an interaction

engine on a device with a monochrome display will not try to render a user interface with color-based presentations.

Interaction engines contain default renderings for the basic interaction acts to make sure that user interfaces can be generated even if customization forms are not present. In such cases, only the type of each interaction act is used to determine how it should be rendered.

Developers of interaction engines for new devices or user interface types need to implement the following functionality: parsing of interaction acts and encoding of user actions, creation of mappings between interaction acts and elements of the user interface, and, based on the above, generation of a user interface. In order to support the development of interaction engines, we have implemented a library of modules that handle the parsing, encoding, and mapping functionality.

Some devices may have several interaction engines for different types of user interfaces. A personal computer might for example have an interaction engine for GUIs and another for web user interfaces. We have implemented interaction engines for Java Swing GUIs, HTML user interfaces, and std I/O based user interfaces.

### **Customization Forms**

The customization forms are implemented as a structured set of mappings between templates of user interface components and interaction acts types and names of elements. The structure can also contain media resources. Customization forms allows the generation of different user interface components for the same type of interaction act, based on the symbolic name of the interaction act.

In our implementation, customization forms can be arranged in hierarchies, allowing one form to inherit mappings, resources, and links from another form. This allows for easy implementation of look and feel that is shared between several services.

### **Application**

We have developed a calendar service to test the system. The calendar is written in java and supports the basic calendar operations: adding, editing and deleting events, browse calendar information, and presenting calendar information in different views (day, week, month). The calendar uses interaction acts to describe the user-service interaction, and thus, makes no assumptions about how the user interface will be presented. To complement the interaction acts, customization forms have been developed for Java Swing, std. I/O, and HTML.

Figure 2 shows screenshots of a day view of the Swing and the std I/O user interface of the calendar service.

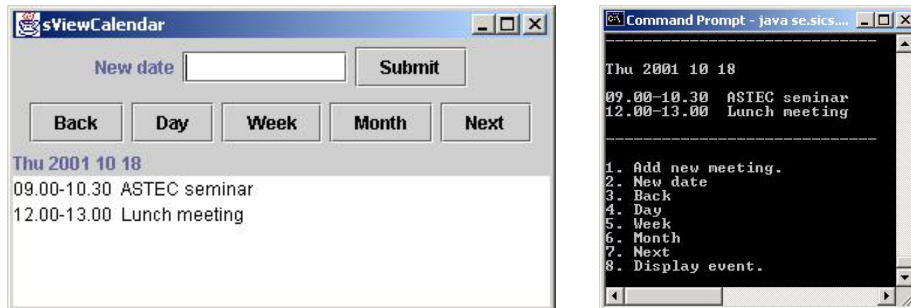


Fig. 2. A Java Swing and a std I/O day view of the calendar

## Related Work

Adapting services and their user interfaces to capabilities and preferences of different users and devices is a research issue addressed in both the mobile and the universal access research communities.

The mobile community has mostly been concerned with device capabilities and technical problems, and has thus been more oriented towards solutions with a common ground for all devices, and questions of service discovery. The XWeb project has been inspired by the Web and Web browsers [6]. Data that is sent between services and user interfaces are encoded in a general way, and measures have been taken to provide full interactivity. Users can choose a client that interprets data and presents a user interface in a modality that they prefer. However, XWeb cannot take advantage of device specific features, and it does not provide means for service providers to control the presentation of the user interface. Both these features are provided by our approach.

Hodes et al. [4] describe a model-based approach that provides different user interfaces to choose from and complements these with a general interface description. If there is no user interface that suits the device, the general description is used to generate a user interface with the help of the user. This approach suffers from the same drawbacks that traditional model-based user interfaces do: generated user interfaces are unpredictable and the models are difficult to work with [1, 5]. They are also inflexible in that the hierarchy of the user interface is fixed with the model, which prevents generation of user interfaces that require other hierarchies (e.g. a GUI vs. a speech user interface). With our approach, the presentation and the structure of the user interface can be controlled with a customization form.

The mobile research community, with its focus on technical aspects, often has problems taking user needs into account. In contrast, the universal access community has primarily focused on how to adapt services to users' capabilities or preferences, both in terms of software and hardware aspects of user interfaces. Another issue in universal access research has been how to provide different user interfaces at the same time. For example, blind persons might be best served by a user interface tailored for

blind people, but still need a GUI to be able to share their work with seeing co-workers [7].

The HOMER UIMS allows the parallel design of user interfaces for blind and seeing people using different structure and different metaphors for the different user interfaces when needed [7]. The authors argue that non-visual user interfaces should not be mere adaptations of graphical user interfaces, but need their own structure and their own tools. The concepts of Dual User Interfaces and virtual interaction objects are presented as a way to create a single user interface specification for both the visual and the non-visual user interface. There are strong similarities between the HOMER system and our approach, but the focus on blind users vs. seeing users in the HOMER project has strongly influenced the design of the language and the implementation. In contrast, our approach is developed to create user interfaces of many different types for different devices. This requires a more general design of the language and possibilities to handle a larger number of interface types than only visual and non-visual.

Unified User Interfaces (UUI) is a design and engineering framework composed by three parts: a method for design, a software architecture, and tools [9]. The goal of UUI is to provide user interfaces tailored to different user groups and situations of use in terms of users' physical capabilities, preferences and usage context. Our approach has a narrower scope, since we focus on adapting user interfaces to different device capabilities, which allows for greater flexibility. However, it could easily be seen as a tool that would fit in to the UUI framework, or alternatively, as a part of the software architecture suggested by Stephanidis and Savidis [8].

## Conclusions

We have presented a vision for creating electronic services with user interfaces adapted to different user needs and different device capabilities. We are experimenting with a set of interaction acts to describe the user-service interaction, and then complement this description with device and user specific information. Although this is still work in progress, we find the results promising. We have implemented interaction engines for Java Swing, HTML and std I/O, and a calendar service as sample service with customization forms for Java Swing, HTML and std I/O. Even though the calendar service is very simple, it shows that the approach is working on an initial stage. Future work will include the implementation of interaction engines for a wider range of devices together with an assessment of how our approach scales in terms of creating more complex services and customization forms than described herein. The assessment will be used as input to an iterative redesign of our approach.

## Acknowledgements

This work has been funded by the Swedish Agency for Innovation Systems (www.vinnova.se). Thanks to the members of the HUMLE laboratory at SICS, in particular Annika Waern, for thoughtful comments and inspiration.

## References

1. Eisenstein, J. and Puerta, A., Adaption in Automated User-Interface Design. in *International Conference on Intelligent User Interfaces*, (2000).
2. Eisenstein, J., Vanderdonckt, J. and Puerta, A., Applying Model-Based Techniques to the Development of UIs for Mobile Computers. in *International Conference on Intelligent User Interfaces*, (2001).
3. Esler, M., Hightower, J., Anderson, T. and Borriello, G., Next Century Challenges: Data-Centric Networking for Invisible Computing. The Portolano Project at the University of Washington. in *The Fifth ACM International Conference on Mobile Computing and Networking, MobiCom 1999*, (1999).
4. Hodes, T.D., Katz, R., H., Servan-Schreiber, E. and Rowe, L., Composable Ad-hoc Mobile Services for Universal Interaction. in *MobiCom 1997*, (1997).
5. Myers, B.A., Hudson, S.E. and Pausch, R. Past, Present and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction*, 7 (1). 3-28.
6. Olsen, D.J., Jefferies, S., Nielsen, T., Moyes, W. and Fredrickson, P., Cross-modal Interaction using XWeb. in *UIST 2000*, (2000).
7. Savidis, A. and Stephanidis, C., Developing Dual User Interfaces for Integrating Blind and Sighted Users: the HOMER UIMS. in *Human Factors in Computing Systems*, (Denver, Colorado, 1995).
8. Savidis, A. and Stephanidis, C. The Unified User Interface Software Architecture. in Stephanidis, C. ed. *User Interfaces for All - Concepts, Methods, and Tools*, 2001, 389-415.
9. Stephanidis, C. The Concept of Unified User Interfaces. in Stephanidis, C. ed. *User Interfaces for All - Concepts, Methods, and Tools*, 2001, 371-388.
10. Vanderheiden, G.C. Universal Design and Assistive Technology in Communication and Information Technologies: Alternatives or Complements? *Assistive Technology*, 10 (1). 29-36.