Providing Device Independence to Mobile Services

Revised version of SICS technical report T2002-02, Device Independent Services

Stina Nylander and Markus Bylund
Swedish Institute of Computer Science
Lägerhyddsvägen 18
SE-75237 Uppsala, SWEDEN
+46 18 471 70 {41, 53}
{stina.nylander, markus.bylund}@sics.se
SICS Technical Report T2002:02A
ISSN 1100-3154
ISRN:SICS-T--2002/02A-SE

ABSTRACT

People want user interfaces to services that are functional and well suited to the device they choose for access. To provide this, services must be able to offer device specific user interfaces for the wide range of devices available today.

We propose to combine the two dominant approaches to platform independence, “Write Once, Run Every-where™” and “different version for each device”, to create multiple device specific user interfaces for mobile services. This gives possibilities to minimize the work with development and maintenance, while still keeping the control of how the user interface is presented to the end user. A calendar service has been implemented with user interfaces for Java Swing, HTML and std I/O.

This report is a shortened and improved version of the SICS technical report T2002-02.

Keywords

User interface design, device independence, mobile services, user interfaces.

INTRODUCTION

Today, users can choose from a wide range of devices to access services they need. Desktop computers, laptops, PDAs and mobile phones are all designed with different features and capabilities to be used in different situations. To offer aesthetical and fully functional user interfaces on a wide range of devices, services must be able to present device specific user interfaces. If developers are to keep up with the pace of which new devices are released on the market, and the number of already existing devices, we

need to find simple and robust ways to create different user interfaces for different devices (Myers, et al., 2000).

Service providers use two different development methods to obtain user interfaces for different devices. The first is to send the same code to all devices and leave the presentation of the user interface to the device, where XWeb (Olsen, et al., 2000) is a recent example. The second is to maintain a library of different presentations and send the right one to the right device, as Hodes et al. do (Hodes, et al., 1997). Both methods have problems. With the first, service providers lose control of the presentation of the user interface, with the latter, work with development and maintenance gets multiplied.

We have identified some development requirements that need to be fulfilled to avoid the problems described above. To achieve this, we combine the two development methods, “write once, run everywhere” and “different version for each device”, to provide tailored user interfaces to different devices while avoiding maintenance of a vast range of different versions. We use 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 then 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 control of the user interface presentation is kept with the service provider.

BACKGROUND

Developing services in a way that makes it possible for them to run on different devices is not a new research issue. In the early days of computer history, lack of standardization gave birth to many different user interface styles and input devices. Model-based programming was one way to solve the problem, with systems like ITS (Wiecha, et al., 1990) and Mastermind (Castells, et al., 1997), where the user interface was generated from a

Figure 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.

declarative model. This approach never caught on since the models were complicated to work with and the user interfaces generated by the model were unpredictable (Eisenstein and Puerta, 2000, Myers, et al., 2000). In the eighties, the need for applications able to run on different devices almost disappeared with the emergence of graphical direct-manipulation that made desktop computers on Windows, Macintosh and Unix look similar.

Today, we are back to a situation where we need to face a wide range of devices with great differences in presentation and interaction capabilities that are used in many different settings, both mobile and stationary. However, this time differences are due to design decisions rather than lack of hardware standards. Users with devices designed for specific use in specific situations will not settle for user interfaces created to suit all devices. Moreover, since devices today are designed to *be* different, it is unlikely that standardization will solve the application problem this time.

In the last decade, the answer to this problem has been either implementing a new version of the service for each device, or finding a common ground between devices and settling with a single minimal implementation for all of them. Neither of these solutions is satisfactory. Implementing a new version of a service for each device that will be used to access it makes both development work and maintenance cumbersome. The different implementations will be made by different people at different times, which will make consistency checks

necessary to keep the user interface consistent over different platforms (Eisenstein, et al., 2001). 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.

REQUIREMENTS

Development methods must fulfill some requirements to produce truly device independent services with fully functional and aesthetical device specific user interfaces.

To avoid multiplying development and maintenance work, development methods must be able to express service interactivity on a level of abstraction that is independent of device, application and type of user interface. They must not restrict services to certain types of interfaces, e.g. exclude voice based user interfaces, or rely on certain types of user-service interaction, e.g. only 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.

It must 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 must not affect the existing application.

Development methods must also give service providers all possibilities to control the presentation of services to end-users, the “look and feel” of the product (Esler, et al., 1999). Branding and look and feel are commercially important, and a method that supports this is more attractive than others.

DESIGN

We have designed 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 systems or modalities, while the means of presenting the alternatives and performing the choice may change, e.g. between a pull down menu or radio buttons. Using interaction acts, services can offer users all kinds of interaction without any assumptions about 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 architecture 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 back, selection is a choice between at least one alternative, and modification is a possibility of 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, the service presents hierarchically grouped sets of interaction acts from which a user interface will be generated, and all interaction acts performed by the user is sent back to the application. 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, the interaction can be generally specified once and for all for a service. Since the user-service interaction is general, services never needs 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 description of the interaction is complemented with an optional interaction customization form that contains information about how the user interface should be rendered on a given device. Customization forms map single or groups of interaction acts and customization information to behavior. It can for example contain GUI widget templates for generation of dialog boxes based on select 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, a customization form is both device and service specific. If a customization form is not provided, the user interface is rendered with default settings for look and feel. We have seen from earlier attempts that it is important that the service provider can control the way services are presented to their end-users. The model-based systems suffered from not being able to offer that, and many of the HTML plug-ins stem from the same need (Esler, et al., 1999). By providing a detailed customization form, an application provider gets full control of how every part of the user interface is generated for a particular device.

This approach will not only facilitate service development, it will also benefit the users. Users will get interfaces tailored to the devices used to access services. With interaction acts 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 the user interface does not include components that the device cannot handle, for example sound on a device without speaker.

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 *interaction 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 calendar service.

Interaction Engine

Interaction engines are specific to both user interface type and device. Their task is to interpret interaction acts presented by services, map them to a customization form if there is one, and generate a user interface of a certain type for a specific device. They are also responsible for interpreting user interaction acts, which are encoded and

Figure 2: A Java Swing day view of the calendar.

returned to the service. Since the 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. An Interaction engine on a device with a monochrome display will not render a user interface with color-based operations.

An interaction engine contains default renderings for the basic interaction acts to make sure a user interface can be generated even if a customization form is not present. In this case, only the type of the elements is used to determine how it should be rendered.

Some devices might 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 standard I/O based user interfaces. The interaction engines accept customization forms in the form of tables with mappings between templates of user interface components and types and names of interaction acts. The tables can also contain media resources.

Interaction Customization Forms

The interaction customization forms are implemented as tables with mappings between templates of user interface components and interaction acts types and names of elements. The tables can also contain media resources. Interaction 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, interaction customization forms can be arranged in hierarchies, allowing one form to inherit mappings, resources, and links from another form. Several services could also share customization form. This allows for easy implementation of look and feel that is shared between several services. Customization forms are developed separately from services. When novel devices and user interface types appear, service providers can implement new customization forms without having to modify the service logic.

Figure 3: A std I/O day view of the calendar.

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, Interaction Customization Forms have been developed for Java Swing, std I/O, and HTML.

Figure 2 and 3 shows screenshots of a day view of the Swing and the std I/O user interface of the calendar service.

RELATED WORK

Device independence is a research issue addressed in both the mobile research community and that of Universal Access. Both communities strive for good user interaction, and to give users possibility to choose the equipment that best suites their goals and preferences. However, they have different focus in their work. The Universal Access is focused on users' preferences and capabilities, including physical disabilities, as in the AVANTI project, where a web-browser is developed with a traditional graphical user interface, an interface for blind people, and one for motor-impaired people (Stephanidis and Savidis, 2001).

The mobile research is more focused on the capabilities of applications and devices. Several recent attempts have been made to develop methods for creating device independent applications and appropriate user interfaces to them. The majority of these attempts adhere to one of two traditional approaches. The first is to send the same code to all devices, the second to maintain a version of the application for each device.

HTML is the most spread representative of the first approach. A basic commonality is used for all user interfaces and the presentation is left to the device that is running the service. However, to settle with a basic commonality has some drawbacks. For example, it makes it impossible to take advantage of device specific hardware, e.g. hard buttons or scroll wheels, and very difficult to change interface modality (Banavar, et al., 2000). HTML is also limited to one type of user interface, and page based user-driven interaction. Leaving the presentation to the

client device implies that the developer only has limited control of how the user interface is presented to the user, the “look and feel” of the service. Our approach provides means to tailor user interfaces to device specific features, and supports different types of user interfaces as well as application-driven interaction. We also provide a possibility for the service provider to control the presentation of the user interface.

The XWeb project has been inspired by the Web and Web browsers (Olsen, et al., 2000). 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 the service provider to control the presentation of the user interface.

The User Interface Markup Language (UIML) is another representative of the first approach. It offers a possibility to specify user interfaces in a markup language that can be converted to another language, e.g. Java or HTML (Abrams, et al., 1999). However, it shares the drawback of not being able to take advantage of device specific features with HTML and the XWeb approach. UIML also only supports user-driven interaction.

The CC/PP framework is an initiative to create a standard for expressing capabilities and preferences of clients and users, to which service providers can adapt content and presentation (Reynolds, et al., 1999). CC/PP is well suited for adaptations of differences between user interfaces of the same type, such as adaptations to differences in screen size of mobile terminals, to the number of colors available to a Web-browser, or to the preferred interaction language of the users. It is however cumbersome to implement services that, based on CC/PP profiles, generate user interfaces with fundamental differences in structure and presentation abilities (e.g. standard I/O-based interfaces, GUI-based interfaces, and speech interfaces).

The Challenges project presents an application model for pervasive computing where services are created with focus on the tasks the user wants to perform and the information the user needs to do so, rather than on what device the service will run on or how the user interface should look (Banavar, et al., 2000). The application model states that abstract user interface specifications, based on the identification of tasks and subtasks, could help to provide device independent services. While this is a promising approach that in some ways resembles our solution, it has not been implemented and the claim that tasks and subtasks are the most suitable abstraction for device independent interface specifications remains to be verified.

Hodes et al. are representatives of the second approach, with different versions (Hodes, et al., 1997). They 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 (Eisenstein and Puerta, 2000, Myers, et al., 2000). 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 Web-based user interface).

CONCLUSIONS

Handling different user interfaces is necessary for a service that is accessed from different devices, something that is more and more common these days. We need to find new and better ways to create different user interfaces for these kinds of services. In this paper, we have presented a design where the interaction between users and services is the common denominator for the different user interfaces. The user-service interaction is specified using interaction acts. Based on sets of interaction acts, different device specific interaction engines generate different user interfaces for a service. The interaction acts can be complemented by an interaction customization form, which gives detailed information about how the user interface should be generated on a certain device. This gives the service provider a possibility to control how the service is presented to the user.

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. The implementation shows promising results in that our design supports somewhat complex services, and allows users to access the information from different user interfaces in an appropriate way. It also clearly shows that different user interfaces with different structure can be derived from the same sequence of interaction messages.

ACKNOWLEDGMENTS

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

REFERENCES

1. Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S.M. and Shuster, J.E. UIML - an appliance-independent XML user interface language. *Computer Networks*, 31. 1695-1708.
2. Banavar, G., Beck, J., Gluzberg, E., Munson, J., Sussman, J. and Zukowski, D., Challenges: An Application Model for Pervasive Computing. in *7th International Conference on Mobile Computing and Networking*, (2000).
3. Castells, P., Szekely, P. and Salcher, E., Declarative Models of Presentation. in

- International Conference on Intelligent User Interfaces*, (1997).
4. Eisenstein, J. and Puerta, A., Adaption in Automated User-Interface Design. in *International Conference on Intelligent User Interfaces*, (2000).
5. 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).
6. 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).
7. Hodes, T.D., Katz, R., H., Servan-Schreiber, E. and Rowe, L., Composable Ad-hoc Mobile Services for Universal Interaction. in *MobiCom 1997*, (1997).
8. 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.
9. Olsen, D.J., Jefferies, S., Nielsen, T., Moyes, W. and Fredrickson, P., Cross-modal Interaction using XWeb. in *UIST 2000*, (2000).
10. Reynolds, R., Hjelm, J., Dawkins, S. and Singhal, S. *Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation*, W3C Note, World Wide Web Consortium W3C, 1999.
11. Stephanidis, C. and Savidis, A. Universal Access in the Information Society: Methods Tools, and Interaction Techniques. *Universal Access in the Information Society*, 1 (1).
12. Wiecha, C., Bennett, W., Boies, S., Gould, J. and Greene, S. ITS: a Tool for Rapidly Developing Interactive Applications. *ACM Transactions on Information Systems*, 8 (3).