

Defining Services for Mobile Terminals using Remote User Interfaces

Richard Verhoeven

and

Walter Dees

Eindhoven Embedded Systems Institute
Eindhoven University of Technology
Den Dolech 2
5612 AZ Eindhoven
p.h.f.m.verhoeven@tue.nl

Philips Research Laboratories
Information & Software Technology
Prof. Holstlaan 4
5656 AA Eindhoven
walter.dees@philips.com

Abstract

Mobile phone operators and manufacturers are looking for ways to create network services and to increase revenue. Services like WAP and HTML are available, but limited in their capabilities and user interaction models, that is, page-based or form-based interaction. The remote user interface protocol has the advantages that it is device independent and allows for better interaction between the user and the service, due to incremental updates. The protocol is useful for broadband applications, such as highly interactive airport services and responsive remote control of devices within in-home wireless networks. We tested example services under different circumstances.

1 Introduction

The operators of mobile phone networks are looking for ways to generate more revenue on the existing GSM and GPRS infrastructure, in order to make up for the costs of the UMTS licenses and to be able to install the UMTS infrastructure. SMS is a very successful technology in the GSM network with several *services* built on top of it, but it is not sufficient for more advanced *services*, such as the ones found on the Internet. After a failed attempt with WAP 1.0, parts of the Internet are now made accessible for mobile phone users in the form of I-mode sites, and in the near future WAP 2.0. Both I-mode and WAP 2.0 will use an always-on connection to be more cost effective and responsive towards the user.

Similar to Internet services, Intranet services are also made available to mobile terminals. For example, on a wireless LAN, the airport services could be made accessible for business travelers. In the future, when in-home networks are more common, it is likely that similar techniques are used to allow access to the *services in your house*, which are provided by electronic devices like TVs, VCRs and thermostats, to allow remote monitoring, diagnostics and usage. For example, the user can set the starting time of the central heating with a mobile phone, or answer the telephone via the television. The conditions to enable these scenarios include easy installation, ease of use and interoperability.

A mobile phone would be a good choice to provide access to these in-home services. It provides a reasonable user interface, it has some processing power and the user tends to carry it around. Future mobile phones will support different technologies to connect to services. These could be used to connect to the home network differently, depending on the situation. In the house, it could use InfraRed[12], ZigBee[21] or BlueTooth[1], around the house, it could use wireless LAN[19], and further away, it could use GPRS[6] or UMTS[17]. The location of the phone could also be used in location-aware services, like turning on the nearby light.

In this paper, the *remote user interface protocol* is described, from the perspective of services for mobile devices, also called *mobile services*. In section 2, several existing protocols for such services, like WAP and i-mode, are described with their limitations. In section 3, different architectures for network enabled remote control are described and the drawbacks of existing protocols are discussed. In section 4, the remote user interface protocol is described,

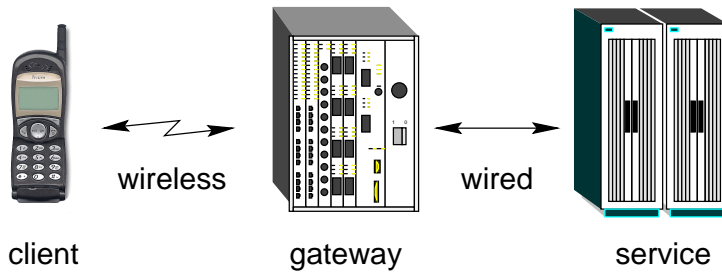


Figure 1: The network architecture for mobile services. The client contacts a gateway through wireless communication, to access a service on some server. The gateway and server are typically connected by wired communication.

including the main differences with the existing protocols. In section 5, some examples are given on how the remote user interface protocol can be used to create interactive mobile services. In section 6, some concluding remarks are given.

2 Mobile services

Currently, several protocols are available to define mobile services. The protocols and solutions which are used for this are different from the solutions used in the PC world. The memory resources of mobile devices are limited, which makes it difficult to process the services from the PC world with their increasing richness with respect to images, animations and screen layout. Other topics of concern are the limitations on processing power, the battery power and the communication channel. Furthermore, the variety of input and output devices makes the design of mobile services very complex.

The general network architecture for mobile services is given in Figure 1. The *client* makes a connection to a *gateway* provided by the network operator. Through this gateway, the client can access *services* provided by large powerful servers. The gateway provides wireless network connectivity, access control, protocol translation and billing for network usage.

The communication from the service to the client consists of *user actions*. The communication from the service to the client consists of *user interface descriptions*. A *presentation engine* on the client presents these user interface descriptions to the user and enables the user to perform actions on the service through this user interface.

The most common technologies for providing services are shortly described in the following subsections.

2.1 SMS

SMS[16] stands for Short Message Service and allows the exchange of short textual messages between mobile phones on a GSM network. SMS is a very popular service in Europe, where it is mainly used by young people. Several services are defined on top of SMS, such as weather forecasts, traffic information and news on rock concerts. To receive these services, you either subscribe to the service or you send a specific SMS message to a service phone number to receive a one-time reply.

One advantage of SMS over other protocols is the ability to easily send messages between mobile phones, thereby allowing peer-to-peer communication. Another advantage is the small size of the SMS presentation engine, as it only has to present a short textual message to the user, with an option to send a new message or reply to an existing one.

SMS can be considered as the mobile version of e-mail, where the services are comparable to moderated mailing lists and auto-replyers. Similar to e-mail, it is difficult to construct a highly interactive service with this technology.

2.2 WAP

WAP[18] is a set of protocols for wireless applications. Version 1.0 uses the markup language WML. Since WML does not resemble HTML, it requires additional authoring efforts for service providers. Version 2.0 is based on XHTML[20] with the Mobile Profile and cascading style sheets, which is a small subset of HTML. It does not support scripting or frames, as commonly used in web sites, but it can support basic tables and embedded objects (if code is available to handle them). For describing user preferences and device capabilities, the CC/PP framework[3] has been specified. For services that need to send notifications to clients, a push model is available, but it requires a Push Proxy Gateway.

Due to the resemblance with HTML, web sites can be made available under WAP 2.0 more easily, so services will emerge faster. Furthermore, the larger bandwidth and always-on connectivity of GPRS and UMTS will improve the user experience and reduce the usage costs, in comparison to the smaller bandwidth and connection-oriented connectivity of GSM.

2.3 I-mode

I-mode[10] is a protocol to allow service access on mobile phones, similar to WAP 2.0. I-mode is very successful in Japan and is recently introduced in Europe as well. It uses a compact version of HTML, called cHTML[2], which removes all the complicated elements and features from HTML, such as scripting, frames, tables, styles sheets and embedded objects. I-mode supports notifications for services, which is mainly used by I-mode mail[11].

Java-enabled I-mode terminals support downloadable Java applications, called *I-appli*. The available Java functionality consists of the Connected Limited Device Configuration (CLDC[4]) of Java 2 Micro Edition (J2ME[13]). In addition, it provides I-mode and manufacturer specific extensions. For security reasons, an i-appli is not running within the I-mode presentation engine, it can only communicate using the HTTP protocol with the server that made the i-appli available and it can not start other java applications. Together with the severe code size limitations, an i-appli is restricted in its functionality.

Existing Web sites can be made available under I-mode, although the restrictions for I-mode are more severe than for WAP. Currently, I-mode does not support common image encodings like JPEG and the documentation on how to build an I-mode site specifies that an I-mode page should be limited to ten kilobytes, including cHTML markup and images. For many Web sites, such limitations are too strict.

2.4 HTML and HTTP

HTML[8] and HTTP[9] are the main markup language and protocol behind the Web sites on the Internet. Many web sites rely on a powerful presentation engine or a specific screen resolution, which are not available on a mobile device, although limited presentation engines for PDAs are available to make web sites accessible to some extent. Advanced features like scripting, frames, java applets and embedded objects require too many resources (memory, screen space or processing power) to be properly supported in the near future. However, the services available as a web site on the Internet can be used as a basis for services on mobile phones, although they have to be adjusted to fit within the imposed limitations.

3 Network enabled embedded systems

It is expected that more and more embedded systems become network enabled, such that they can co-operate with each other or can be monitored and controlled remotely. For network communication, proprietary protocols are being replaced by Internet protocols like TCP/IP to allow easy co-operation. For remote monitoring, diagnostics and usage of such a network enabled embedded system, there is not yet an appropriate standard protocol to present a *remote user interface* for the system that is being controlled.

The network architecture for network enabled embedded systems is slightly different from the architecture for mobile services, as a large powerful gateway or server might not be available in the intended network environment, such as an in-home network or personal area network. An example of such an environment is depicted in Figure 2, where the client and



Figure 2: The possible network architecture in the home environment. The client communicates directly to the service.

service communicate directly. In this scenario, the client and service have to be part of common resource constrained consumer electronics devices, where one is used to control the other remotely. A service is in this case the functionality provided by the device, for example, “recording” for a VCR.

The technologies described in the previous section are sufficient for information services, but do not suffice for controlling network enabled embedded systems in and outside the house. Therefore, within Philips Research Laboratories in conjunction with Eindhoven University of Technology, we look at requirements and protocols for network enabled control, especially with respect to user experience and responsiveness of the service. The requirements are:

1. Preferably, the remote user interface of the controlled device should be as responsive as a local user interface, especially if the user expects immediate feedback. For example, when a slider is used to select a certain frame from a video tape, the user wants immediate feedback while using the slider, and it should not result in glitches due to full screen replacements.
2. The service should be able to notify the client of the changes that occur at the service, such as progress while a VCR is rewinding.
3. The protocol should be optimized for resource usage, bandwidth and cost, which is very important for a consumer electronics device like a VCR.
4. The protocol should be device independent, as the capabilities of the client are unknown. For example, a VCR could be controlled through the TV set or through a mobile phone.
5. The protocol should be flexible with respect to bandwidth usage. For example, inside the house, the mobile phone and VCR could communicate over a BlueTooth connection, while outside the house, they might communicate over a GSM connection through a residential gateway.
6. The protocol should be robust and provide full user control. It should not allow unintended behavior.

These requirements are not covered very well by the previously described technologies and also not by current middleware standards for Internet and in-home networks. To some extent, existing technologies can be used, but there are always some disadvantages, which we will discuss in the following sections.

3.1 Existing Markup Languages

Similar to services on the Web, I-mode and WAP, it is possible to use markup languages like HTML or WML to describe the user interface and let a presentation engine present it to the user. This solution is very generic and the presentation engine can be used for other services as well. For remote control and monitoring, this solution has several disadvantages.

- The interaction with the service is *form based*, where the user makes all the selections, commits them to the service and receives a complete new interface. This does not meet requirement 1.
- It is difficult to send *notifications* from the service to the client, which is needed for requirement 2. In HTML over HTTP, the client must regularly request for updates, which

is resource consuming, thus breaking requirements 3. In WAP, a special push proxy gateway is needed, which might not be available in an in-home network. In I-mode, notifications are mainly used in combination with I-mode mail, where the implementation method depends on each operator's network[11]. In either case, the notifications would result in complete screen replacements, which breaks requirement 1.

3.2 Downloadable code

A different architecture is to use *downloadable code* or *applets*, where the service provides the client with the code that handles the user interface and the communication with the service. This is the approach taken by e.g. J2ME midlets[15], MHP[14] and I-applis. This solution is very generic and is mainly limited by the functionality provided for applets by the client. There are also some disadvantages:

- Due to privacy, security and cost aspects (see requirement 6), the provided functionality might be *too limited*, as the applet is often untrusted software. As a result, communication with the service might not be possible, thus breaking requirements 1 and 2.
- The capabilities of clients are *very diverse* in terms of memory, screen capabilities and input methods. This has to be taken into account by the applets. Due to requirement 4, the service has to provide different applets for different clients, which involves a major authoring effort and requires considerable resources at the service, thus breaking requirement 3.

4 Remote User Interface Protocol

The *Remote User Interface (RUI) protocol* is developed to address the requirements for network enabled embedded systems, as listed in section 3, while providing sufficient functionality to support mobile services, as listed in section 2. The RUI protocol is based on the Data Driven Interface (DDI) part of the Home Audio Video Interoperability (HAVi) specification[7]. The DDI part describes how one HAVi enabled device can control another by receiving an abstract user interface description and allowing the user to interact with it. As HAVi is not based on Internet standards, the DDI part had to be translated into XML and improved to be more Internet friendly. Several features have been added to improve flexibility, bandwidth usage and device independence.

4.1 General overview

With the RUI protocol, the typical behavior pattern for the client and the service is the following.

- The client *discovers* the service. This is not part of the RUI protocol. Possible ways to discover a service are a central registry, broadcasted network messages or references by other services.
- To use the service, the client *subscribes* to the service with information about the client's contact address and optionally a *terminal profile*, which describes the capabilities of the client. In return, the client receives the description of the initial user interface, which is presented to the user. This is called a *UI fragment*.
- When the user performs a *user action* on the user interface, this action is forwarded to the service, indicating the element on which the action occurred and the related values. The service replies with instructions on how to adjust the user interface (like changing the label of a button), which is also called a *UI fragment*.
- When the service wants to *notify* the client, it contacts the client via the provided contact address and sends a UI fragment with a description of changes to be made to the user interface.
- When the client or service wants to *terminate* the communication, an appropriate message is sent.

The main parts of the protocol are discussed in the following sections.

4.2 UI fragments

The messages that are transmitted from the service to the client contain XML-based *user interface descriptions*, so called *UI fragments*, which are processed by a presentation engine and presented to the user. These fragments contain abstract user interface *elements* which are common for many user interfaces, augmented with style sheets for presentation hints. The following elements have been defined:

- Container: to group several elements together
- List: to select from a list of options
- Range: to select a value from a range of values
- Entry: to enter text input
- Button: to perform an action
- Progress: to indicate progress on the service
- Text: to present textual information
- Image: to present graphical information

It is possible to extend this element set by using XML name spaces, for presentation engines that support additional elements.

The elements are abstract in the sense that the client can present them in a way that best fits the client terminal, possibly using the hints provided by the style sheets. For example, on a client with a textual interface and only four cursor keys, a Range element can be presented as a value with the option to increase or decrease it, while on a client with a graphical interface and a touch screen, a Range element can be presented as a slider. In general, each element could be presented with a native widget implementation of the client terminal.

As the interface description might contain too many elements for the available screen estate of certain clients, priorities can be used to specify which interface elements are important and which can be left out if the screen is too small. For example, a VCR can provide advanced timer features which are not absolutely necessary for normal operation.

The interaction style between the client and the service is also important. For certain services, only the final setting of an option is important, while other services require intermediate settings as well, which are called *incremental user actions*. For example, a slider to control the temperature setting of the central heating only needs to report the final setting, while a slider to select a certain frame from a video needs to report every setting to give the user direct visual feedback of the actions and as a result better control of the selection mechanism. Therefore, the interaction style of every element can be set individually.

4.3 Incremental updates

When every action results in a complete replacement of the user interface, it has an impact on the performance of the client. Sending the complete interface requires more bandwidth and more processing power to handle. Furthermore, to present the new user interface to the user, the old user interface has to be replaced by the new one, which often results in noticeable and confusing glitches in the user interface.

Instead of sending the complete interface as the result of an action, only the required changes to the existing interface are returned in the form of a partial interface description, also called a *UI fragment*. The client can adjust the existing interface by following the instructions from the UI fragment. These instructions are available on several levels: attributes of an element can be modified, elements can be added, removed or replaced and, worst case, the complete interface can be replaced. The designer of the user interface of a particular service decides how these levels are used. For example, after pressing a “play” button on a client of a VCR, the returned UI fragment could change the label of this button into “pause” or “stop”, or it could replace the button with a new one. Some examples of UI fragments are given in section 5.

The use of incremental updates puts requirements on the client with respect to caching of UI fragments and their content to allow the service to construct proper incremental updates. In particular, the service can assume that the element that triggered the user action is still available, together with the elements in the enclosing container. All other elements might not

be available due to resource restrictions on the client. The RUI protocol defines these caching rules to guarantee interoperability.

4.4 Notifications

A service might need to inform the client of changes that occurred in the service, for example, that a VCR finished rewinding the video tape. When the client subscribes to a service, it provides a contact address for the notifications. The service uses this address to send notifications in the form of UI fragments. By using UI fragments, a notification has the same functionality as the result of a user action, except that it originates from the service. That is, an attribute can be changed to indicate progress or a complete new interface can be shown to show a warning.

4.5 Client diversity

The diversity of clients is potentially very large and it is impossible for a service to address all these differences appropriately, as it would require a considerable authoring effort. The client can be a TV set, a pager or a mobile phone. Each of these devices has its own characteristics with respect to the user interface, where both input and output should be considered. A TV set has audio and a large screen to interact with the user, where a common InfraRed remote control is used to navigate. A pager only has one or two lines of text display, with limited user input facilities. A mobile phone has audio and a small screen, which could be a touch screen with character recognition. The screen diversity alone is already so large that it is impossible to address this issue on the service side or on the client side alone.

The RUI protocol provides several techniques to address these problems.

- The user interface descriptions contain abstract user interface elements, which gives a client the possibility to optimize the presentation of these elements to its own capabilities.
- The user interface descriptions can refer to style sheets, which indicate how the user interface can best be presented. To improve the reuse of style sheet descriptions and reduce the authoring effort, the style sheets are divided over multiple hierarchical levels, according to common terminal capabilities. This is described in more detail in [5].
- The user interface elements can be given priorities, which allows a client to further adjust the interface. See section 4.2 for an example.
- When a client subscribes to a service, it provides a terminal profile. The service can use this terminal information to select or construct the best interface for that client.

By using combinations of these techniques, it makes it possible to access service even with exotic interfaces like braille readers.

4.6 Implementation

The RUI protocol has been implemented on top of CORBA, SOAP and HTTP. Several presentation engines have been built to demonstrate the RUI protocol on several devices, such as DVD recorders, PCs and several PDAs, using different communication technologies, such as serial line, InfraRed and Wireless LAN.

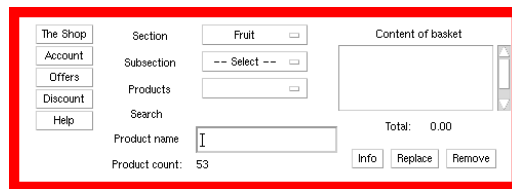
Some services have been defined using the RUI protocol. Due to the incremental updates, service definition is different from the equivalent on the Internet, as the differences between consecutive user interfaces are exchanged instead of the complete new user interface. Initially, this might require more effort, but in return, the service has extra functionality, better performance and better user interaction. With respect to ease of service definition, a comparison with WAP or I-mode is difficult to make, as those protocols also require an extra effort to convert existing services, however, without the improvements in user interaction.

5 Examples of RUI services

The features of the RUI protocol are useful for remote control of embedded systems, but can also be used for providing more advanced mobile services on the Internet. The protocol at the



(a)



(b)



(c)

Figure 3: Three different presentations of the same shop: (a) a graphical presentation, (b) a text-based presentation and (c) a tab-based presentation.

moment has limited provisions for textual layout, but supports interaction with the service in a more advanced way, using incremental updates, incremental user actions and notifications.

In the following examples, a fictitious shop is constructed with product sections, shopping baskets, a search function and special offers. Similar shops are already available on the Internet, although some of the functionality is implemented differently or not available.

5.1 Initial user interface

When the client accesses the shop through the subscription procedure, the shop entrance is presented in a form of an initial user interface. This consists of three parts:

- The *navigation* part, which gives easy access to different parts of the shop, through a number of buttons. If sufficient screen space is available, an advertisement is present as well, which is indicated with a different priority in the UI fragment.
- The *product selection* part, where the user can browse through different product sections and select products. A search function is provided as well.
- The *shopping basket* part, where the collected items are listed, together with buttons to modify the content of the basket.

The three parts are similar to frames in HTML, except that the client can determine how the layout will be, possibly helped by a style sheet from the service. For example, as shown in Figure 3, on a large screen, all the information can be shown, with the navigation part at the top and the two other parts below it. When the capabilities of the screen decrease, the presentation engine can remove the advertisement (if it has been given a lower priority), it can replace images with their labels and it can change the layout, for example, by putting different parts on different tabs in a tab-based interface.

The following interface description is used for initial user interface, as presented in Figure 3 using three different presentation engines. Certain parts are abbreviated with dots.

```

<?xml version="1.0"?>
<!DOCTYPE uifragment PUBLIC "-//...//UI_FRAGMENTS 1.0//EN"
    "ui_fragments-1.0.dtd">
<uifragment>
  <init servicename="Foobar Shop" version="1.0" vendor="ShopMaker"/>
  <container id="shop" label="Foobar Shop" priority="10">
    <container id="navigation" label="The Foobar Shop">
      <button id="main" label="The Shop" image="logo.jpg"/>
      <button id="account" label="Account" image="account.jpg"/>
      ...
      <button id="rules" label="Help" image="Help"/>
      <button id="adver" label="Buy Tippy Now!" image="tippy.jpg"
        priority="-1"/>
    </container>
    <container id="products" label="Products">
      <container id="sections" label="Sections">
        <list id="section" label="Section" informtarget="incrementally">
          <option id="o1" selected="true" label="Fruit"/>
          <option id="o2" label="Vegetables"/>
          <option id="o3" label="Dairy"/>
          <option id="o4" label="Meat"/>
          ...
        </list>
        <list id="subsection" label="Subsection" informtarget="incrementally">
          <option id="o10" label="-- Select --" selected="true"/>
          <option id="o11" label="Berries"/>
          <option id="o12" label="Apples"/>
          <option id="o13" label="Citrus"/>
          ...
        </list>
        <list id="product" label="Products" informtarget="incrementally">
        </list>
      </container>
      <container id="search" label="Search">
        <entry id="find" label="Product name" informtarget="incrementally"/>
        <text id="itemcount" label="Product count:" content="53"/>
      </container>
    </container>
    <container id="basket" label="Shopping basket">
      <list id="content" label="Content of basket">
      </list>
      <text id="total" label="Total:" content="0.00"/>
      <button id="info" label="Info"/>
      <button id="replace" label="Replace"/>
      <button id="remove" label="Remove"/>
    </container>
  </container>
</uifragment>

```

After the standard XML tags, some general service information is given in the “init” tag. Afterwards, the three parts are listed in three containers: the first container provides navigational buttons, the second container lists the sections, subsections and products of the shop, and provides a search function in a separate container, the third container provides the shopping basket, with options to retrieve information, replace a product or remove a product. As no products are selected yet, the content of the shopping basket is still empty.

Different presentations of the shop are given in Figure 3. The actual layout is decided by the client, together with the hints provided through style sheets. In the graphical presentation,

images are used to present the options and the advertisement is shown as well. In the text-based presentation, the advertisement is removed and the list of options is represented by the selected option, which allows access to the other options. In the tab-based presentation, the user interface is divided over tabs according to the use of container elements, where the shopping basket and search function are accessible but not visible.

5.2 Incremental updates

In the shop, incremental updates are used at different locations. When a section is selected, the list of subsections is updated accordingly. Similarly, when a subsection is selected, the list of products is updated accordingly. For example, when the list of subsections of “Fruit” is opened and the “Berries” option is selected, the following UI fragment will update the list of products and the number of products for the search operation:

```
<uifragment>
  <actionresult type="user">
    <setchildren element="product" children="o110,o111,o112,o113"/>
    <changeattribute element="itemcount" attribute="content" changeto="3"/>
  </actionresult>
  <references>
    <option id="o110" label="-- Select --" selected="true"/>
    <option id="o111" label="Strawberries"/>
    <option id="o112" label="Blueberries"/>
    <option id="o113" label="Raspberries"/>
  </references>
</uifragment>
```

When a product is selected, it is added to the shopping basket with an incremental update as well. For example, after the previous action, the “Strawberries” option can be selected from the products list. One possible result of this selection could be to add the selected product directly to the shopping basket. When the product is added to the basket, a new element is added to the list of selected products and the total costs of the basket is updated as well:

```
<uifragment>
  <actionresult type="user">
    <addchildren element="content" children="o111b"/>
    <changeattribute element="total" attribute="content" changeto="1.46"/>
  </actionresult>
  <references>
    <option id="o111b" label="Strawberries, 1.46"/>
  </references>
</uifragment>
```

Note that the application logic for calculating the prices and the total of the shopping basket is located at the service. Note also that in case a client presents the three parts on different tabs, as is the case in the Figure 3(c), the changes to the shopping basket would not be visible to the user immediately.

5.3 Notifications

The shop can use notification for several purposes. A virtual product in the shopping basket might not be available anymore, as the actual product is sold out in the actual shop. With a notification, the shop can indicate this to the user to allow the selection of an alternative product. The following UI fragment, the shop indicates that the selected strawberries are not available anymore.

```
<uifragment>
  <actionresult type="user">
    <changeattribute element="o111b" attribute="label"
      value="Strawberries, SOLD OUT"/>
    <changeattribute element="total" attribute="content" changeto="0.00"/>
  </actionresult>
```

```
</uifragment>
```

The notifications can also be used to update the prizes of products or to replace the advertisements based on shopping behavior, available stock and expiration dates.

5.4 Incremental user actions

The search function of the shop can make use of the incremental interaction model, for example, where every change to the search term results in a user action for the service. This way, the user already receives results while entering the search term, which reduces the number of characters that have to be entered. For a mobile phone user, it is quite important as the input methods are often quite limited. However, network usage should also be taken into consideration, so instead of sending the actual results, only the number of results is returned, which requires less resources and is still very useful. For example, if you are searching for a pineapple, after “pine”, the number of matching products could be three, which might be enough to retrieve the actual results. However, if there are zero results, it is clear that the shop doesn't have the product or that you make a typing error. In any case, the user saves himself some typing. The UI fragments to apply these changes to the user interface are typically very small. After the user action which sends the value of “find” entry field to the service, the returned UI fragment could be

```
<uifragment>
  <actionresult type="user">
    <changeattribute element="itemcount" attribute="content" changeto="3"/>
  </actionresult>
</uifragment>
```

The network usage can be further decreased by applying dedicated XML compression, as used by the WAP protocol, which reduces the size of the XML message considerably.

6 Conclusions

The Remote User Interface protocol is useful for both mobile services and in-home networks. Due to the incremental updates, the processing requirements on the client are reduced and the communication requirements are less for similar functionality. The protocol provides several mechanisms to be device independent, both for the device that provides the service as well as the device that acts as the client. The protocol allows notifications and the service can indicate how tight the communication with the client should be to get the best user experience.

It is the first solution for mobile services, which solves the main problems of existing technologies, like WAP and I-mode. That is, it improves service interaction, it reduces network traffic for similar functionality, allows notifications without user interface glitches and improves the user experience.

To our knowledge, no middleware standard for in-home digital networks, other than HAVi, does provide similar facilities as defined in the Remote User Interface protocol. If the protocol would be available on a mobile phone, it opens the home environment for devices which are network enabled, where the mobile phone can be used to control them.

Acknowledgments

Special thanks to Tom Sutera, Bernard van Vlimmeren and Johan Lukkien for the interesting discussions on the RUI protocol and its possible applications.

References

- [1] Bluetooth SIG - Public Specifications. Online: <http://www.bluetooth.org/specifications.htm>
- [2] Compact HTML for Small Information Appliances. Online: <http://www.w3.org/TR/1998/NOTE-compactHTML-19980209>

- [3] Composite Capabilities/Preferences Profile Working Group Public Home Page. Online: <http://www.w3.org/Mobile/CCPP/>
- [4] CLDC and the K Virtual Machine (KVM). Online: <http://java.sun.com/products/cldc/>
- [5] Walter Dees. Device Independent User Interfaces. Online: <http://www.w3.org/2002/02/DIWS/submission/Wdees-position.htm>
- [6] GSM World - GPRS Platform. Online: <http://www.gsmworld.com/technology/gprs/index.shtml>
- [7] HAVi - Home Audio/Video Interoperability. Online: <http://www.havi.org/>
- [8] HyperText Markup Language Home Page. Online: <http://www.w3.org/MarkUp/>
- [9] Fielding, et. al. HyperText Transport Protocol - HTTP/1.1 RFC 2616, June 1999 Online: <http://www.ietf.org/rfc/rfc2616.txt>
- [10] DoCoMo Net - i-mode. Online: http://www.nttdocomo.co.jp/english/p_s/imode/index.html
- [11] i-mode service guideline. Online: http://www.nttdocomo.com/images/disclaimer/i-mode_service_guideline.pdf
- [12] IrDA Specifications. Online: <http://www.irda.org/standards/specifications.asp>
- [13] J2ME - Java 2 Platform, Micro Edition. Online: <http://java.sun.com/j2me/>
- [14] DVB-MHP - Digital Video Broadcasting Multimedia Home Platform. Online: <http://www.mhp.org/>
- [15] J2ME Mobile Information Device Profile (MIDP) Online: <http://wireless.java.sun.com/midp/>
- [16] SMS Forum. Online: <http://www.smsforum.net/>
- [17] UMTS Forum. Online: <http://www.umts-forum.org/>
- [18] WAP Specification Releases Online: <http://www.wapforum.org/what/technical.htm>
- [19] IEEE 802.11 Wireless Local Area Networks Online: <http://grouper.ieee.org/groups/802/11/>
- [20] XHTML 1.0: The Extensible HyperText Markup Language. Online: <http://www.w3c.org/TR/xhtml1/>
- [21] ZigBee Working Group. Online: <http://www.zigbee.org/>