WAP User Interfaces

Titti Kallio and Toni Komu
Sonera Corporation, Mobile Operations,
P.O.Box 970, FIN-00051 Sonera, Finland
titti.kallio@sonera.com toni.komu@sonera.com

Regardless of the WAP Forum standardisation, user interfaces in different WAP devices differ from each other. In fact, the differences between devices are so great that you can use very few user interface features while developing services, i.e. when developing the services to be used with *all* WAP phones.

In this presentation we present three ways to cope with the situation: 1. To apply the principle of the lowest common denominator, 2. To design separate applications for different devices or device groupings, 3. A converter.

1. THE DIFFERENCES BETWEEN DIFFERENT DEVICES

In the Finnish market there are four main WML browsers according to (presumed) market shares: Nokia, Ericsson, Phone.com's UP.Browser (e.g. Motorola, Siemens) and Microsoft's Mobile Explorer microbrowser (e.g. Benefon).

Next we present some of the differences between these devices. The findings are based on heuristic evaluation by usability experts and on several usability tests with different WAP services and devices.

Input fields: The web analogy among WAP users is very strong (based partly on marketing slogans like: "Now you can use web in your GSM phone!"). However, a web page does not start to show on a screen from the input field area forward (but from the actual beginning of the page forward) but UP.Browser will always focus on the first input field area on a page (card) and not on the beginning of the page. This confuses the user. Therefore the input fields must be separated into their own pages (cards).
<select> feature: This is found difficult to be used by users. E.g. many phones do not show check boxes (<select> multiple) and radio buttons (<select> single), so users do not get the point here. Select lists without check boxes or radio buttons are mixed easily with link lists by the users.
Perceived amount of information on a screen: The perception depends on the actual screen size, pixel size, resolution, and the fonts being either absolute or relative. With some handsets you can present so little information on a screen that it is worthless to use any pictures, logos, animations, etc.
Software versions of devices may work in a different way on a user interface level, too.
WAP gateway: A non-device factor, which also affects the user interface and programming, is the WAP gateway and the version used: e.g. some gateways do not support the Scandinavian alphabet and force the programmer to use hexacoded characters.

Basically, what remains when you find the lowest common denominator between handsets is very little. If you want to design services which work smoothly with all possible WAP phones, the specific user interface features you have left are *links* and *input fields* and nothing else.

2. ONE APPLICATION FOR ALL HANDSETS: THE LOWEST COMMON DENOMINATOR

Jakob Nielsen states that designers of WAP services need to optimise each service for each of the different telephones (to be able to squeeze every last bit of usability out of it) [1]. That is ideal but when the pressure is high for service providers and operators, and WML programmers are still few, you can seldom do the optimising.

When you have several WAP applications to design and not too much labour force for programming, maintenance etc., then you will probably develop device-independent applications.

A good idea is to launch a WAP style guide which first informs the designer about the basic WAP user interface rules [2] and then states the lowest common denominator (i.e. links and stand-alone input fields and their use).

3 DIFFERENT VERSIONS FOR DIFFERENT HANDSETS

If you have much labour force you can make device-dependent applications, i.e. several separate versions of an application. At the moment, this means e.g. versions for:

- Nokia
- Ericsson
- UP.Browser
- MS microbrowser
- PDA or communicator type WAP devices (one on the market, more to come soon).

The labour resources will go for programming and maintenance. There is no need for user interface style guides here, because manufacturers provide their own style guides (see manufacturers' web sites and the WAP Developers' Forums there). However, these style guides are more or less useless when device-independent applications are developed.
In this solution a different version of the same service is deployed based on the handset's capabilities. This can be implemented, for example, by designing the service so that the first page the user sees when entering a service is actually a program that is able to check the handset used and hence to redirect the user to the address that is suitable for his/her handset. In services that consist of only one page this is quite an easy solution. However in many services the service consists of many different pages. If the service includes e.g. ten pages and we want to support four different kinds of devices, we have in the worst case 40 pages we have to take care of and update when the service's content changes. That means quite a lot of extra work.

Another solution is to design the service so that the content of the service is generated only when needed. This means that the service does not consist of static files, but of application(s) that produce different content based on the service state and especially based on the handset type. So the application(s), which could be for example cgi-scripts or servlets, produce different content based on handset type. This helps, but there is still extra work when a new service is designed.

4. A CONVERTER

The third solution is to concentrate the resources for making a converter. The converter handles the handset recognition and modifies the service content accordingly. The converter could be for example modified WWW proxy software. In order for the proxy software to be able to modify the content, the service platform should be configured so that the service content is fetched through the proxy software.

In order to make the content modification procedure simple enough so that no higher-level artificial intelligence is needed, the source service content should follow certain principles. In practice this means that the services to be used through the converter should be designed using a WAP style guide. Of course we can try to convert any content whatsoever, but then the conversion software must be quite complex. However, having to use a style guide is no extra work because the service designer probably follows some style guide anyway. And now when the service is correctly designed it can be used with different handsets and it is always optimised for the handset in use.
One thing to remember here is that the size (in bytes) of the content may change as a result of the conversion procedure. The converter can handle this by splitting the deck if necessary. In some cases this could change the service logic, but on the other hand in some cases this can also be of big help. Different devices can handle different amount of content in one deck. So in some cases the service might not work with some devices at all without splitting the deck into two decks.

We recommend this solution as the easiest way for programmers and as the way in which all the possible user interface features of separate handsets can be benefited from. A service can be designed so that it uses all the best features of a given handset, probably the most popular one, and with the help of the converter other handset users get the best possible service too.

Also, we would like to urge WAP Forum to start the standardisation work on the user interface consistency as soon as possible.

5. REFERENCES

[1] http://www.useit.com/alertbox/20000709.html
[2] Schmidt, A., Schröder, H. & Frick, O. *WAP – Designing for Small User Interfaces* in Proc. CHI2000 Extended Abstracts, 2000.

