Building 'Usable' WAP Applications

Luca Passani, Cell Network AS

Usability is a term that indicates the degree of user-friendliness of a system. A *usable* system is one that lets its users complete tasks in a reasonably easy way. Assessing the value of a system's user interface has become increasingly important with the growth of computer use — so much so that there is a whole field of computer science (Human-Computer Interaction, or HCI) that deals with building usable systems.

Building usable WAP applications is not simple. Wireless devices have many limitations, and the average user of a WAP application is not technically oriented (and possibly not even used to the Internet). Finally, the interpretation of WML varies greatly between devices from different vendors. This poses an extra challenge to good usability.

If you're a web developer, you might think that this situation closely resembles the browser war we're still witnessing on the Web. And you are right.

This paper looks into these issues in depth, and discusses methods for overcoming some of the problems you'll encounter when attempting to code or convert applications for use on a variety of different browsers. This is done with reference to two particular microbrowser examples:

- The Nokia 7110
- The Phone.com UP.Browser

The paper assumes familiarity with the WAP architecture and WML programming.

Usability

For the purpose of this presentation, we can define usability as follows:

> **Usability refers to the ease with which users of an application can perform the operations and complete the tasks that the application is supposed to help them achieve.**
>
> **If users perceive your application as being easy to use, straightforward, and forgiving, then you have a usable application.**

Limitations of Wireless Devices

Let's review the limitations of WAP devices here:

- **Small screens:** In general, WAP devices are tiny. Those accustomed to web browsers will find navigating with a WAP phone a real pain.
- **Limited input facilities:** Most wireless devices lack a keyboard that is anything like a traditional QWERTY PC keyboard. Simple, mass-market, consumer-class data input technology that does not depend on a keyboard has yet to be invented.
- **Limited processor power and memory:** WAP browsers are simple and unforgiving.

- **Limited bandwidth:** At this stage, WAP devices have very little bandwidth available when compared to PCs. In Europe, users can count on a speed of 9600 bps (bits per second) as of April 2000. The introduction of GPRS may improve the situation slightly by the end of 2000.
- **Lack of graphics:** Or at least, very limited support for them. Icons and graphics can go a long way towards helping the user in complicated situations.
- **Limited deck size:** A deck can contain only a limited amount of information.

These limitations have serious implications for the way you design your WAP application.

WAP Users

WAP users are not sitting in front of a PC. They are on the move, on their way to a meeting, or in a crowded train. Sometimes they're under pressure. Building usable WAP systems is not straightforward, and goal when doing so should be to make it *as simple to use as possible*. While this is true for any application, it's an absolute must in the context of WAP. WAP users are subject to many distracting events in the environment that surrounds them, and this adds to the input/output limitations of WAP phones described above.

In three years, it is estimated that there will be half a billion WAP-enabled mobile phones around. This means that potential WAP users will outnumber conventional Internet users by far. One implication of this is that, in general, you cannot assume that the users of your application are also conventional Internet users.

WML Interoperability Issues

WML delivers content and user interfaces across very different kinds of devices. The various browser implementations render WML in different ways, and this will affect the usability of our applications. This paper will discuss this issue in detail.

A usable WAP application should never confuse users, in that users should ideally be able to find the most obvious operations intuitively — just one click away. Unfortunately, if you tweak your application to be more usable on a particular device, the chances are that usability will suffer on other devices.

This awkward situation is not simple to solve. In the worst case, implementing multiple versions of an application (one for each family of browsers) might be the best option you have. Learn about usability, and think about what can best be achieved for *your* specific applications in *your* specific context.

Different Devices

Fine-tuning usability necessarily implies getting involved with the idiosyncrasies of each device you intend to support. There's a general rule that you should always keep in mind:

> **Applications developed for small displays tend to look and work fine on large displays. Applications developed for large displays tend to look and work very badly on small displays.**

If, while developing your applications, you target the smallest devices that you intend to support, you will find that in most cases you automatically target larger devices too. PDA-like devices will be especially well covered, as they support

hyperlinks and features handled by <do> elements (features that are problematic on very small screens) very well.

At the end of the presentation, I'll focus on two very common microbrowsers: the Nokia 7110 and the UP.Browser family.

General Usability Guidelines

It is important to develop a set of concrete guidelines that you can apply when designing a WAP application. Here are the most important questions you should ask yourself:

- Is the application easy to learn?
- Is the application efficient to use?
- Are unusual operations easy to remember?
- Do users get stuck when there are errors?
- Are users likely to be frustrated by their attempts to use your application?

Answering questions like these will put you on a good track to working out how usable your application is. Let's see some general rules for building usable applications.

- *Top 20% of functionality:* When porting an existing application to WAP (an HTML page, for example), you should identify the main activities that users will be interested in using while on the road. Porting parts of the application that are not commonly used will be more expensive for you, and will degrade the overall usability of the system, because of the extra levels and navigation paths you are bound to introduce. In the case of new applications, think of this rule as "refrain from implementing functions that the majority of users won't use on the road".
- *Rate user activities:* Try to identify the main activities that the majority of the users will perform, and build your application in a way that will let users perform these activities in the fastest way possible. You should make sure that the most common activities are intuitively available for all users.
- *Design it as a tree structure:* Lay out a hierarchical tree of activities. Users should enter the application at the root and be able to perform any of the available activities through some path starting at the root. Each level of the tree should be laid out according to the likely popularity of the activities it contains.
- *Minimize data entry:* Most phones only have a phone keypad. Your application should require textual data entry only when absolutely necessary. Similarly, don't require users to remember codes or other information when visiting your application — try to remember things for them. In addition, the input mode of the terminal should be set to support the expected format for the data that users will enter. This can be achieved through input masks.
- *Text should be terse:* Short, polished, and informative text is vital to guide users.
- *Always implement 'back' functionality:* All users like to explore when confronted with a new application, and a 'back' function should be available to them at all times. But be careful: users should go back to a logical and consistent place in the application, which is not necessarily the previous card.

- *Consistency is very important:* Applications can often require users to perform the same activity (or very similar activities) in different parts of the application. It is important that you deploy a consistent set of metaphors that will help users find their way around easily.
- *Push:* Real-time information is a key piece of functionality that will give extra value to WAP. Unfortunately, push is not part of the WAP standard yet, but we should see it implemented in WAP 1.2. In spite of this, several proprietary possibilities to do push through WAP already exist, and you should exploit them if possible. Don't give up on the extra value your application can acquire through push.
- *Be prepared to test:* If you are deploying a WAP application that is even moderately complex, you should be ready to build prototypes as early as possible in the development process. Find a non-technical person and let them use your application on real phones.

Identifying Activities

When thinking about the main functionality of your application, start by rating the user activities into categories according to how often users perform them. For example:

- Activities that most users perform most of the time
- Activities most users perform occasionally
- Activities that specialized users perform once in a while

We give some more detailed groupings below. Identifying those activities is the basis for breaking down the navigation flow and optimizing the navigation path required to perform the main activities.

- Think of the tasks required to achieve the object of each activity
- Order the tasks by importance
- Lay out your user interface

For each activity, you should understand how users expect to perform them, according to models users are familiar with. This could mean similarity with corresponding PC or phone functions, or with the way users perform the activity in their work.

Classifying activities in the following groups will help you map the design directly into WML:

- *Required Activities* (compulsory activities for all users): These are activities that all of the application users will have to perform. One typical such activity is configuring your e-mail address and POP server when using an e-mail application for the first time. Configuring access to a different gateway is another example. It goes without saying that user-friendliness degrades unacceptably if operations like this are requested each time the user accesses an application. *Avoid required activities or minimize them as far as possible.* In many cases, you can uniquely identify each device and recall user preferences and configuration parameters automatically. Use this possibility and avoid requiring users to log in.
- *Main-path Activities* (high use activities): These are the activities that 80% or more of your users are likely to perform. Such activities should be easily and intuitively available, without any learning curve or unnatural interaction path for the inexperienced user. Performing main-path activities should be as easy as possible (always one click away).

- *Semi-main-path Activities* (high use by a large segment of, but not all, users): These are activities that many users (but not the majority) perform often. Keep access to semi-main-path activities as simple as possible, but make sure that they are not an impediment for users who do not perform them.
- *Side-path Activities* (activities used occasionally by most users): These are activities that 80% of users will perform, but only 20% of the time they use your application. Replying to e-mail is one such activity. It makes sense to implement access to side-path activities in a menu that is not immediately accessible.
- *Rare-path Activities* (activities never used by most users): These activities are there to support power users. If you identify rare-path activities, consider the possibility of removing them altogether. It's better to have a simple system than one with a lot of obscure options.

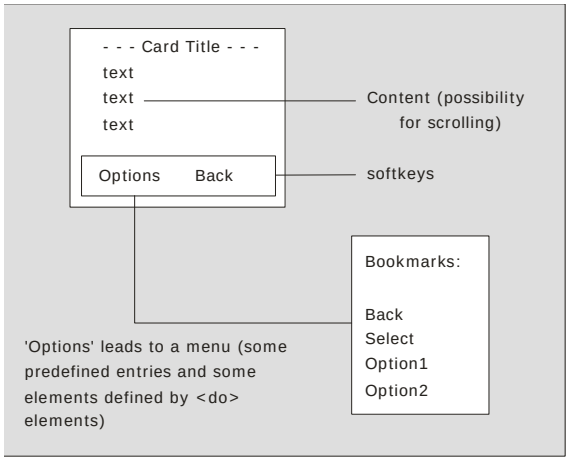
The classification of activities into main-path, semi-main-path, side-path, and rare-path gives developers an indication of how they should implement navigation *to* the different kinds of activities. Once a user is *inside* an activity however — whether it's a frequently used operation or not — you should support that activity like a main-path activity.

Guidelines for Specific Browsers

Good usability requires that you customize your application for each specific browser. I will talk about usability issues on the UP.Browser, and the browser of the Nokia 7110. This should help you to evaluate usability issues for other devices as well.

Guidelines for Nokia 7110

What follows is an overview of the interface for the Nokia 7110 WAP browser:



Entering Data on the 7110

If you have an `<input>` or a `<select>` element in your code, you need to make it clear to the user how they can submit the data in order to move on to the next logical step.

The only sensible thing to do in these cases is to provide a link that will keep users going with one click:

```
<p>
   Name please:
   <input type="text" name="searchkey" value="" />
   <a href="search.asp">Submit data</a>
</p>
```

If you use <do> elements, the 7110 browser will interpret it by adding an extra element in the menu triggered by the left softkey. This is not good. Users will not immediately see this, and so would be at a loss for what to do after they've entered the data.

Forms

There are two types of form: wizard forms and elective forms.

Wizard forms let users insert data one bit at a time. Each card contains an input element. It's a good idea to use Wizards when you can, since they let users focus on entering data rather than navigation.

Forms implemented through a single card containing multiple input fields are called **elective forms**.

It goes without saying that on the 7110, you should provide a link to let users move from one card to the next, rather than using <do> elements.

Menu Navigation on the 7110

Menu navigation is a straightforward way to let your users access all the different parts of your application, laid out in a tree structure.

The best way to implement this for a Nokia 7110 is by building a menu as a list of anchors. For example:

```
<a href="#band" title="find">Artist/Band</a>
<a href="#song" title="songs">Title/Song</a>
<a href="#top" title="top">Top 20</a>
<a href="#new" title="new">New Releases</a>
<a href="#conc" title="live">Concerts</a>
```

Forcing users to do a lot of scrolling is not a good idea. If you need to display more than nine or ten items, you should split your links over several cards or decks.

Backward Navigation

The right softkey of the 7110 can only be used as a back button, and is labeled as such. By default, it has no action at all. Some sub-versions of the 7110 allow reprogramming of the back key with <do type="prev">. Unfortunately, this is not the case with all phones. The way to get the phone to do what you want is:

```
<card id="mycard" onenterforward="#nextcard"
      onenterbackward="http://logical_back">
```

Unfortunately, this has the side effect of spoiling the history stack.

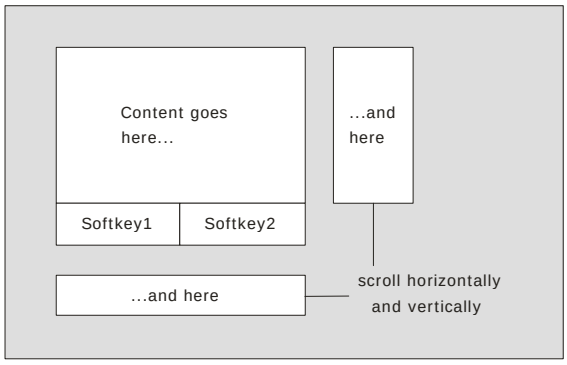
Never provide a label for the <do type="prev"> element. The 7110 will remove the Back label from the right softkey, and instead create a new entry in the menu accessed through the left softkey. This confuses users who expect to find the back key in a well-known position.

Guidelines for UP.Browser

The Phone.com browser has a different interface from the Nokia 7110. The general idea behind the UP.Browser is that a <do> element is mapped directly to a softkey whenever possible.

Users will only ever be one click away from performing a main-path activity. Other activities will also be reachable in a simple way. This is in sharp contrast with the Nokia 7110.

<do type="accept"> elements are normally used to support main-path activities. They are also called ACCEPT buttons. Other activities are supported through <do type="options"> elements (OPTION elements).



If you have a single main-path activity, for example, the best you can do is to associate it with an ACCEPT button, which is always bound to a softkey (usually mapped to Softkey1). This way, your users are only one click away from the activity they are most likely to perform.

If you have a single <do type="option"> element (or OPTION button), this will be mapped to Softkey2. In the case of multiple OPTION buttons, Softkey2 will display the label menu, which leads to a pseudo-card that allows you to select the option from a simple list (similarly to Softkey1 on the 7110).

Some UP.Browser phones support three softkeys. In that case, you can afford an ACCEPT task and two OPTION tasks without the need to go through an extra menu.

In the diagram, you can see softkey1 on the left and softkey2 on the right, but this is not always the case. Softkeys occupy different positions in different implementations of the UP.Browser.

Entering Data on the UP.Browser

Unlike the Nokia 7110, the UP.browser does not require the presence of a link after an input element in order to be usable. A <do> element will work wonders there:

```
<do type="accept" label="Send">
   <go href="receive.asp" />
</do>
<p>
   name please:
   <input type="text" name="username" value="" />
</p>
```

The data can be posted with just one click (on Softkey1). If you insert a navigation link after the input element, as you should do with the 7110, the mechanism will still work, but usability will degrade. The construct that follows will require no fewer than three clicks to post the data from the UP.Browser.

```
<p>
   name please:
   <input type="text" name="username" value="" />
   <a href="receive.asp" title="Send">Submit data</a>
</p>
```

Forms

Use <do> elements to navigate from one card to the next; otherwise, the same guidelines for forms apply as did for the Nokia 7110.

Menu Navigation on the UP.Browser

If you implement menu navigation the same way as you would on the 7110 (with a bunch of links), the UP.Browser will work satisfactorily. Unfortunately, by doing so you'll lose a feature that the UP.Browser supports to facilitate navigation for slightly advanced users.

To demonstrate, the code below implements menu navigation in the optimal way for the UP.Browser:

```
<select>
   <option onpick="#band" title="find">Artist/Band</option>
   <option onpick="#song" title="songs">Title/Song</option>
   <option onpick="#top" title="top">Top 20</option>
   <option onpick="#new" title="new">New Releases</option>
   <option onpick="#conc" title="live">Concerts</option>
</select>
```

Here is how the UP.Browser renders the code:

The numbers on the left of the screen are shortcuts. If you implement navigation menus this way, UP.Browser users will not be required to scroll to the menu item of choice, since they can press the number on their keypads and trigger the `onpick` event for the corresponding menu choice.

The Generic Approach

The basic idea behind the 'generic browser' is to identify a subset of WML that both the browsers used above interpret in a more or less usable way.

Guidelines for the Generic Browser

The generic browser is an attempt at defining the lowest common denominator features of WML that work acceptably well on the two main browsers.

Entering Data on the Generic Browser

The 7110 requires the use of links. On the other hand, it's better to use a <do> element on the UP.Browser, as a link would introduce two extra steps. Using <do> elements on the Nokia 7110 is not acceptable — but there's nothing to prevent you from using both a link and a <do> element in the same card, as shown in the example below (a modified version of the example you saw earlier):

```
<card id="card1" title="mycard">
   <do type="accept" label="Send">
      <go href="receive.asp" />
   </do>
   <p>
      name please:
      <input type="text" name="username" value="" />
      <a href="receive.asp" title="Send">Submit data</a>
   </p>
</card>
```

In the UP.Browser, users will be able to navigate with the Accept button after they have acknowledged the text they wrote in the form. As far as the Nokia is concerned, users will get a link after the input form — business as usual for them.

Menu Navigation on the Generic Browser

No matter how useful the shortcuts supported by the `option-onpick` pair in the UP.Browser, the lack of support for this feature on the 7110 means that you should implement menu navigation with a list of links. The list of links works well enough on the UP.Browser, after all.

Conclusion: Is Generic Navigation Good Enough?

At this point, you might wonder if the generic approach is good enough. Generally speaking, the answer is 'No!'

The generic approach forces you to discard all of the really good things that each microbrowser has to offer, and to settle for an alternative that proves to be mediocre in both environments, despite being useful in some cases.

The alternatives are the creation of multiple applications, or support for multiple versions through XML/XSLT transformations. For more information about these approaches, see http://www.webtechniques.com/archives/2000/03/passani/, and check out the XML/XSLT session at this conference, by Wei Meng Lee.

Resources

General application style guidelines for WML services:
http://updev.phone.com/dev/ts/technotes/userfriendly.pdf

Application style guidelines for WML services in GSM markets:
http://updev.phone.com/dev/ts/doc/style/gsm900-1800.pdf