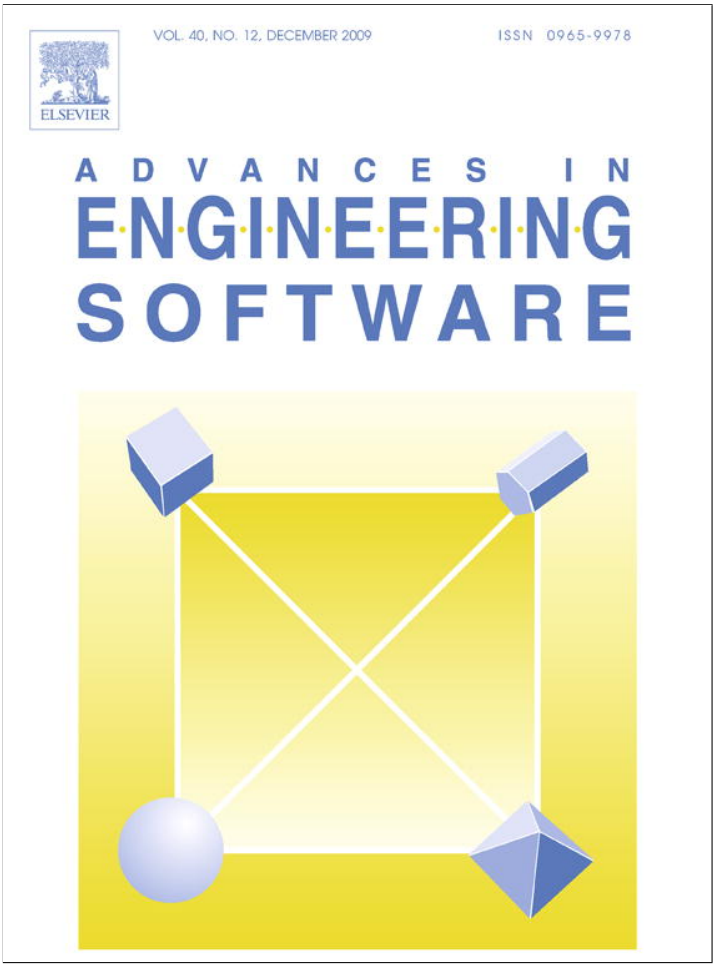

Advances in Engineering Software 40 (2009) 1318–1328

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

# Design patterns for user interface for mobile applications

Erik G. Nilsson *

*SINTEF ICT, Postboks 124, Blindern, N-0314 Oslo, Norway*

ARTICLE INFO

*Article history:*
Received 22 September 2008
Received in revised form 19 November 2008
Accepted 19 January 2009
Available online 25 April 2009

*Keywords:*
User interface design patterns
User interfaces for mobile applications
Patterns collection
Mobility
Design guidelines

ABSTRACT

The topic of this paper is a collection of user interface (UI) design patterns for mobile applications. In the paper we present the structure of the patterns collection – the patterns are suggested solutions to problems that are grouped into a set of problem areas that are further grouped into three main problem areas – a structure which is valuable both as an index to identifying patterns to use, and it gives a fairly comprehensive overview of issues when designing user interfaces for mobile applications. To show the breadth of the patterns collection we present six individual problems with connected design patterns in some detail – each coming from different problem areas. They represent important and relevant problems, and are on different levels of abstraction, thus showing how patterns may be used to present problems and solutions on different levels of detail. To show the relevance and usefulness of the patterns collection for usability professionals with a mixed background, we present some relevant findings from a validation of the patterns collection. In addition to verifying the relevance and usefulness of the patterns collection, it also shows both expected and surprising correlations between background and perceived relevance and usefulness. One important finding from the validation is an indication that the patterns collection is best suited for experienced UI developers wanting to start developing mobile UIs. Using a patterns collection for documenting design knowledge and experience has been a mixed experience, so we discuss pros and cons of this. Finally, we present related work and future research.



## 1. Introduction

In the UMBRA and FLAMINCO projects, we have developed a set of design guidelines to aid developing more user-friendly applications on mobile devices (PDAs/SmartPhones), giving practical advices for how to solve various problems that arise when designing user interfaces on mobile devices. The main part of these design guidelines is a collection of user interface design patterns for mobile applications [8] (the patterns collection is available at http://www.flaminco.net).

The patterns collection suggests solutions to a number of problems that may occur when designing such solutions. These problems are grouped in three main problem areas

1. Utilizing screen space.
2. Interaction mechanisms.
3. Design at large.

Each problem is presented using a design pattern approach [6,1,18] where the problem itself, general guidelines for solving the problem and a number of relevant design patterns are discussed. In the presentation of possible solutions and design patterns, pros and cons of different solutions are discussed, and examples of good solutions are given where appropriate. The "sources" for the contents of the patterns collection are problems identified in the requirements elicitation phase of the UMBRA and FLAMINCO projects. The problems stem from the pilot projects conducted by the project partners, from general experience gathered by the project participants when developing mobile applications, from literature surveys [17,13,14], and from experiences with using different applications on mobile devices. The problems presented in the patterns collection tries to span both the most common problems when designing mobile UIs today, and future challenges like multimodal (including use of gestures) as well as contextual and adaptive UIs. Despite this, the main focus of the patterns collection is professional application in general and forms-based UIs in particular. The patterns collection is constantly being refined and further developed, especially into future challenges and new types of UIs.

## 2. Main problem areas

The design patterns presented in this document follow a given structure. On the top level, they are grouped into the *three main problem areas* mentioned in the introduction. Within each of these three main problem areas, a small number of *problem areas* are defined. In Table 1, a brief description of the main problem areas and

* Tel.: +47 22 06 73 00; fax: +47 22 06 73 50.
*E-mail address:* egn@sintef.no

0965-9978/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.advengsoft.2009.01.017

**Table 1**
Main problem areas and problem areas structuring the patterns collection with their connected problems.

| Main problem area | Problem area | Description and individual problems (with connected UI design patterns) |
|---|---|---|
| Utilizing screen space | | Focus on problems connected to the limitations regarding screen space on mobile equipment |
| | Screen space in general | Addresses problems connected to different layout challenges on small screens<br>• Presenting elements in lists<br>• Principles for grouping information<br>• Mechanisms for grouping information<br>• Mechanisms for packing information<br>• **Horizontal scrolling** |
| | Flexible user interfaces | Addresses problems connected changing the layout dynamically at runtime because the either the information that should be presented and/or the environment in which to present the information change<br>• Presentation based on models or data – how to do this on a small screen<br>• **Handling crowded dialogs when software keyboard is shown and hidden**<br>• Versions and variants – dynamic and configurable user interfaces on small screens<br>• User interfaces that facilitate switching between portrait and landscape mode<br>• User interfaces that are able to run on equipment with different screen size |
| Interaction mechanisms | | Focus on problems connected to the limitations regarding interaction mechanism on mobile equipment |
| | Handling input | Addresses problems connected to entering information more efficient and/or with less probability for entering incorrect information<br>• **Mechanisms for entering text**<br>• Order entry<br>• Mechanisms for entering numerical data<br>• Multimodal input<br>• Controlling input cursor from an application |
| | Not using the stylus | Addresses problems connected to entering information in situation when it is not possible/convenient/desired for the user to use a stylus<br>• **Interacting with applications without using stylus**<br>• Retrieving data from a database without using keyboard |
| Design at large | | Focus on problems connected to design principles for user interfaces on mobile equipment |
| | Guidelines | Addresses design guidelines on different levels of generality<br>• Design guidelines for data base applications, including automatically generated user interfaces<br>• Standard features that should be available in an automatically generated prototype<br>• **Design that supports branding, is aesthetic, and utilize screen space optimally**<br>• Solutions for searching large amounts of data<br>• Visually coding of entry fields to mark editability (must, may, may not)<br>• Standard solutions vs. usable tailored solutions |
| | "Difficult to understand" | Addresses problems connected to providing understanding of what is happening when a mobile application performs functionality that can be difficult to understand for the end user – usually functionality that is specific for mobile applications<br>• User interaction during synchronization<br>• User interaction for log-on/log-off<br>• **User interaction during waiting for long-lasting operations to complete** |

problem areas are given, as well as the identified problems within each of these problem areas. The problems presented in bold font are presented in the next section.

## 3. Selected problems with connected design patterns

In this section we present brief versions of a selection of the design patterns for six of the identified problems. The selection is partly based on their importance, partly to show examples of the different parts of the patterns collection, and partly to show problems and patterns on different levels of abstraction. For most of the problem presented below, we also provide some general guidelines in addition to the selected design patterns.

### 3.1. Horizontal scrolling

Horizontal scrolling is usually worse than vertical scrolling. One of the reasons for this is that information on the same line usually is closer connected than information on different lines. This means that the user looses more context information when scrolling horizontally than vertically.

Using a selection and projection analogy, avoiding or handling horizontal scrolling (especially for lists) is often a question of projection, i.e. which attributes that are to be shown and in which sequence. If the projection is wrong (compared to the user's need), horizontal scrolling is often needed.

#### 3.1.1. General guidelines

One solution is to *optimizing the sequence and size of the attributes shown*.

Looking specially at avoiding horizontal scrolling, *changing the layout* may solve this without using any of the packing solutions. This will usually increase the need for vertical scrolling.

Changing the screen orientation (see the design pattern below) may be considered being a fairly simple way of changing the layout of a UI, which in many cases will not cause any layout changes at all. In other cases it may cause a need for moving or regrouping some fields. This is an example of what we call a *simple redesign* – usually restricted to changing the layout of a given set of fields. This is an inexpensive solution to the problem of horizontal scrolling, but may not be sufficient in some cases, either because the UI becomes too cluttered or requires very much vertical scrolling to avoid the horizontal one.

A slightly more radical solution is to perform a *minor redesign* of the UI. By this we mean to do more than just a layout change, but keeping the style (usually forms-based). In the case of a forms-based UI this typically implies changing the controls that are used in the UI to more space conservative ones, or to use controls that may replace a number of other controls. A medium redesign may also imply redesigning which information that is to be presented in which window (often resulting in a larger number of windows). This is also a fairly inexpensive solution (depending on the degree of changes), which may work in more cases than just a simple redesign.

The most radical solution is to perform a *major redesign* of the UI. By this we mean to change the style of the UI, e.g. from a forms-based UI to one using a more visual metaphor. This is usually more difficult and may be much more expensive than a simple or medium redesign, but may also result in a much more user-friendly solution. Such changes may not only reduce the need for horizontal scrolling in a single window, it may reduce the total number of windows in the application.

*3.1.2. Design pattern: Change the screen orientation*

*3.1.2.1. Use when.* If this is an option on the platform, it will by default reduce the need for horizontal (and increase the need for vertical) scrolling, as illustrated in Fig. 1. An important choice if this solution is used is whether the user should be given the opportunity to switch between landscape and portrait, or if only landscape should be available. The first choice imposes a number of new problems. The latter choice is easier to realize, but offers less flexibility for the user, and may reduce which devices/versions of the operating system that may be used to run the application.

*3.1.2.2. How.* Provide a version of the UI in landscape format, alone or in addition to a version in portrait format.

*3.1.2.3. Why.* Horizontal scrolling should be avoided in all UIs, but is probably worse on mobile devices than on larger displays, partly because the amount of context information is larger when the screen is larger. Also, on a larger screen, it is usually possible to make the window larger to decrease the need for horizontal scrolling. And even worse, because the screen is smaller, the need for horizontal scrolling occurs more often.

*3.2. Handling crowded dialogs when software keyboard is shown and hidden*

On PDAs without keyboard, a common solution for entering text is to show a software keyboard on the bottom of the screen where the user can enter text using the stylus. The area in which the keyboard is shown may already be used by the application. This means that the application have less room for its "normal" interaction.

The main problem when designing a user interface that should be able to handle that the software keyboard comes and goes is how to resize the dialogs. Resizing may just imply adding/adjusting a scroll bar, but often some other adjustments are needed to avoid some parts of the dialogs becoming invisible.

The severity of this problem is depending on the type/style of the user interface. If the UI only contains an arbitrary text, adding or adjusting the scrollbar is a sufficient solution. A forms-based UI may be much more difficult to resize. The same may be the case for a more visual presentation, depending on whether the visualization is tailored for the screen size or not. Handling tab folders and buttons that are placed on the bottom of the screen is also a challenge.

The design patterns presented in this problem focus on forms-based UIs, because it is mainly for this type of UIs that this problem occurs.

*3.2.1. General guidelines*

A buffer solution (see design patterns below) may also be used with two or more large UI controls sharing the amount of size reduction to be applied. Generalized, this solution ends up as *dynamic resizing* of the controls in the UI. This may be done using two different approaches. The first is to decide a resizing rule for each window and apply that as tailored code for each window. The second is to have a general layout adjustment algorithm doing it for all windows.

*3.2.2. Design pattern: Add or adjust scroll bars*

An obvious and simple solution to this problem is to add or adjust scroll bars when the keyboard appears, as illustrated in Fig. 2. The other solutions presented below are solutions where the need for adding scroll bars are removed or reduced.

*3.2.2.1. Use when.* This solutions should be used when a simple and inexpensive solution is sought, or when none of the other patterns are useful.

*3.2.2.2. How.* Provide two sizes of the view for the dialog.

*3.2.2.3. Why.* The solution is simple and inexpensive, yet easy to understand.

*3.2.3. Design pattern: Let the keyboard cover part of the UI*

How "bad" this solution depends on what is placed on the part of the screen that will be covered by the keyboard, as illustrated in Fig. 3.

*3.2.3.1. Use when.* If this part is occupied by output fields, the solution may work fine as long as the keyboard is removed when not needed. If this part of the screen contains important input fields or tab folders the solution is useless.

*3.2.3.2. How.* This solution is in essence "doing nothing".

*3.2.3.3. Why.* The solution is simple and inexpensive, though not always very user friendly.

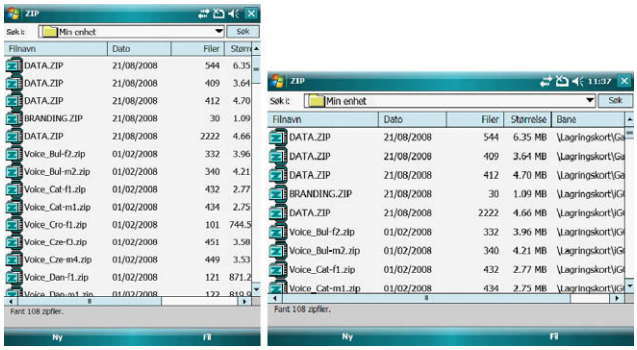

**Fig. 1.** Changing screen orientation to reduce horizontal scrolling.


**Fig. 2.** Adding scroll bar when keyboard is shown.


**Fig. 3.** Keyboard covers part of UI.

*3.2.4. Design pattern: Only use the part of the screen that will not be covered by the keyboard*

In practice, what this solution does is to reduce the size of the part of the screen that may be exploited.

*3.2.4.1. Use when.* This solution may be OK for dialog boxes as illustrated in Fig. 4, but is seldom practical for normal windows.

*3.2.4.2. How.* Restrict the amount of information in the dialog.

*3.2.4.3. Why.* The solution is simple and inexpensive.

*3.2.5. Design pattern: Use one large UI control as a buffer*

By this we mean that when the keyboard is added, one of the controls is reduced vertically to be just as much smaller as the size of the keyboard, as illustrated with the list box control in Fig. 5.

*3.2.5.1. Use when.* The solution is relevant when the UI contains one or more controls that may be used as a buffer.

*3.2.5.2. How.* General controls that may be used for this are primarily list boxes and multi line text boxes.

*3.2.5.3. Why.* The solution is simple and inexpensive, yet it usually does not confuse the user.

*3.2.6. Design pattern: Keyboard as part of layout*

Instead of using a built-in software keyboard that the application have to adjust to, it is also possible to have an application specific keyboard that is designed to be part of the layout, as illustrated in Fig. 6.

*3.2.6.1. Use when.* The solution is most appropriate in mass market products where the extra costs for designing application specific keyboards will pay off, or when such solutions are supported by the OS (like on the iPhone platform).


**Fig. 4.** Dialog box which leaves room for keyboard.

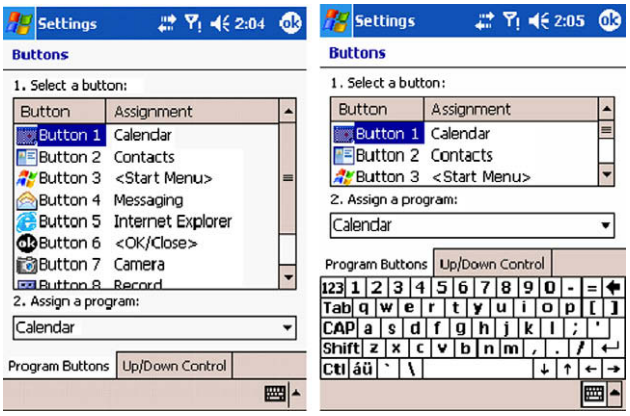


**Fig. 5.** List box control that shrinks when keyboard is added.

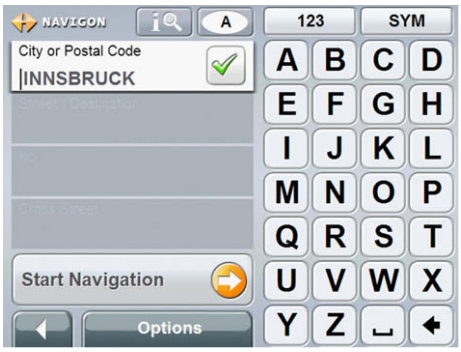


**Fig. 6.** Keyboard as part of layout of an application.

*3.2.6.2. How.* An application specific keyboard must be developed.

*3.2.6.3. Why.* The solution may provide both very efficient, and user – as well as finger–friendly UIs.

*3.3. Mechanisms for entering text*

The main mechanisms for entering text on PDAs are SW keyboards, small physical keyboard and stokes-based input. Common for all built-in physical keyboards is that they are so small that the user must enter text with one or two finger. Common for most SW keyboards and all strokes-based input is that it is difficult to operate them without using the stylus.

When designing a PDA application where the user must enter some text, there are two related problems to solve: how to avoid that the user must enter text and how to make it easier for the user to enter text. The main goal is often to avoid having to use the generic text entering mechanisms.

*3.3.1. General guidelines*

A different type of solution than the design patterns presented below is to collect data from some other source than user interaction, usually by *exploiting contextual data* [7,12]. This principle is based on an assumption that the context in which a mobile user operates changes more rapidly than it does for a stationary user and that this knowledge may be used to make the application more user friendly, i.e. obtaining data that the user would have had to enter if the application did not have this ability. An example of this is using an RF-ID sensor to identify a piece of equipment that is to be inspected so that the user is relieved from entering a long and cryptic equipment id. A related alternative to exploiting contextual data is to use *multimodal input* [16].

*3.3.2. Design pattern: Auto complete*

This is a mechanism that tries to guess what the user is about to write and suggests this by filling in the suggested text ahead of the writing of the user, as illustrated in Fig. 7.

*3.3.2.1. Use when.* The solution is relevant when there are some patterns in what the user writes that are repeated over time.

*3.3.2.2. How.* On the Windows Mobile platform an adaptive auto complete mechanism is included in all the generic input mechanisms. Specialized applications specific auto complete mechanisms in certain field are usually more efficient. This is common when writing an URL in most web browsers and when writing names in an email client. Common for such solutions is that they use the history of values used earlier to suggest the new ones.

*3.3.2.3. Why.* The solution reduces the amount of repetitive typing.

*3.3.3. Design pattern: Predefined values*

By this we mean having a list of all (or the most common) texts to enter in a field.

*3.3.3.1. Use when.* The solution is relevant when there is a small set of words or phrases that are used more often than others.

*3.3.3.2. How.* The list of values may be accessed from a menu or from a combo box, as illustrated in Fig. 8. The values in the list may also be dynamic based on user behaviour.

*3.3.3.3. Why.* The solution reduces the need for typing commonly used words and phrases.

*3.3.4. Design pattern: Alternative input mechanisms*

By this we mean using UI controls that are operated directly on the screen as an alternative to keyboard, as illustrated in Fig. 9.

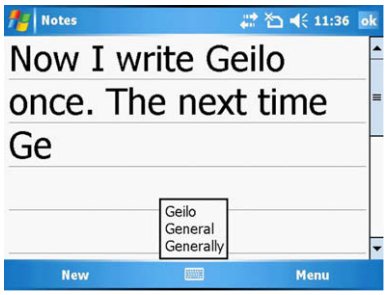


**Fig. 7.** Auto-complete in a notes application.

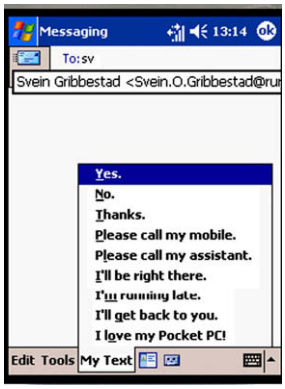


**Fig. 8.** Using predefined answer alternatives in a messaging application.

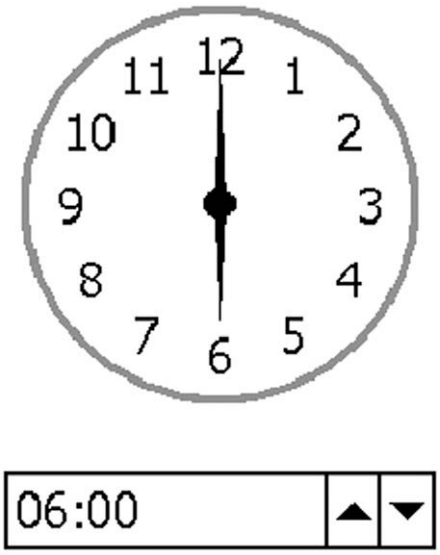


**Fig. 9.** Using clock and spinners for adjusting the time.

*3.3.4.1. Use when.* Most of the relevant mechanisms require that there is some sort of restrictions on the domain of the attributes that should be entered through the mechanism.

*3.3.4.2. How.* In addition to radio buttons, combo and check boxes, spinners, sliders, and menus are the most common controls for this.

*3.3.4.3. Why.* Direct manipulation is usually more efficient and easier to perform than typing.

*3.3.5. Design pattern: Specialized input mechanisms*

By this we mean using (a combination of) existing controls in a new way to implement a creative solution.

*3.3.5.1. Use when.* The solution is appropriate in most situations, specially when other patterns are not relevant.

*3.3.5.2. How.* An example of this approach is the mechanism used in an application for service technicians, where the user may write common fault description in a natural language like syntax by choosing from a set of drop down list with commonly used nouns, verbs and preposition expressions.

*3.3.5.3. Why.* Having a restricted number of values in each drop-down list still facilitates entering a very large number of possible sentences in a simple way.

*3.4. Interacting with applications without using stylus*

This problem is only applicable for devices that are designed to be used both with and without a stylus (not for devices that only can be controlled using HW keys).

For some users it is not practical to use the stylus. This could be because the user wears gloves, because it too cold to fumble with the stylus, because the user has only one hand available, because the user has lost the stylus, or because the user prefers not using it.

The most obvious problem that occurs when using the finger instead of a stylus to control a PDA is that the precision when pointing is coarser. Combined with the fact that a finger conceals larger parts of a UI than a stylus does, makes it even more difficult to hit small details using a finger than with a stylus. This problem gets

even worse if the user uses gloves of some kind. The most obvious solution, i.e. making the controls bigger easily increase screen space problems.

*3.4.1. General guidelines*

Controlling the PDA using a finger instead of a stylus is partly a question of choosing, partly a question of adapting and partly a question of making UI components (controls). These three levels of solving the problem are proportional regarding cost vs. potential benefits.

Just *choosing most appropriate UI components* as well as *simple adaptation of UI components* has no additional costs connected to the components, but may not facilitate an "optimal" solution. By simple adaptation of UI components we mean adjusting the components' properties to make them more fit for finger use. Exactly which components that work best for finger control vary between the different PDA platforms. For the Windows Mobile platform Table 2 gives some characteristics of most of the standard components with respect to "finger friendliness".

By *advanced adaptation of UI components* we mean adaptations that require programming. Typically, this is done by making subclasses of built-in UI controls. Both the possibilities for doing this, the effects it may have, and how difficult it is to do this type of adaptation varies between platforms, development tools and available libraries. When tailoring an existing UI control is an option, this is usually a less expensive effort than implementing a self-made UI control from scratch – depending on how much of the existing functionality that is to be kept.

In cases where advanced adaptation of existing components is not possible or feasible for the required custom UI controls, *developing custom UI controls* is an option. The benefit of doing this is that it will give full control of the appearance and behaviour of the control. The main disadvantage is the costs involved.

*3.4.2. Design pattern: Finger friendly menu choices*

There are design patterns for finger friendly interaction for a number of interaction mechanisms, like finger friendly lists, menus, buttons, keyboards, tab folders etc. Here we include finger friendly menu choices as an example of such design patterns.

*3.4.2.1. Use when.* The solution is appropriate when the user wants or is required to operate an application using finger interaction.

*3.4.2.2. How.* An alternative to standard menus or buttons that are always visible is to provide menu choices in a small popup panel at the bottom of the screen, as illustrated in Fig. 10, showing how this is done in an iPhone application. Similar solutions are applied on HTCs TouchFlo 3D user interface on Windows Mobile.

*3.4.2.3. Why.* Menu items, both used as part of pull-up menus and context menus are difficult to operate using fingers. The solution also uses less screen space than buttons that are always visible.

*3.5. Design that supports branding, is aesthetic, and utilize screen space optimally*

Most organizations developing applications want to apply their own brand to the products. For application developers the branding may either be connected to the developer organization or the user organization (which may be the same) or even a combination.

Branding of UIs may be difficult to combine with following standard look and feel. If the branding includes a graphical profile, the UI controls will look different from platform standard. This may cause usability issues, among others that the UI is difficult to learn. Certain visual aspects of UI controls, often added for aesthetic reasons may decrease the affordance of the control. In most cases, branding occupies screen space. This is the case both for added visual elements, and if the various controls are given a special look.

*3.5.1. General guidelines*

A simple branding mechanism is to include the company logo various places in the UI. A more extensive branding mechanism is to also use colour and other visual elements from the graphical profile of the company in UI controls and backgrounds. A more subtle branding mechanism is to have a company specific UI design, i.e. a standard way of designing UIs that is specific for the organization, and that is easy to recognize.

**Table 2**
Different UI component and how they may be adapted to finger use.

| Component | Appropriateness for finger navigation |
|---|---|
| Button | Standard Button size is a bit small, but given a bigger size, Buttons are OK for finger use |
| TextBox | For entering text, it is sufficient to click on a TextBox – the rest of the interaction is done through some kind of text entry mechanism. Clicking on a TextBox is feasible using the finger when it has standard size, increasing the size will make it easier. Increasing the height may only be done by increasing the font size (unless it is multiline) Changing the text in the TextBox – e.g. by selecting and changing three characters in the middle of the text – is not trivial using just the fingers. Increasing the font size too much may easily result in a number of other usability problems |
| CheckBox | CheckBoxes are not too difficult to operate with fingers, depending on the distance to other UI controls. To trigger a CheckBox, not only the tick box, but the whole control (including the text and any additional space around the text) may be clicked. Given large enough size and distance to other components, CheckBoxes are easy to control using fingers |
| RadioButton | RadioButtons have the same characteristics with regards to finger friendliness as CheckBoxes. As RadioButtons always appear in groups, the distance/size requirements are essential |
| ListBox/ ComboBox | A standard size ListBox/ComboBox is only partly suited for finger use, as both the elements in the list and possible scroll bars are fairly small. Increasing the font size will make the elements in the list larger, but also makes it more likely that there is a need for using scroll bars (that do not increase in size) |
| ListView | Using icons and LargeIcons as View, ListView may be appropriate to control with fingers. |
| TreeView | A TreeView is difficult to control using fingers, and it is not possible to make it more appropriate by adjusting its properties |
| TabControl | The tabs in a TabControl are not too difficult to operate with fingers, depending a bit on the size. The size of each tab is partly dependent on the length of the text on the tab, and partly on the font size. All tabs should fit on the screen to avoid having to use the scrolling features of the TabControl (which are not easy to operate using fingers). So there is a clear trade-off that needs to be balanced |
| ScrollBar | ScrollBars are notoriously difficult to use on a PDA, even with stylus. The ScrollBars size may be increased to enhance their suitability – but then of course leaving less space for other components. Using alternative scrolling mechanisms should be considered as an alternative |
| UpDown | In their default size, UpDowns are almost impossible to operate using fingers, and there is no apparent way of adjusting size or fontsize |
| TrackBar | TrackBar are not specifically easy to operate using fingers, and there are no obvious ways of adjusting their properties to make them more suited |
| MenuItem | MenuItems (i.e. members of the pull-up menu on the bottom of the screen) are not too difficult to operate using the finger although the choices are fairly small. There are no ways for the application to change the size of its MenuItems |
| ContextMenu | Items in a ContextMenu are the same UI controls as MenuItems in a main menu, and are used in the same way – but triggering a ContextMenu is more difficult than a main menu using the finger, as the user must hit the control to which the ContextMenu is connected |

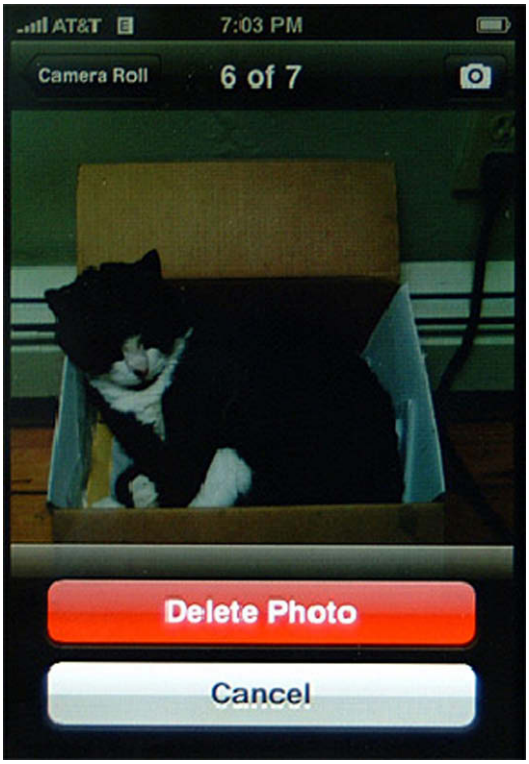


**Fig. 10.** Finger friendly menu in photo application on iPhone.

*3.5.2. Design pattern: Brand the standard*

By this we mean adding branding elements to the platform standard instead for building the elements that make up a brand from scratch, as illustrated in Fig. 11.

*3.5.2.1. Use when.* The solution is appropriate when both branding and compliance to standards are needed.

*3.5.2.2. How.* This should be done using subtle means, like changing background colours or adding a pattern or an abstract image as part of controls and/or backgrounds. Also, using a specific font may be a good branding mechanism. The main problem with this solution is that the branding may be difficult to recognize. On the other hand, implementing it does not need to be too costly.

*3.5.2.3. Why.* The solution combines having controls that are close to platform standards with branding.

*3.5.3. Design pattern: Branding the controls*

By this we mean generalizing the principle to also cover branding that is further from the standards, as illustrated in Fig. 12.

*3.5.3.1. Use when.* The solution is appropriate when branding is more important than compliance to standards.

*3.5.3.2. How.* Specialized controls need to be developed.

*3.5.3.3. Why.* This will usually take up less screen space than adding additional purely visual elements (like icons and advanced borders) as the main branding means. Doing more "radical" branding of UI controls may be quite expensive to implement. Using purely visual elements as branding means is less expensive to implement.

*3.6. User interaction during waiting for long-lasting operations to complete*

One of the things that make this problem special on a mobile device is that the necessary information for showing a reasonable progress indication may not be available on the device. A related issue is that it may be more difficult to predict the duration, e.g. when data is transported using a wireless connection with varying bandwidth. Also, obtaining the necessary information for showing progress – and partly also the process of showing it – may require more overhead on a mobile device than on a PC. As it usually is impossible, difficult or inconvenient to use other applications than the one performing the long-lasting operation on the device, long-lasting operations will probably make mobile users more impatient than a PC user. This increases the need for good feedback solutions.

The issues just presented may make it difficult to use the established mechanisms for showing progress for long-lasting operations. Sometimes it may be a trade-off between giving information and not delaying the long-lasting operation even more. Balancing this trade-off is probably the main problem when trying to find an optimal solution for user feedback during long-lasting operations.

*3.6.1. General guidelines*

When faced with a problem like the current where there are no optimal solutions, a possible approach is to try to eliminate the problem instead of solving it. In this case it means trying to *avoid*


**Fig. 11.** Adding branding elements within a standard UI.


**Fig. 12.** Compact RSS client with branded controls.

*or bypass the long-lasting operation*. There are of course lots of situations where this is not possible, because an operation requires transferring large amounts of data or very much computation. Still, trying to find smart implementations may speed up the application. When there is a need to transfer large amounts of data, a smart caching solution both pre-fetching data that the user may start working with before it is asked for, and continuing to transfer data while the user works with it – even with reduced performance – may be a much better solution for the user than waiting. If there is a need for a very demanding computation involving modest amounts of data, doing the computation on a server computer instead of on the device may enhance performance significantly. This is of course only possible if the application is on-line.

Given that it indeed is not avoidable that the user has to wait, information may be given on three levels. The normal way of doing this is providing a wait cursor, preferably supplied with a message saying "please wait...". A more advanced solution is to inform the user that something is happening, and indicate progress. This is normally done with a counter and/or a slider/gauge that shows the percentage of time spent. If it in addition is possible to estimate in actual time values the time left, this is a benefit for the user. The third level is presented as a pattern below.

*3.6.2. Design pattern: Inform the user about what is happening*

Informing the user about what is happing (in addition to indicating progress).

*3.6.2.1. Use when.* The solution is appropriate when this level of information is required or easily obtained.

*3.6.2.2. How.* This may be done as a scrolling text that the user can browse back in, just a small list showing the latest events, or as single text changing as events happen, as illustrated in Fig. 13. Independent of how the information is shown, it should be presented in a way that is comprehensible by the user – i.e. related to user concepts and user tasks.

*3.6.2.3. Why.* Providing information about level of progress as well as what is happening will make the user more patient.

## 4. Validation

The patterns collection was presented in half day tutorials at the HCI International conference in 2007 [9] and at the IASTED HCI conference in 2008 [10]. At the tutorials, the structure of the patterns collection was presented, and all problems were presented at a very brief level. Then, 11 or 12 of the 26 problems were presented in more detail. Partly, the problems considered most general was chosen, but the selection included problems from all the problem areas (and thus also all the main problem areas). During the presentation, the participants were asked to fill inn a questionnaire. In addition to background information, the participants scored the relevance of the main problem areas (based on the brief presentation of all patterns), the relevance and usefulness of each of the presented problems, and finally the relevance and usefulness of the patterns collection and such as well as their expectations for future use of the patterns collection. Relevance, usefulness and future use were scored on a scale from 1 to 6, where 1 = Not relevant at all/Not useful at all and 6 = Highly relevant/Very useful.


**Fig. 13.** Informing the user what is happening.

### 4.1. Respondents

Forty-eight of the participants at the tutorials handed in the questionnaire. There was a small majority of male, age varied from 25 to 50, most being around 30 years old. Geographically, the largest group had their origin in Asia, followed by Europeans. All continents are represented among the participants. Looking at educational level, most participants were on master level, closely followed by Ph.D. holders. There were relatively few undergraduates. A majority had a technical educational background. UI development experience varied from 0 to 20 years, the majority having 5 years or less experience. Experience in developing mobile solutions varied from 0 to 6 years, the majority having 1 year or less experience. Regarding device class, the largest respondent group having a preference was focusing on SmartPhones. Respondents having a preferred main platform were equally spread between Windows Mobile and other platforms.

### 4.2. Results

Scores on the main problem areas – based on a brief presentation of all the problems – shows an average score on relevance of 5.0 for Utilizing screen space, on 5.2 for Interaction mechanisms, and 4.7 for Design at large. Scores on the patterns collection as such – given after everything was presented – shows an average score on relevance of 4.8, on usefulness of 4.4 and on future use of the patterns collection of 4.3. All these scores verify that the patterns collection both addresses relevant problems and gives useful and practical advices on how to solve these problems. Regarding subjects scoring usefulness higher than future use, they may know many of the solutions in beforehand, and thus find the solutions useful, but do not need to consult the patterns collection to be able to utilize the solutions.

Looking at the scores for the individual problems that were presented in more detail, the scores vary a bit, but are still fairly high. Fig. 14 shows the average scores for relevance and usefulness for the 11 problems presented at both tutorials (names marked with asterisks indicate the problems presented above), sorted descending on sum of the scores for relevance and usefulness.

As for the patterns collection as such, the average scores for relevance are higher than the corresponding scores for usefulness for all problems but one. This is not surprising, as it is usually easier to agree with a problem description than a proposed solution.

Looking a bit more into the connection between relevance and usefulness, making a scatter plot of the values for each problem may be used as a way to categorize the different problems. Thus a problem with high score for both relevance and usefulness are excellent problems/design patterns. Problems with low scores for both relevance and usefulness should be considered removed from the collection. Problems with high relevance but low usefulness need to be improved. The last combination (low relevance and high

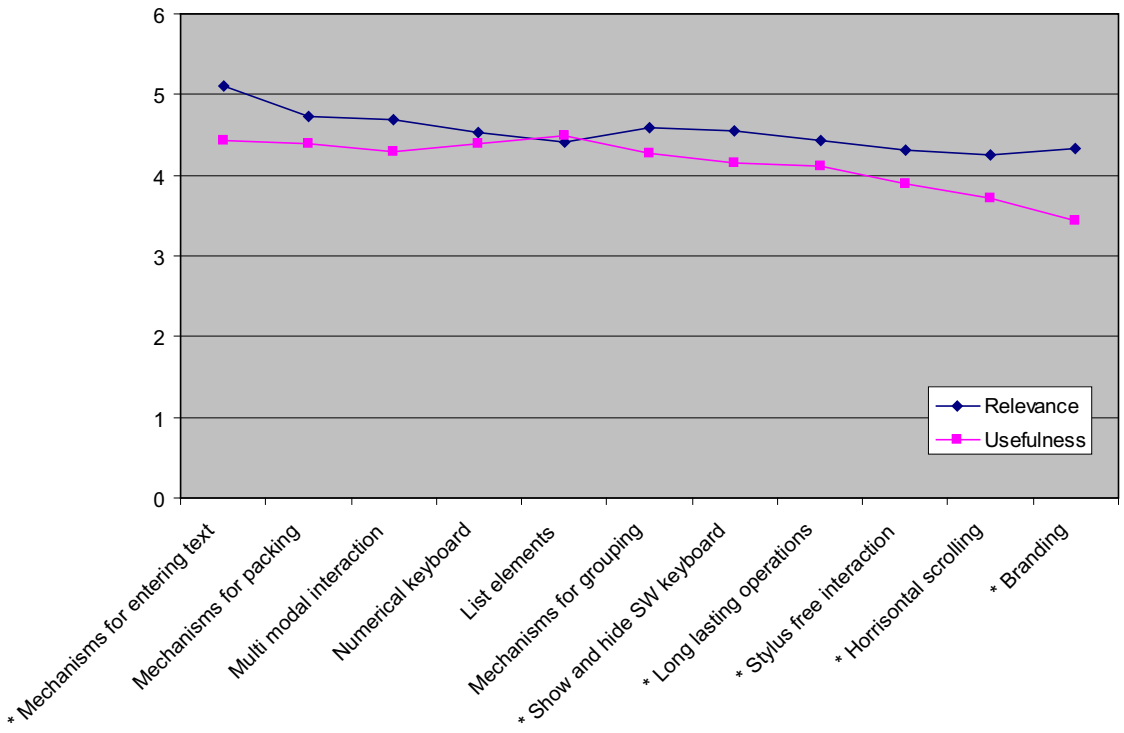


**Fig. 14.** Average scores for relevance and usefulness.

usefulness) is difficult to interpret, but as all but one problem score higher on relevance than usefulness, and this problem scores high on both, no problems fall into this combination. If we interpret high score to be on the top half of the scale, all problems but one fall into the excellent category. If we adjust high scores to mean 4 or above, some extra problems fall into the category "needs improvement" (like the problems on branding, horizontal scrolling and stylus free interaction presented above). Even with this adjustment, no problems are candidates of exclusion from the collection. In addition to the problems with scores below 4 for usefulness, also the problems where the difference between relevance score and usefulness scores are highest are candidates for further work on finding new and better solutions.

It may also be noted that correlation analyses show that the scores for relevance and usefulness correlates on the 0.01 level for 9 of the 11 problems. The problems not correlating on this are Mechanisms for entering text and Branding. Also, the scores for relevance for the collection as such, usefulness for the patterns collection as such and future use of the patterns collection all correlate pairwise on the 0.01 level. Furthermore, the scores seems consistent in the way that both relevance scores and usefulness scores correlate on the 0.01 or 0.05 level with the relevance and usefulness scores for the collection as such. The same is the case for average relevance and usefulness score correlated respectively with the relevance and usefulness scores for the collection as such.

Analyses of variance (ANOVA) show some patterns for five of the six problems presented in this paper. Usefulness scores for Mechanisms for entering text are significantly higher (0.05 level) for subjects coming from Asia. This may be explained by the extra challenges related to entering text in many Asian languages. Furthermore, subjects working with applications for personal productivity and professional application score usefulness significantly higher (0.05 level) for the same problem. This is not surprising as the patterns collection draws much of its background from this type of applications, and because it focus much on professional applications in general and forms-based UIs in particular. Usefulness scores for the problem Show and hide SW keyboard are significantly higher (0.05 level) for subjects with 0–1 years of mobile development experience than for subjects with longer experience. This problem may thus be considered fairly obvious for developers having worked with mobile UIs for some time. The same is probably the case for the scores for usefulness for the problem Horizontal scrolling, that are significantly higher (0.01 level) for subjects with 0–1 years of mobile development experience than for subjects with longer experience. The same effect may also explain that the usefulness score for the same problem is significantly higher (0.05 level) for devices classes other than SmartPhones. The relevance score for the problem Long lasting operations, scores significantly higher (linear on 0.05 level) for subjects with longer UI development experience. This shows that this problem pinpoint an issue that developers may need some experience to realize the importance of. A more surprising finding is that both relevance and usefulness scores for the problem Stylus free interaction receives significantly higher scores from female than from male subjects. One could speculate that a technical device like a stylus appeals more to male than female users, or that finger navigation is more suitable for users with smaller fingers, but both explanations are dubious. Relevance scores for the same problem shows that subjects with non-technical educational background have higher scores than subjects with technical background. This may be explained by the special layout and esthetical challenges posed by finger friendly mobile UIs.

For the whole collection, there are a number of general patterns. The most significant ones are for the independent variables gender, mobile development experience and main device class. The collection seems to appeal more to female than to male participants, the patterns are considered more useful to participants with little mobile development experience than participants with more such experience, and the collection appeals most to participants focusing on other device types than SmartPhone. The find-

ing regarding gender is difficult to explain. The finding regarding experience is interesting compared to an other (less significant) finding showing generally higher scores for subjects with longer UI development experience. This combination indicates that the patterns collection is best suited for experienced UI developers wanting to start developing mobile UIs. This is not a bad target group for such a patterns collection. The finding regarding device types is a bit surprising as the patterns collection use many examples from Windows Mobile and SmartPhones. The finding may be explained by a presumption that the patterns collection contains more "news" for the participants without experience with Windows Mobile and SmartPhones.

## 5. Using patterns format to document design knowledge

In this section we take a closer look at the appropriateness of using design patterns to document user interface design knowledge. The chosen patterns approach is in many ways well suited, as it captures the essential aspects of a problem and gives both general guidelines and more specific solutions through the patterns. Also, as design patterns may be on different abstraction levels they can be used to describe problems of different "sizes", as shown in the examples above. Furthermore, dividing a problem field into a limited number of well defined problems makes it possible to handle a set of manageable problems separately. Finally, having a patterns collection makes it possible to combine the just mentioned "divide and rule" principle with having an overall structure.

Despite these pros, there are also a number of problems with using patterns in this way. As the patterns format use its own structure, it is difficult to combine this with a rich problem structure without ending up with too deep hierarchies. In our case, we choose to present the problem structure separately, and with this as a basis present the individual problems and design patterns separately in a flat structure. This challenge is a general one for patterns collections handling more than just a very small number of problems, as using the collection will be very difficult without a structure grouping the problems in it.

The biggest challenge we have had using the patterns format is the connection between problems and solutions. Very often this is a many to many connection, i.e. in addition to having many solutions to one problem, a given solution may apply to more than one problem. An example of the latter occurs in the problem area "flexible user interfaces" in the main problem area "Utilizing screen space". The problems addressed in this problem area may together be solved with a set of techniques for handling adaptive and adaptable UIs both at design and especially at run time [12], but the solution for each pair of problems only partly overlap.

Presenting the same or very similar solutions to a number of problems, either causes a lot of cross-references between the individual patterns, or large amounts of repetition. Cross-referencing reduces the readability of the descriptions, while large amounts of repetition make the collection difficult to maintain. In prior versions of the patterns collection [11] we chose to use cross-references, as is being done in other patterns collections [4]. Recently, we have restructured the patterns collection so that each pattern represents either one solution or a unique combination of one problem and one solution. This has reduced the need for cross-referencing, but it has increase the number of patterns, thus making it more difficult to get an overview of the patterns collection. This is one of the reasons we have kept the labelled problem as part of the problem structure (and not replaced it with design pattern), and also kept some of the solution descriptions (i.e. the general guidelines) independent of the individual design patterns.

## 6. Related work

There are a number of patterns collections and even collections of patterns collections on the web,[1,2,3,4,5] see also [3] for an assessment of such collections. There are also a few collections of patterns for mobile user interfaces, like The Design Pattern Wiki[6] and Little Springs mobile UI design patterns.[7] The latter overlaps with two of our main problem areas. The patterns presented in [15], although focusing on mobile interaction, are much wider in its scope than our collection, with only two user interface patterns. [4,5] present a design patterns collection for ubiquitous computing that is fairly large, but has a broader scope with patterns that are on a higher abstraction level and/or are less comprehensive in the suggested set of solutions than our patterns. Also [2] presents a patterns collection for ubiquitous computing, though on a preliminary stage, presenting a set of what the authors call pre-patterns because they are not in common use.

## 7. Conclusions and future work

In this paper we have presented a structured collection of user interface design patterns for mobile applications. The structure is valuable both as an index to identifying patterns to use, and gives a fairly comprehensive overview of the problems that needs to be addressed when designing user interfaces for mobile applications. We have presented six individual problems with connected design patterns in some detail (but still abbreviated versions of the original problem and design patterns descriptions, as well as the number of design patterns included for each problem). In addition to representing important and relevant problems, they also act as examples of problems in all the three main problem areas, and thus show the breath of the patterns collection. The patterns are also on different levels of abstraction, showing how patterns may be used to present problems and solutions on different levels of detail.

The patterns collection has been validated using a questionnaire at two different tutorials. This validation shows that both the individual patterns assessed and the whole collection were perceived as relevant and useful by the participants, and that it is likely that they will use the collection in future work. It also identifies patterns that need more work.

Finally, we have discussed pros and cons of using a patterns collection for documenting design knowledge. The main pro being the ability to divide a large problem area into a structured set of manageable problems, the main con being that the same solution may apply to a number of problems, causing a lot of cross-references.

To manage this better, we have currently restructured the patterns collection. Furthermore, the collection is continuously being improved and enhanced, e.g. in the areas of multimodal interaction and exploiting context.

## Acknowledgments

The work on which this paper is based is supported by the UMBRA and FLAMINCO projects funded by the Norwegian Research Council and the industry partners in these projects, that have also given contributions to the contents of the patterns collection. I would also like to thank my colleagues Asbjørn Følstad and Jan Heim for contributions to designing the questionnaire used for

[1] http://www.developer.yahoo.com/ypatterns/.
[2] http://www.designinginterfaces.com/.
[3] http://www.visi.com/~snowfall/InteractionPatterns.html.
[4] http://www.welie.com/patterns/.
[5] http://www.deyalexander.com/resources/design-patterns.html.
[6] http://www.gibbert.net/DPWiki (in German).
[7] http://www.patterns.littlespringsdesign.com/~newlsdpatterns/index.php/Main_Page.

the validation and in analysing the results, and Mike Stiso for contributions to the structure and presentation of the patterns collection online.

## References

[1] Borchers J. A pattern approach to interaction design. John Wiley & Sons; 2001. ISBN:0471498289.
[2] Chung ES, Hong JI, et al. Development and evaluation of emerging design patterns for ubiquitous computing. DIS2004, ACM, Cambridge (Massachusetts, USA); 2004.
[3] Deng J et al. Managing UI pattern collections. In: Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on computer–human interaction; 2005.
[4] Van Duyne DK, Landay JA. Design patterns, course documentation; 2004.
[5] Landay JA, Borriello G. Design patterns for ubiquitous computing. In: IEEE computer; August 2003.
[6] Gamma E, Helm R, Johnson R, Vlissides J. Design patterns – elements of reusable object-oriented software. Addison-Wesley; 1995.
[7] Nilsson EG, Rahlff O-W. Mobile and stationary user interfaces – differences and similarities based on two examples. In: Proceedings of HCI international; 2003.
[8] Nilsson EG. Design guidelines for mobile applications. SINTEF Report STF90 A06003; 2005. ISBN:82-14-03820-0.
[9] Nilsson EG. Design patterns for user interfaces on mobile equipment. Tutorial documentation, HCI international; 2007.
[10] Nilsson EG. Design patterns for user interfaces on mobile equipment. Tutorial documentation, IASTED HCI; 2008.
[11] Nilsson EG. Design patterns for user interface for mobile applications. In: Proceedings of 7th international conference on computer-aided design of user interfaces (CADUI); 2008.
[12] Nilsson EG, Floch J, et al. Model-based user interface adaptation. Comput Graph 2006;30(5):692–701.
[13] Rahlff O-W. State of the art in using context in mobile information systems. SINTEF report A2299; 2007. ISBN:978-82-14-04069-2.
[14] Rolfsen RK. State of the art in form-based mobile user interface design and configuration. SINTEF report A2300; 2007. ISBN:978-82-14-04070-8.
[15] Roth J. Patterns of mobile interaction. Person Ubiquit Comput 2002;6(4).
[16] Vraalsen F, Holter T, et al. A multimodal context aware mobile maintenance terminal for noisy environments. IFIP mobile information systems (MOBIS '04).
[17] Vraalsen F. State of the art in design of mobile user interfaces. SINTEF report STF90 A05014; 2005. ISBN:82-14-03648-8.
[18] van Welie M, van der Veer GC. Pattern languages in interaction design: structure and organization. In: Proceedings of interact; 2003.