UNIVERSITY OF OSLO
Department of informatics

# Mobile, flexible user interfaces

Master thesis
60 credits

Tommy Jansson

1. may 2007

# I. ABSTRACT

This master thesis provides a **set of principles, or guidelines**, which aim to give practical and useful information in the process of developing user interfaces on mobile devices.

When talking about mobile devices the main focus is on PDAs and Smartphones, but it also incorporates portable PCs (including ultra portable PCs), tablet PCs and other mobile phones. The commonly used platforms, or operating systems, on these devices are Windows Mobile, Palm, Symbian, Linux and the new mobile adapted OS X. The guidelines aim to be platform independent, but most of the examples and illustrations are from the Windows Mobile environment.

The problem scope is found in the space between three main axes: user interface categories, user interface challenges and design approaches. The first two have been emphasized more than the latter.

Five **user interface categories** have been defined: *form* based, *icon* based, *document* based, *graphic* based and *repetition* based user interfaces. These categories are on an abstraction level above simple components and mechanisms but within the graphical user interface paradigm.

As **user interface challenges** go, a selection of six relatively common and often encountered challenges has been selected: *switching between portrait and landscape screen orientation*, *software keyboard showing/hidden*, *content larger than its display container*, *finger optimized user interface*, *switching between stylus and finger optimized user interface* and *user interface able to run on equipment with different screen size.* These challenges are found within the problems of utilizing screen space, flexible layout at run-time and flexible layout at design-time.

For designing and developing user interface four **design approaches** has been selected: *programming*, *drawing*, *modeling* and *marking up* a user interface. Each of these has differing pros and cons, which are dependent on the task at hand and where in the development process one might be.

Current mobile device applications have been examined. These were grouped under user interface categories and paired up against one or two selected challenges. The challenges where in most case handled in a relatively basic way. In some cases these solutions maintained usability, but far from always. Some challenges were rarely handled at all.

The guidelines found and presented conform to some main principles, which are: **Facilitating scrolling functionality** in one form or another (e.g. *Panning* and *Borders Facilitate Scrolling*), **adjust or adapt components** which the user interface consists of (e.g. *Struts and Straps* and *Simple Adaptation of Existing Components*), **group content differently** (e.g. *Group Similar Components in Tab Folders* and *Partial Tab Folders*) and **do nothing, or lock** the user interface (e.g. *Lock User Interface in One Orientation*).

## II. PREFACE

This thesis is done as the final assignment of a master's degree in informatics at The Department of Informatics at the University of Oslo, Norway.

The master thesis was presented by SINTEF ICT and is a part of the FLAMINCO (FLexible Applications exploiting Multi modal INteraction and COntext) project. This project focuses on development of user friendly solutions on mobile devices and will run from September 2006 until December 2008.

The FLAMINCO project is a user-driven innovation project which is supported by the VERDIKT program[1], and participants in the project are Captura, IT liberator, Locus, Teleplan and SINTEF ICT.

The main goal of the FLAMINCO project is to support and guide Norwegian service and applications developers, as well as tool vendors, in meeting the challenges they face when developing user interfaces on mobile devices. This is to be achieved through finding solutions and presenting guidelines to some of the major problems facing developers of future mobile solutions. The project will focus on user interface design within these fields, but it will also address challenges with regards to evaluation of mobile user interfaces. (SINTEF, 2007)

Many thanks go to those who contribute to this work in some form or another, both at the University of Oslo and SINTEF. Special thanks go to my main teaching supervisor Erik Gøsta Nilsson, senior scientist in Cooperative and Trusted Systems at SINTEF ICT, which made all this possible. First and foremost by contributing with invaluable knowledge and insight of this particular problem domain, but also for providing me with various resources, equipment and material which were necessary during the work with and completion of this master thesis.

University of Oslo, 16.05.2007

Tommy Jansson

[1] The VERDIKT program has been imitated by the Research Council of Norway as a process to strengthen and reorganize its ICT activities. ( The Research Council of Norway, 2005)

# III. Table of Contents

# 1 INTRODUCTION

The need and demand for solutions that offer mobility and interactivity is increasing in today's society. This can be seen amongst both the everyday people and within the industry and commerce. To meet this demand, mobile devices such as PDAs and Smartphones have come forth. These mobile devices have dramatically expanded the versatility of computers, by bringing them off the desktop and into new and unique environments.

Contrary to what many may believe, mobile devices are not merely a special case of a distributed system. Mobile computing poses a series of unique challenges for user interface design and development (Satyanarayanan, 1996). As mobile devices open a new horizon for computing it also brings with it several new, unique and conceptually different challenges, hereunder the challenge of building and maintaining flexible, mobile user interfaces. The need for user friendly, efficient and reliable mobile applications is pressing

The aim of this master thesis is to systematize and evaluate alternative principles, or guidelines, which can be used to design and realize flexible, mobile user interfaces – i.e. user interfaces that are able to adapt and conform to varying form and size of the available screen area. The alternative guidelines will be mapped with regard to how realistically they can be implemented and their key advantages and disadvantages. Then the alternatives are recommended according to under which context they are best suited.

Firstly a relatively brief summary of the mobile device context, which this thesis operates within, is presented. Hereunder the most common variations of mobile device types are listed. Then an overview of the most used and known device platforms is given.

The problem scope of this thesis is set by the space between three defined axes; user interface categories, user interface challenges and design approaches. The first two of these axes emphasized and discussed more in-depth, respectively under *User Interface Categories* (chapter 4 – page 23) and *User Interface Challenges* (chapter 5 – page 29).

Under user interface categories the following have been define:

- *Form Based*
- *Icon Based*
- *Document Based*
- *Graphics Based*
- *Repetition Based*

These categories are used as a means of grouping the varying types of user interfaces and examine complexity and general guidelines.

Further the following common challenges, which one can encounter in a mobile context, are used:

- *Switching Between Portrait and Landscape Screen Orientation*
- *Software Keyboard Showing/Hidden*
- *Content Larger Than its Display Container*
- *Finger Optimized User Interface*
- *Switching Between Stylus and Finger Optimized User Interface*
- *User Interface Able To Run on Equipment with Different Screen Size*

Each challenge is studied, and thereafter various guidelines are proposed.

The guidelines themselves are presented through the following structure with the user interface challenges as the top caption. Under these, appurtenant guidelines are listed. Each guideline has box at the top which informs about which user interface categories the current guideline most often applies for.

7.X – User Interface Challenge (no. 1 – 6)
  7.X.X – Guideline (no. 1 – n)
    *Relevant for user interface categories: X, Y and Z based*
    Various information (e.g. description, pros and cons and recommended design approach)

At the end all is summed up in a conclusion with main principles and key guidelines. Finally possible future work is also discussed and commented.

## 1.1 Central Words and Conceptions

### 1.1.1 Usability

Usability if often described as one of the two parts in usefulness, where the second part is utility (Grudin, 1992). Usefulness is the issue of whether a system can be used to achieve some desired goal, and utility is the question of whether the functionality of the system in principle facilitates what is needed.

Usability is not a single, one dimensional property of a user interface, but rather a collection of characteristics which together embodies the concept of usability. It is traditionally associated with these five usability attributes (Nielsen, 1994):

1. *Learnability* → This determines how easy the system is to learn – in other words, higher learnability means that the user more rapidly can start getting some work done.
2. *Efficiency* → This describes how efficient the system is to use for the user once learned. A high level of efficiency facilitates a high level of productivity.
3. *Memorability* →This tells how easy it is to remember the system and how to operate it. Good memorability allows the user be away from the system, come back after some period and still remember how to use it.
4. *Errors* → The system should have a low error-rate and prevent the user from using the system incorrectly. If an error does occur it should be concrete, easy to understand and recover from.
5. *Satisfaction* → Describes how subjectively satisfying the system is to use. In short terms: the user should like it.

### 1.1.2 Plasticity

Plasticity is a term with roots from the property of materials that are able to expand and contract without breaking and thus preserving continuous usage.

Applied to user interface design, plasticity is the capacity of a user interface to withstand variations of both the system's physical characteristics and the environment while preserving usability. In addition, a plastic user interface is specified once to serve multiple sources of physical variations, thus minimizing development and maintenance costs. Plasticity may be static and/or dynamic. It may be achieved automatically and/or manually. (Thevenin, et al., 1999; Calvary, et al., 2002)

# 2 MOBILE DEVICE CONTEXT

## 2.1 TYPES OF DEVICES

There is a distinction between users moving between devices and user moving with devices. Although both may be considered being mobile, only the latter is considered and discussed here. A user bringing with him a device may use any computer that is possible to carry (we also exclude use of stationary computers in cars etc.).

In this master thesis, we focus mainly on PDAs and Smartphones, but many of the challenges and how they can be handled may also be relevant when designing user interfaces for other mobile phones and tablet PCs, in some cases even for Portable PCs. (Nilsson, 2005)

A rough categorization of relevant computers or devices with computing abilities is presented below.

### 2.1.1 PORTABLE PC (INCLUDING ULTRA PORTABLE PCS)

By a portable PC (including ultra portable PCs) we mean a PC running a standard version of the operating system (usually Windows XP or the newer Windows Vista or a Linux distribution).

*Example 1 - Lenovo 3000 V100.*

These devices have a full size keyboard and support mouse as a pointing device. Some portable PCs have touch screens that may be operated by a stylus, but this is a supplement to the mouse, keeping the characteristics of a mouse operated environment (e.g. having a mouse pointer, supporting double click and supporting tool tips in a “normal” way).

### 2.1.2 TABLET PC

By a tablet PC we mean a PC running a version of the operating system tailored for tablet PCs (usually Windows XP Tablet Edition or the newer Windows Vista).

*Example 2 - LG C1 Express Dual.*

These devices may or may not have a keyboard (the keyboard may often be “hidden” when the user is walking around), but are tailored for usage without a keyboard. Modus operandi for these devices is operating through a “pure” screen based dialog where text is entered through some on-screen text entry mechanism(s), and controlled by a stylus. The Tablet Edition of Windows uses the “mouse” model of interaction with a visual pointer. Compared to a PDA, these devices share most of the characteristics regarding interaction mechanisms, although the tablet PCs focus more on handwriting. Thus, most of the material handling this in this document also applies to tablet PCs. As the screen size is larger on a tablet PC, the material on screen size are less relevant.

### 2.1.3 Ultra Mobile PC (UMPC)

By UMPC we mean a PC which is a specification for a small form factor tablet PC. It was developed as a joint development exercise by Microsoft, Intel and Samsung, among others. Current UMPCs feature the Windows XP Tablet PC Edition 2005, Windows Vista Home Premium Edition or Linux operating system. UMPCs will be able to run any software that has been written for the Windows XP platform, though the small form factor will mandate some changes to the interface.

*Example 3 - Samsung NP-Q1B.*

Due to the small size, most UMPCs do not feature a physical keyboard, but a virtual keyboard, known as DialKeys[2], is provided in the Touch Pack Interface. Also, since the device has standard USB 2.0 connectivity, external keyboards and mice can be attached. (Wikimedia Foundation, Inc., 2007)

### 2.1.4 Personal Digital Assistant (PDA)

By PDA we mean a small size computer whose main function is to support mobile tasks. The main common functions of these devices are Personal Information Management (PIM) applications (calendar, tasks, notes, email, etc.). It has become increasingly common that these devices also include camera and phone.

*Example 4 - HTC TyTn.*

Most devices have touch screen (operated by a stylus), and no or limited keyboard. If a keyboard is available, it is either a telephone type keyboard or a keyboard with very small keys. The screen resolution is usually in area of ¼ VGA, but recently, devices with VGA (640x480) resolution have become available. The most common pointing model is that there is no visual mouse pointer on the screen, and that there is no distinction between click and double click (i.e. click (called tap) means double click). Also, as there is no mouse pointer on the screen, there is no mouse move event, which is a severe hindrance for utilizing tool tips in a natural way. In addition to tap, the gesture "tap and hold" is supported, which in most cases plays the same role as right click on a mouse.

[2] A new text input method was implemented for the Ultra-Mobile PC. Consisting of two rings of keys around the lower corners of the screen, DialKeys is intended for use with the thumbs.

### 2.1.5 Smartphone

By Smartphone we mean a high end phone with PDA functionality. These devices share most characteristics with PDAs, and as the devices evolve, the distinction becomes increasingly superficial. Usually, Smartphones are considered to be less powerful, often smaller with less memory and weaker processor (and thus often longer battery life). As there are two versions of the Pocket PC/Windows Mobile operating system, one for Smartphones and one for PDAs (with the main difference being screen size requirements and requirements for using stylus), we have kept the distinction.

*Example 5 - HTC S310.*

Most Smartphones lack a touch screen and thereby does not support stylus based interaction. In addition to normal hard keys (hard coded key such as the number key pad), Smartphones normally have context-sensitive soft keys (Kiljander, 2004; Lindholm, et al., 2003) located along the display device, which various functions can be mapped to. This mapping is usually visualized by showing appurtenant function, on the display, near to the soft key. As a rule of thumb, a Smartphone is a phone with PDA functionality, while a PDA with phone is, well, a PDA with phoning capabilities. Although we have made a distinction here, the material in presented below applies both to PDAs and Smartphones.

### 2.1.6 Other Mobile Phone

By Other Mobile Phones we mean low end phones with no or limited PDA functionality. Again, the distinction is not very clear as almost all phones have some PIM support, but these devices have smaller screens, seldom touch screens, have telephone keyboard, and are less powerful. As they are seldom a target platform for professional applications, this type of devices is not focused in the material below. Despite this, many of the problems described also apply for these devices, in many cases even to a larger degree. The solutions, though, may not.

*Example 6 - Nokia 6300.*

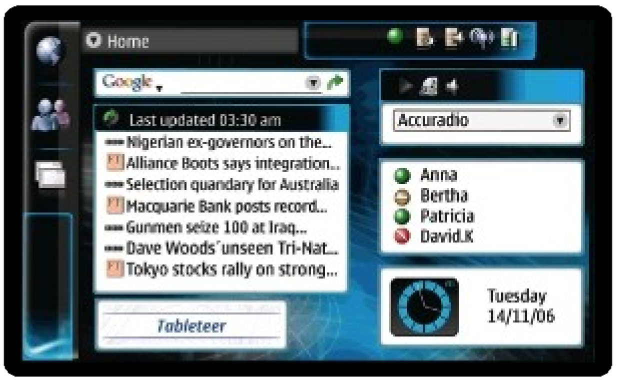
## 2.2 Device Platforms

There are different device platforms out there. The diversity is large, but here we have listed the five that are the most used and known.

The experience and guidelines/best practices formed in this thesis are base mainly on devices which run under Windows Mobile OS, more specific Windows Mobile 2003 SE. Even though this is the case, most of the content here should be of general interest and independent of preferred platform. Should there be some material which is operating system dependent then it will be noted. Nor do we find it appropriate to recommend a specific platform for developing applications and their user interfaces.

### 2.2.1 Linux

In the past several years, there have been numerous announcements of Linux-based handheld computers and PDAs for both general purpose and specialized mobile computing applications, that are either newly released or in various stages of development. There is a rapid emergence of Embedded Linux as a major "third alternative" (Ziff Davis Publishing Holdings Inc., 2001) to Palm OS and Microsoft's PocketPC (and Windows CE) for handheld personal computing devices.



Last updated 03:30 am
Nigerian ex-governors on the...
Alliance Boots says integration...
Selection quandary for Australia
Macquarie Bank posts record...
Gunmen seize 100 at Iraq...
Dave Woods' unseen Tri-Nat...
Tokyo stocks rally on strong...

Anna
Bertha
Patricia
David.K

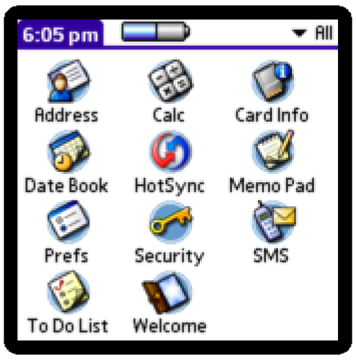
*Screen shot 2.1 - The Linux based Internet Tablet OS 2007 edition from a Nokia N800.*

LinuxDevices.com has created a guide to Linux-based PDAs, which provides a continually updated overview of available and developing Linux-based PDA support. This list can be found at: http://www.linuxdevices.com/articles/AT8728350077.html. (Ziff Davis Publishing Holdings Inc., 2006)

For additional information about Linux on mobile devices, visit: http://www.linuxdevices.com/

### 2.2.2 Palm

Palm OS is a compact operating system developed and licensed by PalmSource, Inc. for personal digital assistants, such as Handspring Treo and Visor, Palm Zire and Tungsten and Sony CLIÉ (Wikimedia Foundation, Inc., 2007), manufactured by various licensees. It is designed to be easy-to-use and similar to desktop operating systems such as Microsoft Windows. Palm OS is combined with a suite of basic applications including an address book, clock, note pad, sync, memo viewer and security software. Palm OS was originally released in 1996. (Wikimedia Foundation, Inc., 2007)

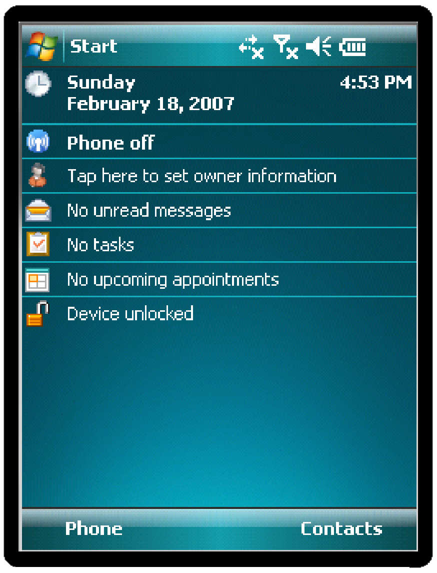


*Screen shot 2.2 - The applications tool running in Icon view on Palm OS 5.3.*

For additional information about Palm OS, visit:
http://www.palm.com/

## 2.2.3 Windows Mobile (PocketPC/Windows CE)

Windows Mobile is a compact operating system combined with a suite of basic applications for mobile devices based on the Microsoft Win32 API. Devices which run Windows Mobile include Pocket PCs, Smartphones and Portable Media Centers. It is designed to be somewhat similar to desktop versions of Windows.


Start
Sunday 4:53 PM
February 18, 2007
Phone off
Tap here to set owner information
No unread messages
No tasks
No upcoming appointments
Device unlocked
Phone Contacts

*Screen shot 2.3 - The "Today Screen" in Windows Mobile 6 Professional.*

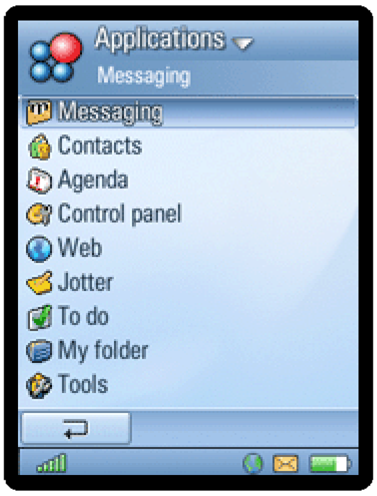
As version history goes, the newest addition to the Windows Mobile OS is Windows Mobile 6, which was released in February 2007. Predecessors to this are Windows Mobile 5.0 (released May, 2005), Windows Mobile 2003 and Windows Mobile 2003 SE (released June, 2003).

Pocket PC 2002 was powered by Windows CE 3.0. Targeted specifically at 240 × 320 (QVGA) Pocket PC (keyboardless) devices, PocketPC 2002 was, like the original PocketPC 2000 release, a stand-alone entity in the Microsoft Embedded device range. (Wikimedia Foundation, Inc., 2007)

For additional information about Windows Mobile, visit:
http://www.microsoft.com/windowsmobile/default.mspx

## 2.2.4 Symbian

Symbian OS is an operating system, designed for mobile devices, with associated libraries, user interface frameworks and reference implementations of common tools, produced by Symbian Ltd. It is a descendant of Psion's EPOC and runs exclusively on ARM processors.



*Screen shot 2.4 - UIQ3 Pen Style with default theme used on e.g. Motorola's RIZR Z8.*

Symbian is as of 4th of March 2007 owned by Nokia (47.9%), Ericsson (15.6%), Sony Ericsson (13.1%), Panasonic (10.5%), Siemens AG (8.4%) and Samsung (4.5%). Whilst BenQ has acquired the mobile phone subsidiary of Siemens AG the Siemens AG stake in Symbian does not automatically pass to BenQ - this will need the approval of the Symbian Supervisory Board.

Symbian OS, with its roots in Psion Software's EPOC, is structured like many desktop operating systems with pre-emptive multitasking, multithreading, and memory protection. (Wikimedia Foundation, Inc., 2007)

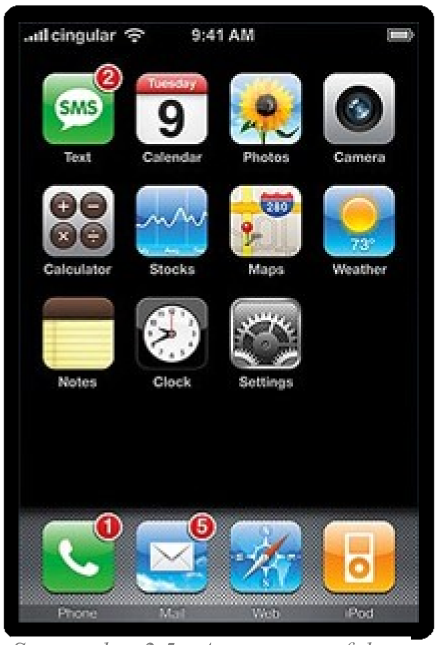
The Symbian OS is in many instances used as a basis for further adaption and specialization of an operating platform targeted a particular mobile devices. There are multiple platforms, based upon Symbian OS, which provide an SDK for application developers wishing to target a Symbian OS device – the main ones being S60 (used on devices like Nokia E90 Communicator, N95 and E61i, and Samsung SGH-i520 and SGH-i400) and UIQ (used on devices like Sony Ericsson M600, P990 and W900, and Motorola Motorizr Z8). (S60, 2007)

For additional information about Symbian, visit:
http://www.symbian.com/

### 2.2.5 Mac OS X

A new consumer PDA coming later this year (scheduled US release date of June 2007, later outside the US) is the Apple iPhone. Even though this is not release as of this date, we believe that it is necessary to mention this PDA and its new PDA operations system.

Apple has confirmed an optimized, full version of the Mac OS X operating system (without unnecessary components) will run on the iPhone, although differences between the operating system (OS X) running on Macs and the iPhone have not been officially explained. It is expected to take up "considerably less" than 500MB. It will be capable of supporting as-yet undetermined bundled and future 1st and 3rd-party applications, which are currently limited to a "controlled environment".



*Screen shot 2.5 - A prototype of the Mac OS X used in the iPhone.*

Apple intends to offer a smooth method for updating the iPhone's operating system, in a similar fashion to the way that Mac OS X and iPods are updated, and touts this as an advantage compared to other cell phones.

The iPhone has a (3.5 inch, 320×480 pixels) touch screen which is specifically created for use with a finger, or multiple fingers for multi-touch sensing. No stylus is needed, nor can one be used, as the touch screen requires bare skin to operate. For text input, the device implements a virtual keyboard on the touch screen. The iPhone interface enables the user to move the content itself up or down by a simple and natural touch-drag-lift motion, much as one would slide a playing card across a table. Additionally, the speed desired for scrolling is computed based on the speed and acceleration with which the drag motion is performed. It is possible to zoom in and out of objects such as web pages and photos by respectively placing two fingers (usually thumb and forefinger) on the screen and moving them farther apart or closer together as if stretching or squeezing the image.

The iPhone's version of OS X includes the software component "Core Animation" which is responsible for the smooth animations used in its user interface. (Wikimedia Foundation, Inc., 2007; Apple Inc., 2007)

# 3 Problem Scope

The problem scope of this master thesis is given by three different axes. Each axis denotes a category to be examined. Within this these axes a space emerges, and it is this space that defines the problem scope.

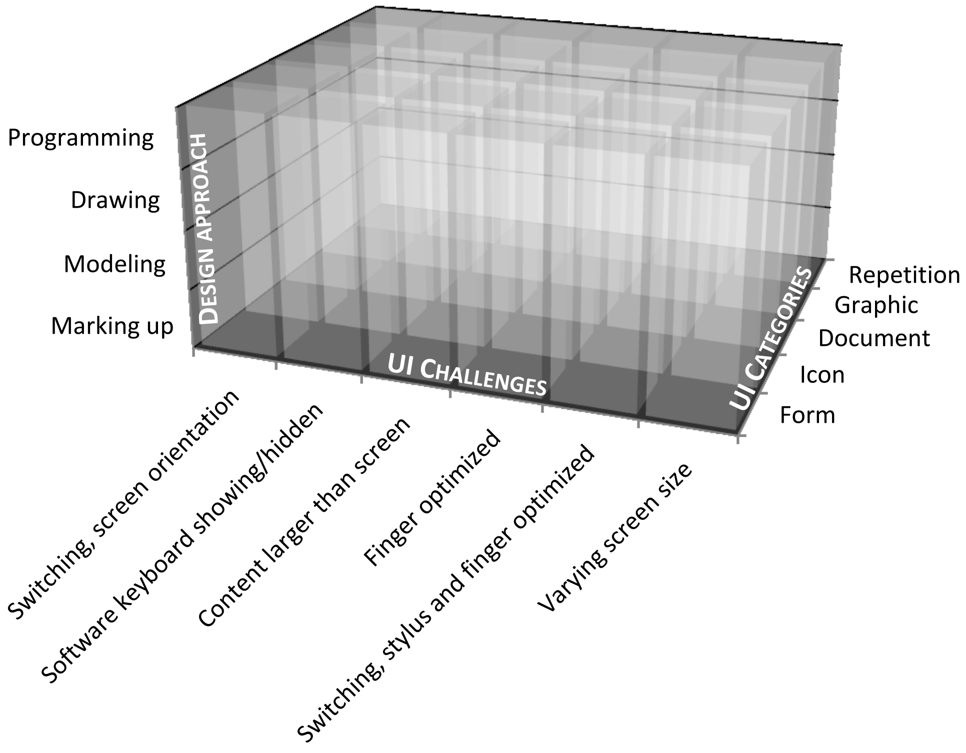


*Illustration 3.1 - Visualization of how the problem scope is defined within three axes.*

The *first axis* denotes a set of *User Interface Categories* (chapter 4 – page 23). These are categories representing a set or a composition of user interface properties or mechanisms which makes it distinct. The user interface categories selected here are abstracted above simple user interface controls and mechanisms but still operate within the concept of graphical user interfaces.

The following five user interface categories have been selected:

- *Form Based*
- *Icon Based*
- *Document Based*
- *Graphics Based*
- *Repetition Based*

*Axis number two* represents a set of *User Interface Challenges* (chapter 5 – page 29) one might need to take into account when designing and implementing a user interface on a mobile device. These challenges are of different sort and may occur in connection with for instance changing interaction styles and varying user interface space.

Six commonly faced challenged are selected for in-depth inspection:

- *Switching Between Portrait and Landscape Screen Orientation*
- *Software Keyboard Showing/Hidden*
- *Content Larger Than its Display Container*
- *Finger Optimized User Interface*
- *Switching Between Stylus and Finger Optimized User Interface*
- *User Interface Able To Run on Equipment with Different Screen Size*

*The third and last axis* denotes different existing and/or emerging design approaches used for designing and constructing user interfaces. These design approaches gives different advantages and disadvantages depending on what type of task at hand or where in the development process one might be.

There are primarily four approaches to designing user interfaces[3] (Nilsson, 2004):

- *Programming the user interface* → This involves using a programming library (or application programming interface – API) that facilitates implementation of a user interface using components, classes, methods, functions etc.
- *Drawing the user interface* → This involves using a screen painter to make a "painting" of the intended user interface, combined with a programming facility (textual or graphical) to specify the behavior of the UI.
- *Model the user interface* → This involves using a modeling language (graphically and/or textual) for specifying the UI on some level of abstraction, and applying a mapping process (automatic, semiautomatic, or manual) transforming the UI model to a running user interface (generated code or by interpreting the model by a run-time system).
- *Mark up the UI* →This involves using a mark-up language (HTML, WML, XML, etc.) to specify the contents and to a varying degree the appearance of a UI. This specification is interpreted by a generic client (browser).

Each axis, *user interface categories*, *user interface challenges* and *design approaches*, will be paired up against each other, though the main focus is on the plan that emerges between user interface categories and user interface challenges. If a design approach has an advantage over the others, for a specific category handling a challenge, this will be commented.

[3] Mark that a given development tool may use a combination of these four approaches

# 4 USER INTERFACE CATEGORIES

In the world of user interfaces there exist different categories or types. Each of these has a distinct set of properties which set them apart from each other and give them their unique form and/or appearance. Here we have chosen to divide the various user interfaces into five categories.

The five user interface categories are:

- *Form Based*
- *Icon Based*
- *Document Based*
- *Graphics Based*
- *Repetition Based*

It is rare that an application's user interface only concur with one of the user interface categories mentioned below. Most consist of a combination of two or more, which together make up the complete user interface. But then again, there is regularly one dominant user interface style or type in use which binds it all together.

When discussing complexity in a user interface category, complexity is here used in the context of how difficult it is to have good user interface plasticity – i.e. the capacity of a user interface to withstand variations of both the system's physical characteristics and the environment while preserving usability (Thevenin, et al., 1999; Calvary, et al., 2002).

The different user interface categories are presented in the following form:

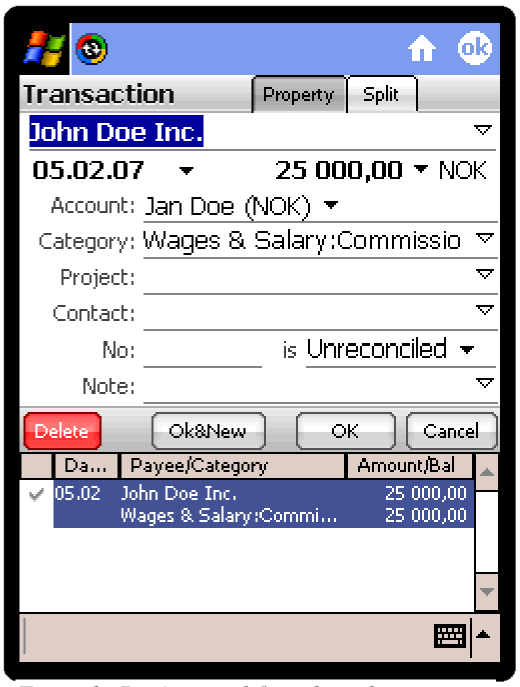
4.X – User Interface Category (no. 1 – 5)
*Complexity*
*Exemplifying screenshot*
Various information (e.g. general description, most common application types, etc)

## 4.1 Form Based

Complexity: *High to medium*

Form based UIs, often referred to as form fill-in, are one of the most common user interface types in use on present software solutions today. It may not always be the dominant part of a software solution’s user interface, but if one e.g. wants to input data or change some settings, the interaction is in most instances done via a form based user interface instance.

The form based user interface is constructed with paper based forms as the guiding interface metaphor, forms like those you might use when filling out your tax return or applying for a loan. Typical elements which are found in these paper based forms like text labels, text entry fields and check boxes, also have their counterpart in the form based user interfaces. In addition, as computer based user interface can be seen as more dynamic, there exists other more loosely inspired components.



John Doe Inc.

05.02.07 25 000,00 NOK

| | |
|---|---|
| Account: | Jan Doe (NOK) |
| Category: | Wages & Salary:Commissio |
| Project: | |
| Contact: | |
| No: | is Unreconciled |
| Note: | |

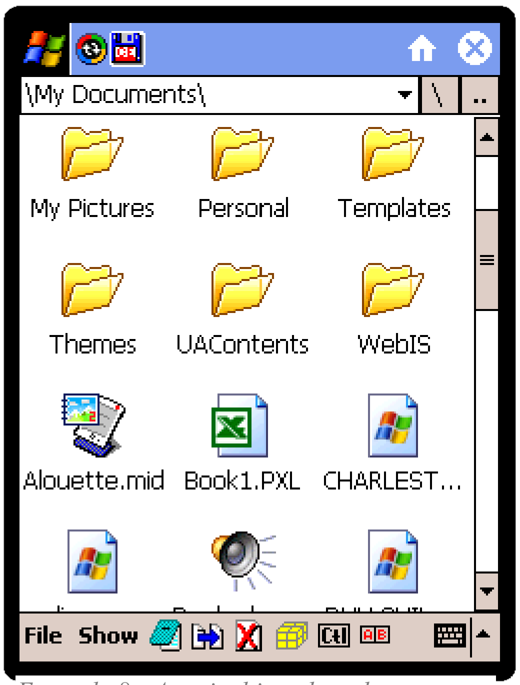
| | Da... | Payee/Category | Amount/Bal |
|---|---|---|---|
| ✓ | 05.02 | John Doe Inc.<br>Wages & Salary:Commi... | 25 000,00<br>25 000,00 |

*Example 7 - A typical form based user interface. This screenshot is from Cash Organizer 2007 Premium.*

This user interface type can be quite complex. The form based user interface can consist of many and multifaceted components which in themselves can be difficult to handle. When these in addition require interaction, not only from the user but from other components, maintain functionality and usability can be challenging. Hereby it is often vital that all components are preserved usable throughout the various context changes.

## 4.2 ICON BASED

Complexity: *Low*



*Example 8 - A typical icon based user interface. This screenshot is from Total Commander /CE.*

This is a fairly straight forward, but none the less, widely used user interface category. The icon based user interface has a main panel for showing icons which governs the major portion of the user interface. These icons may be automatically organized in different patterns, or in some cases customarily arranged only by the user of the system.

The icons themselves may represent instances like a file, folder, application or device on a computer operating system. In modern usage today, the icon can represent anything that the users want it to: any macro command or process, mood-signaling, or any other indicator.

Within this main panel the icons are in most instances presented or projected in three different ways. *Large icons*, with a typical icon size of 32x32 pixels, sorted from left to right and shifting down a step when full width is used. *Icon list*, smaller icons, often with size 16x16 pixels, listed downwards and when total height is used the list continues in a new column to the right. *Detail list*, much like the icon list with similar-sized icons, but this list not only shows an icon representing the file or directory, it also shows appurtenant information like e.g. size and date/time. The view is divided into columns and rows, where each row represents a file or directory and each column show distinct file or directory information.

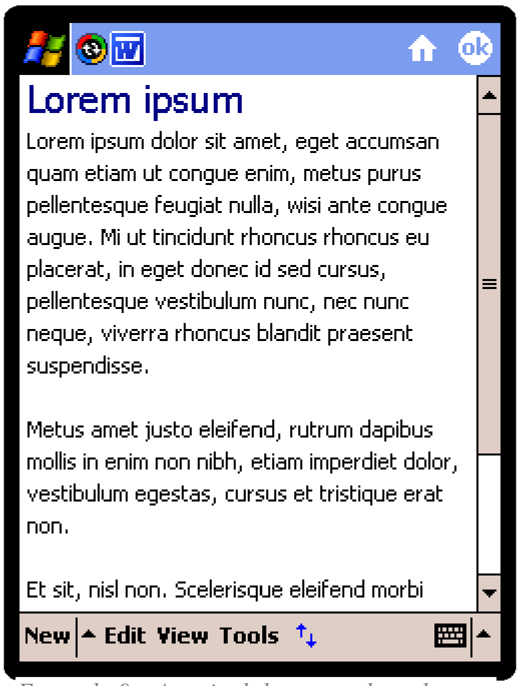
As for the icons, they can have varying sizes, but normally within a range from 48x48 pixels, through 32x32 and 24x24, down to 16x16, sometimes even the size might be higher or lower. 32x32 and 16x16 are the dimensions most frequently used. For instance, 32x32 is often used in connection with a *large icons* view 16x16 used in an *icon list* or *detailed list* projection. The icons are almost always accompanied by explanatory or descriptive text, most commonly place under or to the right of the icon it belongs to.

Icon based user interface are usually relatively easy to handle in terms of complexity. It is mainly a case of adjusting the main panel where the icons are shown, or adjusting how the icons themselves are represented.

## 4.3 Document Based

Complexity: *Medium to low*

This is a user interface category where the main portion of the screen is dominated by a panel presenting some form of document. By document we're talking about a large surface used for displaying, and maybe editing, mainly textual and numerical content. In many instances these document based user interfaces are modeled on typical physical documents one will encounter in working-day life.



Lorem ipsum

Lorem ipsum dolor sit amet, eget accumsan quam etiam ut congue enim, metus purus pellentesque feugiat nulla, wisi ante congue augue. Mi ut tincidunt rhoncus rhoncus eu placerat, in eget donec id sed cursus, pellentesque vestibulum nunc, nec nunc neque, viverra rhoncus blandit praesent suspendisse.

Metus amet justo eleifend, rutrum dapibus mollis in enim non nibh, etiam imperdiet dolor, vestibulum egestas, cursus et tristique erat non.

Et sit, nisl non. Scelerisque eleifend morbi

*Example 9 - A typical document based user interface. This screenshot is from Microsoft Pocket Word.*

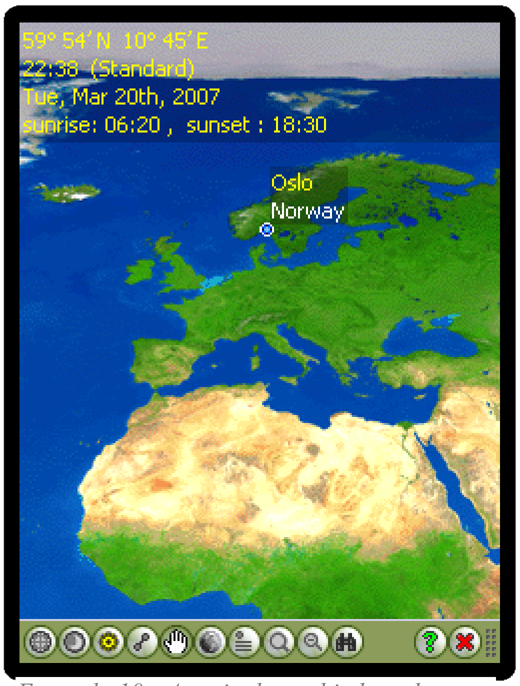
There is large variety between how the documents themselves are designed and presented. Well know application types which often belong in the document based user interface category are word editors, web browsers and spread sheet applications.

A document user based user interface generally stays within medium to low complexity in terms of being plastic. The main challenge is to conform the document itself to the changing context without losing the semantics and continuity.

## 4.4 Graphics Based

Complexity: *High to medium*

The graphics based user interface category is a quite wide category, and we are here talking about applications where the main portion of the user interface, often the whole screen, is governed by a panel dedicated to show a graphical content.



*Example 10 - A typical graphic based user interface. This screenshot is from Pocket Earth 3.4.*

One of the most fitting application types in this category is graphical information system (GIS) software, where most of the application user interface is used for a map with additional information placed upon. Other typical graphic based software types are games. Games often have an original and unusual user interface and frequently use the whole available screen.

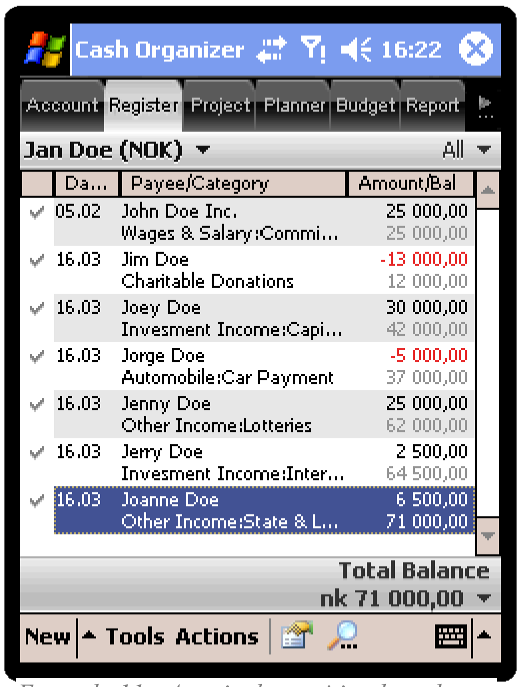
User interfaces in this category often stray away from the conventional instructing WIMP (windows, icons, menus and pointers) (Preece, et al., 2002) model used in other applications. More often the conceptual model behind is the *manipulating and navigating* or *conversing* (Preece, et al., 2002), which effects both how interaction is carried out and the visual appearance of the user interface.

In terms of how complex it is to keep usability in a changing context, this user interface category varies between high and medium. Some user interface instances can be very complex with an unorthodox layout and many custom components, while other instances can be much simpler.

## 4.5 Repetition Based

Complexity: *Medium to low*

The repetition based user interface category is a fairly diverse category when considering application domain. As the name says, it covers user interface where one or a few user interface elements or mechanisms are used repetitively as the main feature. Applications hereunder are typically lists, but also grids, with similar content being shown and arranged in some form or matter. The lists are generally shown with (columns as) attributes arranged besides each other horizontally and (rows as) items placed on top of each other. In some case this can be switched around with columns as elements and row as attributes. An attribute represents a data or information type and an item has instances of each of the data or information types.



| | Da... | Payee/Category | Amount/Bal |
|---|---|---|---|
| ✓ | 05.02 | John Doe Inc.<br>Wages & Salary:Commi... | 25 000,00<br>25 000,00 |
| ✓ | 16.03 | Jim Doe<br>Charitable Donations | -13 000,00<br>12 000,00 |
| ✓ | 16.03 | Joey Doe<br>Invesment Income:Capi... | 30 000,00<br>42 000,00 |
| ✓ | 16.03 | Jorge Doe<br>Automobile:Car Payment | -5 000,00<br>37 000,00 |
| ✓ | 16.03 | Jenny Doe<br>Other Income:Lotteries | 25 000,00<br>62 000,00 |
| ✓ | 16.03 | Jerry Doe<br>Invesment Income:Inter... | 2 500,00<br>64 500,00 |
| ✓ | 16.03 | Joanne Doe<br>Other Income:State & L... | 6 500,00<br>71 000,00 |

*Example 11 - A typical repetition based user interface. This screenshot is from Cash Organizer 2007 Premium.*

The items, which a repetition based consists of can represent various things, among others various textual or numerical data, functions, files and directories, etc.

Applications with a dominant repetition based user interface are sometimes fairly simple and/or one-purposed applications. Many applications have a section where the user interface is repetition based. A good example of this is the typical file browser tree (dialog) that is used in many applications, e.g. to open files, select save directory, etc. This file browser tree (dialog) has properties which place it under the repetition based category - i.a. items consisting of a icon and appurtenant text repeated and/or nested under each other.

The complexity of having good plasticity in repetition based user interface ranges from medium to low, and is mainly dependent on how intricate the individual repeated items are. Should the items be simple text based component it will probably be relatively simple. On the other hand, with items contain many and/or varying element the situation can be more intricate.

# 5 USER INTERFACE CHALLENGES

## 5.1 MAIN PROBLEM AREA

User interface challenges can be grouped under three main problem areas.

### 5.1.1 UTILIZING SCREEN SPACE

When talking about screen space this is defined by two different, but often codependent, factors. Firstly one has the *physical size of the screen*, often measured in inches from a corner to the diagonally opposite corner. Secondly we have the *resolution* used on the screen, normally given in number of pixel, width by height.

One of the most apparent problems with mobile devices is the available screen space, or more accurately put, the lack of screen space. This gives a series of challenges or problems which needs to be handled in order to optimize the user efficiency and experience, and ultimately make an application more usable. Utilizing screen space is not just about cramming as much as possible into a small area, but rather a concept of adapting the user interface to a specific task or challenge at hand. In one instance it can be about having lots of functions available inside a small space which is controlled via a stylus, but in another instance the goal can be to create a user interface suited for finger based interaction which demands very different means.

The screen resolution also varies greatly between existing mobile devices. The screen resolution might be as low as 128x128 pixels on some mobile phones and up to a typical 1024x768 (or widescreen 1280x800) on portable and tablet PCs. On a typical PDA the screen resolution usually is in area of ¼ VGA (320x240), but recently, devices with VGA (640x480) (e.g. Qtek 9000 and HTC X7500) resolution have become available.

### 5.1.2 FLEXIBLE LAYOUT AT RUN-TIME

This is user interface changes that should be handled at run-time, and we are talking about context changes around and within one specific mobile device.

The context, in which a user operates on a mobile device, changes much more rapidly than for a user operating stationary equipment. An important challenge when designing mobile user interfaces is to exploit knowledge about these changes to enhance the user experience. The context changes are multidimensional – and sometimes rapid – comprising location, light, sound, task, network connectivity, and possibly biometrics.(Herstad, 1998)

As a mobile device is used in these different and changing contexts, there is often a need to adapt the user interface at run-time, without the user needing to restart or change the application. The user might go from interaction via a stylus to fingertip, or the available area where the application user interface can unfold itself might be changing frequently (e.g. due to a software keyboard popping up or dropping down, or screen orientation change from portrait to landscape). Restarting or changing application can seriously interrupt the work-flow and already input data can be lost. This can be both inefficient and frustrating for the user.

### 5.1.3 FLEXIBLE LAYOUT AT DESIGN-TIME

In contrast to changing the user interface at run-time, one might have the need to change or adapt at design-time. Here the context change is not within or surrounding one specific device. The different context for a user interface is given by for instance the fact that an application will be installed and used on many different mobile devices. As the user interface may be run on different mobile devices, which often have differences in screen size, interaction possibilities, etc, it is often the easiest to handle this when in the design and development phase.

## 5.2 The Concrete Challenges

### 5.2.1 Switching Between Portrait and Landscape Screen Orientation

Main problem area: *Utilizing screen area*

Switching between a portrait and landscape screen orientating is something which nowadays is supported by many mobile devices. This support is either on application level, for example games might often use the screen in a landscape orientation even though the rest of the operations system is portrait oriented, or as a feature in the mobile device operations system, forcing (if possible) all subordinate applications to follow its configured orientations. Windows Mobile (now in version 6, and also with its predecessors down to PocketPC 2003 SE) supports it, and Palm OS does as well. Currently Symbian does not, although an application called *RotateMe* is under development for this platform (Symbian-Guru, 2007). In recent Symbian OS platforms rotation is supported at applications level. An example is the Nokia N95, which can be used in both portrait and landscape mode, where most of the standard applications can be rotated.

When changing between these two viewing modes the user interface display area changes and it might be necessary to handle this change in order to maintain usability. Whether or not the change is severe enough, is dependent on many factors from user interface type to available screen space and resolution.

This challenge can in most ways be compared to the challenge of having user interfaces run on equipment with different screen size. For instance, on a typical Windows Mobile device the screen has a resolution of 240x320 pixels (width by height) when used in portrait mode. When a user interface then is to be run on the same device in landscape mode, it is simply put a change from a resolution of 240 x 320 to 320x240. I could as well have been two different devices with respectively 240 x 320 and 320x240.

The main difference is that while a user interface quite possibly will be designed to support these two screen resolutions at design-time, the screen resolutions change here is within one device and the change needs to be handled at run-time. It is the latter that will be reviewed here.

### 5.2.2 Software Keyboard Showing/Hidden

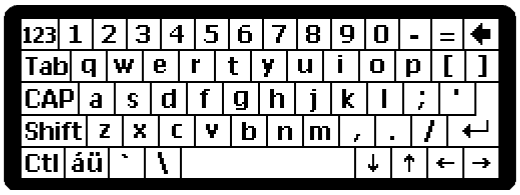
Main problem area: *Utilizing screen area*

As many mobile devices strive to be as compact and small as possible to maintain a high degree of mobility, they often come without at physical keyboard. This makes entering text at the best different, and usually much more difficult than using a keyboard on a stationary PC.

If a mobile device has a physical keyboard it is seldom a full size Qwerty-keyboard. In many instances the keyboard is reduced in either:

- Size – In example the keyboard size might be down-scaled to fit and/or slide under the mobile device front (like on HTC - TyTN (Qtek 9600) and i-mate JASJAR) or to stay within the width of the mobile device placed under the screen (as with HTC S620, Palm Tungsten C and PalmOne Treo 650)**.**
- Number of buttons - E.g. mobile devices where the keyboard is set-up with each button representing three characters (as on HTC - MTeoR Smartphone (Qtek 8600) or i-mate SPL).
- A combination of both.

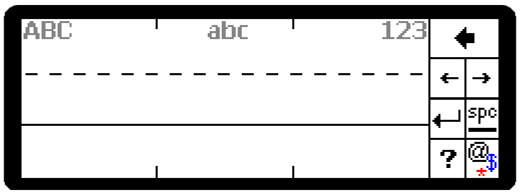
This lack of normal and maybe more user efficient physical keyboards has created a need for other input solutions, and here the software keyboard is one of them. In most cases this is shown, when in use, on-screen and anchored to the bottom part. When the software keyboard is shown it occupies a considerable portion of the screen which again gives the application less space to utilize.



123 1 2 3 4 5 6 7 8 9 0 - =
Tab q w e r t y u i o p [ ]
CAP a s d f g h j k l ; '
Shift z x c v b n m , . /
Ctl áü ` \ ↓ ↑ ← →

*Screen shot 5.1 - A software keyboard, here the default from Windows Mobile 2003 SE.*

It is worth mentioning that there also exist other software solutions for textual and numerical input. The most common ones are letter recognizers and transcribers, were the software recognizes or transcribes close to normal handwriting written on a designated are on touch screen. But usually these also occupy the same space as a normal software keyboard and will therefore not be commented as separate challenges.



*Screen shot 5.2 - A letter recognizer, here default from Windows Mobile 2003 SE.*

There also exist some software keyboards which utilize the whole screen as a virtual keyboard, leaving only a small field showing the entered/edited text. *Spb Full Screen Keyboard* (http://www.spbsoftwarehouse.com/products/fsk/?en) from *Spb Software House* is such a quite popular solution on the Windows Mobile platform. But this is a special case, and stands out as an independent application more than an accompanying utility used for textual and numerical input. It will as a result of the above mentioned not be commented here.

Another solution is where the whole touch screen is used for letter recognition or transcribing on top of the user interface without visually occupying any space. This has both its pros and cons, but will not be commented further.

### 5.2.3 Content Larger Than its Display Container

Main problem area: *Flexible layout at run-time*

As mentioned earlier, the screen space on a mobile device is limited as a result of mobile devices having to be just that, mobile. And it is fairly common that applications have a user interface which at some time or another use more space than what is available screen space. When the content is larger than its display container this will result in some part(s) of the user interface been partially or wholly hidden. This should in most cases be handled in order not to lose valuable information and/or application usability.

Many of the challenges listed here will in some form or another touch this challenge of content being larger than its display container. As a matter of fact, some challenges will in certain case, when it boils down to it, become a “simple” content larger than its display container challenge.

### 5.2.4 Finger Optimized User Interface

Main problem area: *Utilizing screen space*

Many mobile devices have the possibility to interact with the device tangibly via a touch screen, either by using a stylus (a dedicated pen-like pointing device) or something else, normally a finger. Although these two touching methods might seem the same or at least quite similar, there are some acute differences. These differences are largely due to the fact that the pointing accuracy of a stylus is much more accurate than that of a finger – we should specify that when talking about using a finger as pointing device, one can use both the fingertip and the finger nail. Here we assume, what in most instances is worst scenario, i.e. using the fingertip.

The most apparent problem when using your finger, instead of a stylus, is the lack of precision when touching the screen. Then taking into account that the finger also conceals a much larger portion of the screen when it is positioned over the screen, it is even more protruding. To even make matters even worse, the user might be wearing equipment like gloves and thereby further reduction accuracy. In order not to have a completely useless user interface adjustments will likely be necessary in order to handle these challenges and maintain usability.

### 5.2.5 Switching Between Stylus and Finger Optimized User Interface

Main problem area: *Flexible layout at run-time*

Switching between a stylus and finger optimized user interface also incorporates the challenges (with using a finger as a pointing device) as mentioned above. But as creating a finger optimized user interface is handled at design-time, the ability to switch between a stylus and finger optimized user interface is something which should be handled at run-time. It might not be acceptable or even possible for the user if it is necessary to restart the application when changing from using stylus to finger or vice versa.

As a stylus is both more accurate and less concealing, the user interface with its components and mechanisms can be of different form the event where a finger is the main pointing device. If there is a possibility that an application will be used with both a stylus and a finger as a pointing device, this is something which might need to be handled if one wants to preserve usability.

### 5.2.6 User Interface Able To Run on Equipment with Different Screen Size

Main problem area: *Flexible layout at design-time*

As stated under *Types of Devices* (chapter 2.1 – page 15) there exist a vast number of mobile devices, from portable and tablet PCs down to PDAs, Smartphones and mobile phones in general. When designing and developing software, one have to decide whether it is suppose to be used on a wide variety of devices. If this is the case, the probability that these devices have varying screen sizes is quite high.

For instance, within the PDA and Smartphone market alone the screen size, resolution wise, can go as low 176x220 pixels on a 2.2 inch screen (like on the HTC S310) up to VGA (640x480) on 4.0 inch screen (e.g. HP iPaq hx4700).

This great variation, both in screen size and resolution, gives several challenges when one is designing a user interface to be used across these different mobile devices. It can be challenges surrounding how to create a user interface which is usable on “small” Smartphone resolutions and still utilizes what a larger PDA resolution has to offer. Or if the different Smartphones and PDAs support tangible interaction, how to accommodate this when the user interface might have as little as 2.2 inches to unfold itself over and up to 4.0 inches.

# 6 CURRENT SOLUTIONS

There exist a number of solutions or applications which is some way or another encounter the user interface challenges listed in *User Interface Challenges* (chapter 5 – page 29). A set of applications have been selected in order to illustrate existing solutions and how they handle the challenges. We will not pair up all applications against all challenges, but rather show an exemplifying selection.

This section is structured in the following way:

6.X – User Interface Category (no. 1 – 5)
  6.X.X – Application (no. 1 – n)
    6.X.X.X – Challenge (no. 1 – n)

The main sorting category is the five user interface categories (form, icon, document, graphic and repetition based). Under each of these, two different and currently available applications have been selected. For each of these applications one or two challenges (from the total six challenges) have been explored to see how they are handled.

In the studies of how the user interface challenges were being handled by existing software, some challenges were easy to find and test while other were less easy to find, and some was not found at. This is natural since some challenges occur more frequently than other. Some of the challenges were also difficult to test due to the lack of mobile devices where the support of functions was limited. These were limitations which made it impossible to provoke one or more of the challenges. All this is commented further in *Summary* (chapter 6.6 – page 56).

## 6.1 Form Based Applications

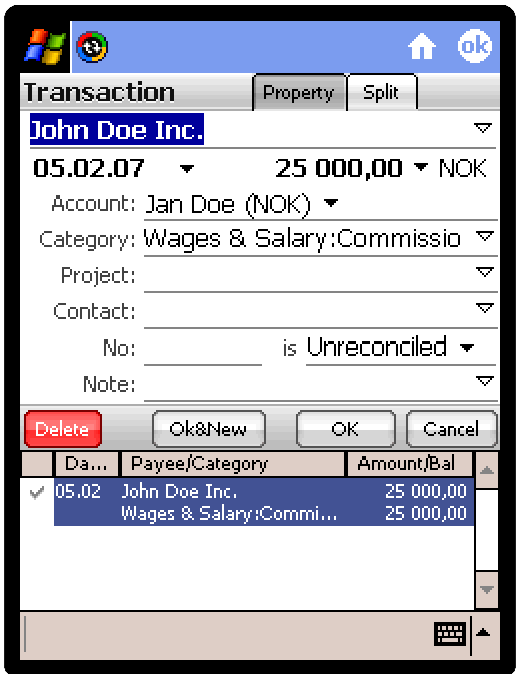
### 6.1.1 Cash Organizer 2007 Premium

Cash Organizer (http://www.inesoft.com/eng/index.php?in=premium.htm) can keep track of your accounts, watches the budget and remind of the planned payments. It is distributed by Inesoft. It has a variety of functions and user interface mechanisms, but a substantial part of it has a form based characteristics. The application user interface often takes on a form based structure when one inputs data.

#### 6.1.1.1 Handling: Switching Between Portrait and Landscape Screen Orientation

The Cash Organizer user interface is primarily set up for use in a portrait oriented way. In this application, registering a new transaction for instance has a typical form based user interface, with text labels, text entry fields, pull-down menus, etc.

In portrait orientation the registering a new transaction user interface has two main panels. The upper panel, occupying two thirds of the available screen, is used for entering transaction data. The lower panel, taking up the remaining one third, shows already registered transactions.

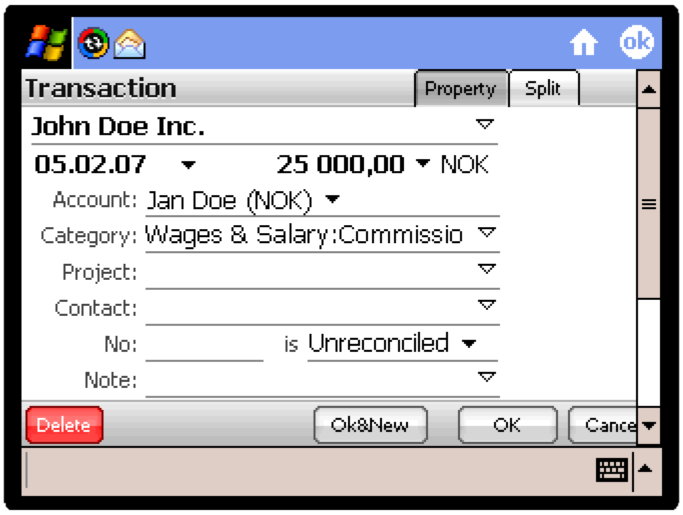


John Doe Inc.
05.02.07 25 000,00 NOK
Account: Jan Doe (NOK)
Category: Wages & Salary:Commissio
Project:
Contact:
No: is Unreconciled
Note:

| | Da... | Payee/Category | Amount/Bal |
|---|---|---|---|
| ✓ | 05.02 | John Doe Inc.<br>Wages & Salary:Commi... | 25 000,00<br>25 000,00 |

*Screen shot 6.1 - Registering a new transaction in portrait orientation.*

When the screen orientation has changed from portrait to landscape, the register new transaction user interface consists of one panel (in contrast to two when in portrait mode). This panel is the part used for registering transaction data.



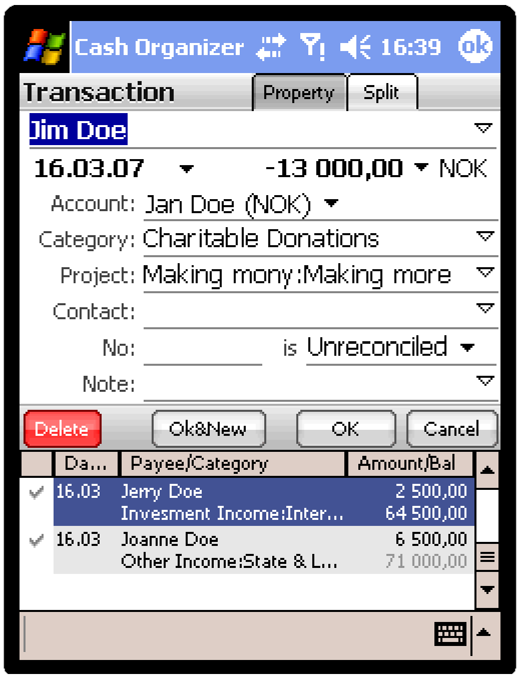
John Doe Inc.
05.02.07 25 000,00 NOK
Account: Jan Doe (NOK)
Category: Wages & Salary:Commissio
Project:
Contact:
No: is Unreconciled
Note:

Screen shot 6.2 - Registering a new transaction in landscape orientation.

The solution used in Cash Organizer's register new transaction user interface for handling switching between portrait and landscape screen orientation is by using one large component as a buffer (as described in *Use One Large Component as a Buffer* – chapter 7.3.2 – page 67). In this case a panel containing already registered transactions. This information was in this instance deemed insignificant enough to be removed in order to sustain usability after screen orientation.

#### 6.1.1.2 Handling: Software Keyboard Showing/Hidden

Again, when registering a new transaction, the user interface is built up of two main panels as described above. This is the case when no software keyboard is shown.

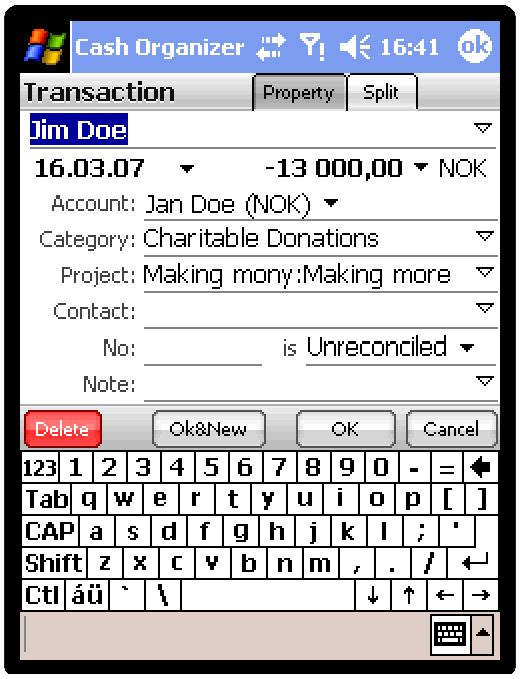


Jim Doe
16.03.07 -13 000,00 NOK
Account: Jan Doe (NOK)
Category: Charitable Donations
Project: Making mony:Making more
Contact:
No: is Unreconciled
Note:

| | Da... | Payee/Category | Amount/Bal |
|---|---|---|---|
| ✓ | 16.03 | Jerry Doe<br>Invesment Income:Inter... | 2 500,00<br>64 500,00 |
| ✓ | 16.03 | Joanne Doe<br>Other Income:State & L... | 6 500,00<br>71 000,00 |

Screen shot 6.3 - Registering a new transaction user interface without software keyboard showing.

When activating the software keyboard, this slides up from the bottom part of the screen and occupies about one third of the available user interface area. This software keyboard slides above the bottom main panel, hiding this.



Cash Organizer 16:41 ok

Transaction | Property | Split

Jim Doe

16.03.07 -13 000,00 NOK

Account: Jan Doe (NOK)

Category: Charitable Donations

Project: Making mony:Making more

Contact:

No: is Unreconciled

Note:

Delete | Ok&New | OK | Cancel

123 1 2 3 4 5 6 7 8 9 0 - =
Tab q w e r t y u i o p [ ]
CAP a s d f g h j k l ; '
Shift z x c v b n m , . /
Ctl áü ` \ ↓ ↑ ← →

Screen shot 6.4 - Registering new transaction user interface with software keyboard showing.

Here again the lower panel, which has information purpose, is used as a buffer (as described in *Use One Large Component as a Buffer* – chapter 7.3.2 – page 67). This is done to not compromise the panel for entering transaction data and still show a full software keyboard.

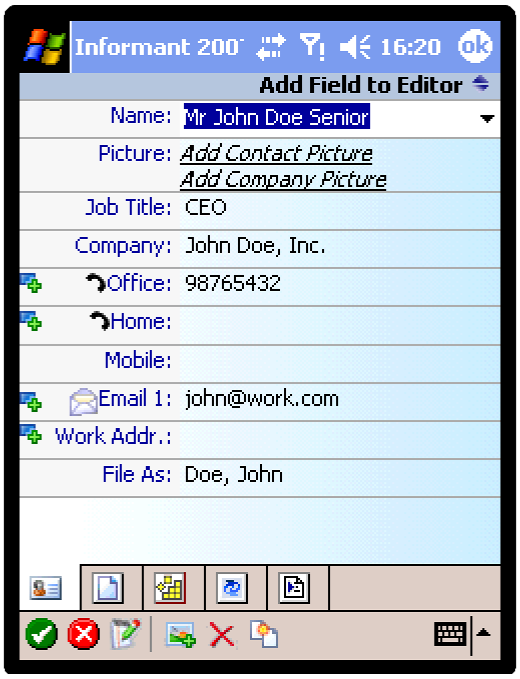
### 6.1.2 Pocket Informant 2007

Pocket Informant 2007 is a classical and all-embracing PIM (personal information manager) system. Among the functions are appointments, tasks, notes, contacts, and search views. Pocket Informant is developed and distributed by Web Information Solutions, Inc. (http://www.pocketinformant.com/). The user interfaces in this PIM can't be placed under just one user interface category (as what is the case with many applications), but one of the protruding categories in use is the form based one.

The edit contact user interface is a typical form based instance, although we could also argue that it has characteristics which could place it under the repetition based category. The contact information available for editing are listed in one main panel, with a title bar placed above and a tabs and toolbar under.

#### 6.1.2.1 Handling: Switching between portrait and landscape orientation

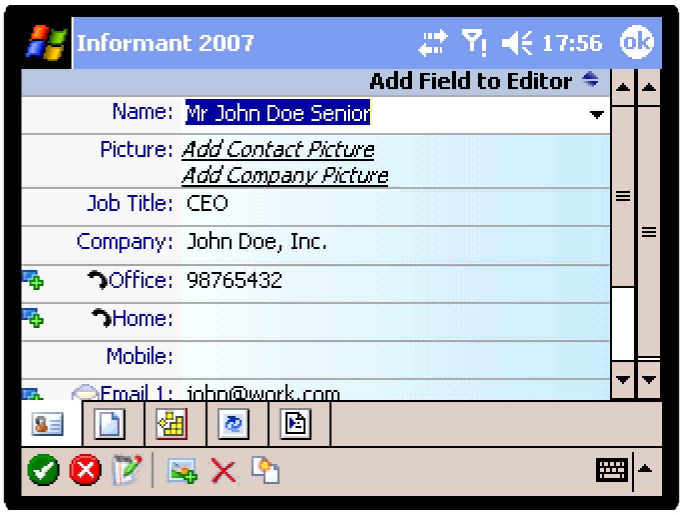
In portrait orientation, the panel show all information about selected contact, just as show below in *Screen shot 6.5*.



Name: Mr John Doe Senior
Picture: Add Contact Picture
Add Company Picture
Job Title: CEO
Company: John Doe, Inc.
Office: 98765432
Home:
Mobile:
Email 1: john@work.com
Work Addr.:
File As: Doe, John

*Screen shot 6.5 – Editing contact data in Pocket Informant 2007 with screen used in portrait orientation.*

When the screen is oriented in the landscape manner, the information is still listed as when in portrait mode, with the same sized fields and components. In this example, the information overflows available screen space (and concur with the challenge *Content Larger Than its Display Container* – chapter 5.2.3 – page 33), and to handle this a ordinary scroll bar is added to the right side in order to make the information outside the projection accessible.



Name: Mr John Doe Senior
Picture: Add Contact Picture
Add Company Picture
Job Title: CEO
Company: John Doe, Inc.
Office: 98765432
Home:
Mobile:
Email 1: john@work.com

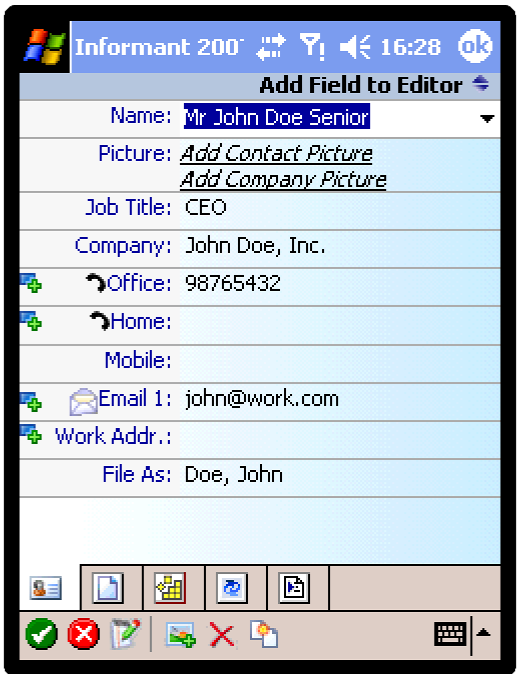
*Screen shot 6.6 - Editing contact data in Pocket Informant 2007 with screen used in landscape orientation.*

Adding a scroll bar when a projection is unable incorporate all content is a quite common, well-known and extensively used method (as described in *Add or Adjust Scroll Bars* – chapter 7.2.4 – page 65). It is also quite common solution to a projection constriction problem.

### 6.1.2.2 Handling: Finger Optimized User Interface

As purely finger optimized user interfaces go, this is not present in Pocket Informant 2007. The user interface is constructed with a stylus based interaction in mind, which supports a greater precision and less screen obstruction.

It this case when editing a contact, it is quite difficult to both see (because of the size of the finger and fingertip) and hit specific fields, tabs and buttons, especially when using your fingertip (and not the nail). The components are either not large enough or the spacing in-between is not sufficient.



| | |
|---|---|
| Name: | Mr John Doe Senior |
| Picture: | Add Contact Picture<br>Add Company Picture |
| Job Title: | CEO |
| Company: | John Doe, Inc. |
| Office: | 98765432 |
| Home: | |
| Mobile: | |
| Email 1: | john@work.com |
| Work Addr.: | |
| File As: | Doe, John |

*Screen shot 6.7 - Editing contact data in Pocket Informant 2007.*

## 6.2 ICON BASED APPLICATIONS

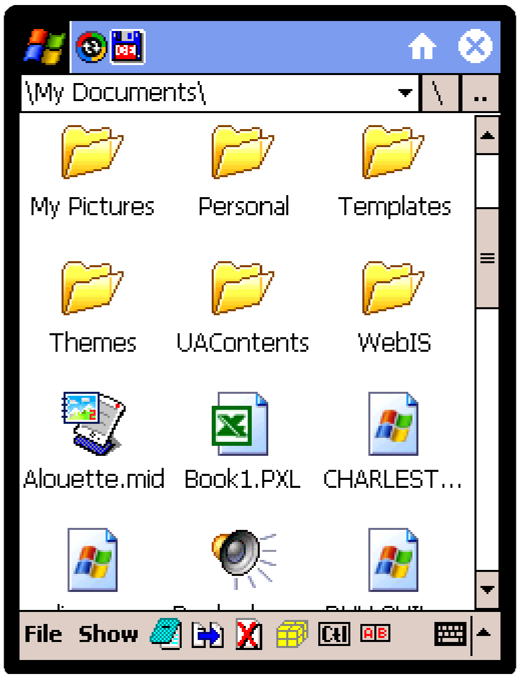
### 6.2.1 TOTAL COMMANDER/CE

Total Commander/CE (http://www.ghisler.com/ce.htm) is a functionally rich file browser and managing utility. The application is developed and distributed by Christian Ghisler, C. Ghisler & Co.

A key part of the application is where the files and directories are shown and can be handled and navigated through. The user interface here is dominated by a large main panel for the files and directories projection, paired together with an address bar placed above and a menu below.  These files and directories in the main panel can be viewed with three different projections; as large icons (typically 32x32 pixels), an icon list (with icons of size 16x16 pixels) or a detailed list (16x16 sized icons with appurtenant information). This user interface is a typical icon based one.

#### 6.2.1.1 HANDLING: CONTENT LARGER THAN ITS DISPLAY CONTAINER

It is often that the number of directories and/or files fills up and exceeds the available screen. The solution used in Total Commander/CE is fairly simple and common for this category of user interfaces; a vertical scroll bar is added, making content outside the current view reachable (as described in *Add or Adjust Scroll Bars* – chapter 7.2.4 – page 65).



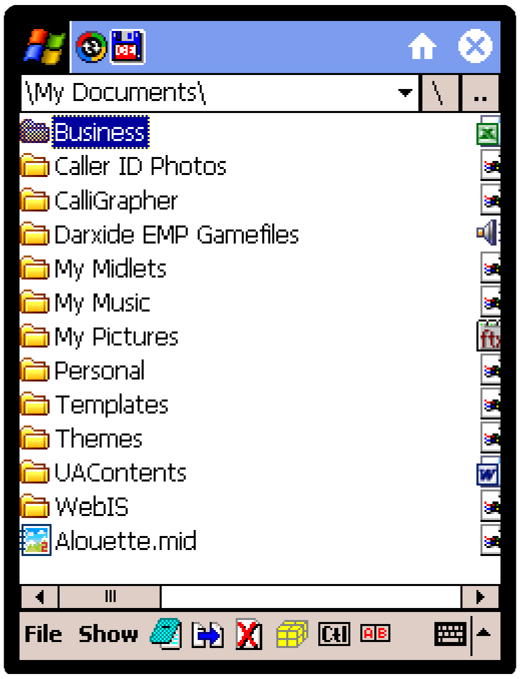
*Screen shot 6.8 - Example of how a view container overflow is handled in Total Commander/CE by adding vertical scroll bars.*

It is also possible to change the font of the text which accompanies the icons. To accommodate more icons within current screen space this font size can be decreased.

#### 6.2.1.2 Handling: Switching Between Stylus and Finger Optimized User Interface

There is little dedicated support in Total Commander/CE for switching between a stylus and finger optimized user interface, and the address bar and menu is static. But as it is possible to switch between different files and directories projections, some of these projections are more suited for stylus based interaction and others for finger based.

For instance the large icons projection, with 32x32 pixels sized icons, is quite fitting for finger based interaction due to the larger sized icons (as shown in *Screen shot 6.9*). If one then wants to use a stylus instead and do not need the larger sized icons it is possible to change to an icon listed projection (as exemplified by *Screen shot 6.10*).


\My Documents\
Business
Caller ID Photos
CalliGrapher
Darxide EMP Gamefiles
My Midlets
My Music
My Pictures
Personal
Templates
Themes
UAContents
WebIS
Alouette.mid

*Screen shot 6.9 - Example of how the files and directories projection can be more suited for stylus use in Total Commander/CE.*

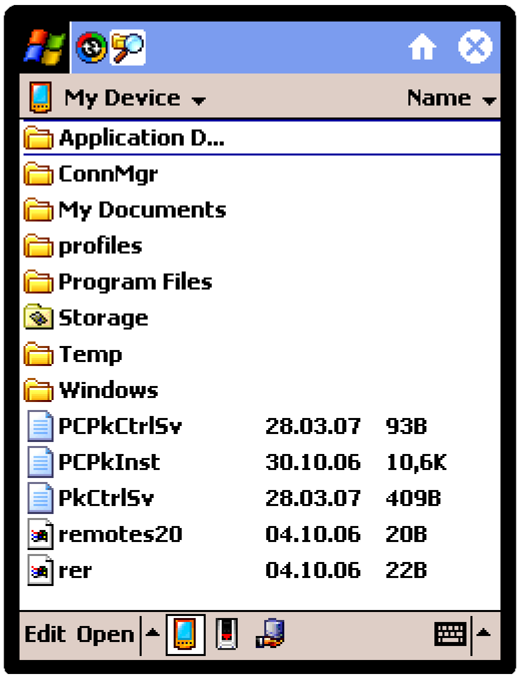
### 6.2.2 Pocket File Explorer

Bundled together with Windows Mobile 2003 SE, Pocket File Explorer is here the default file and directory browser. The functionality consists of basic and essential functions.

In the main panel showing directories and files, directories are show as 16x16 pixel icons with directory name following and files are show as 16x16 icons accompanied by filename, date and size. One line is used for each separate directory or file. This is a variation of a detailed list.

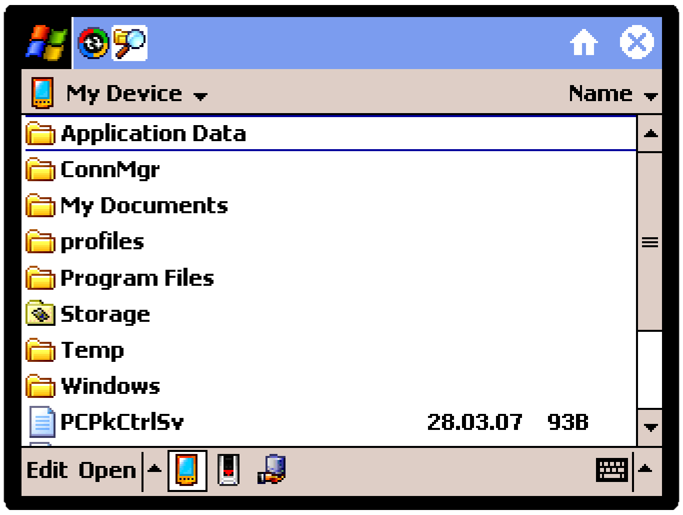
#### 6.2.2.1 Handling: Switching Between Portrait and Landscape Screen Orientation

When switching between portrait and landscape screen orientation there is little done in Pocket File Explorer to optimize the view to the new context. Should the content surpass available displaying area after a change from one orientation to another, vertical scroll bar will be added in order to keep all content accessible.



| | | |
|---|---|---|
| Application D... | | |
| ConnMgr | | |
| My Documents | | |
| profiles | | |
| Program Files | | |
| Storage | | |
| Temp | | |
| Windows | | |
| PCPkCtrlSv | 28.03.07 | 93B |
| PCPkInst | 30.10.06 | 10,6K |
| PkCtrlSv | 28.03.07 | 409B |
| remotes20 | 04.10.06 | 20B |
| rer | 04.10.06 | 22B |

*Screen shot 6.10 – A typical view in Pocket File Explorer in a portrait orientation.*

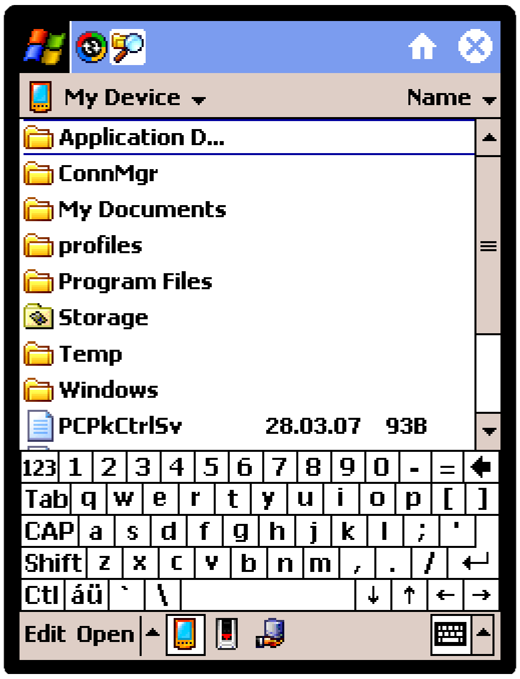


| | | |
|---|---|---|
| Application Data | | |
| ConnMgr | | |
| My Documents | | |
| profiles | | |
| Program Files | | |
| Storage | | |
| Temp | | |
| Windows | | |
| PCPkCtrlSv | 28.03.07 | 93B |

*Screen shot 6.11 - A files and directories view in landscape orientation.*

### 6.2.2.2 Handling: Software Keyboard Showing/Hidden

Pocket File Explorer handles the showing and hiding of a software keyboard in much the same way as when switching between portrait and landscape screen orientation and when the content exceeds available displaying area. If the software keyboard is showing the panel used for viewing files and directories in effect decreases in size, losing space to the software keyboard, and should the content exceed its offered area scroll bars are added or adjusted rendering vertical scrolling possible.



*Screen shot 6.12 - Pocket File Explorer with the software keyboard showing.*

## 6.3 Document based applications

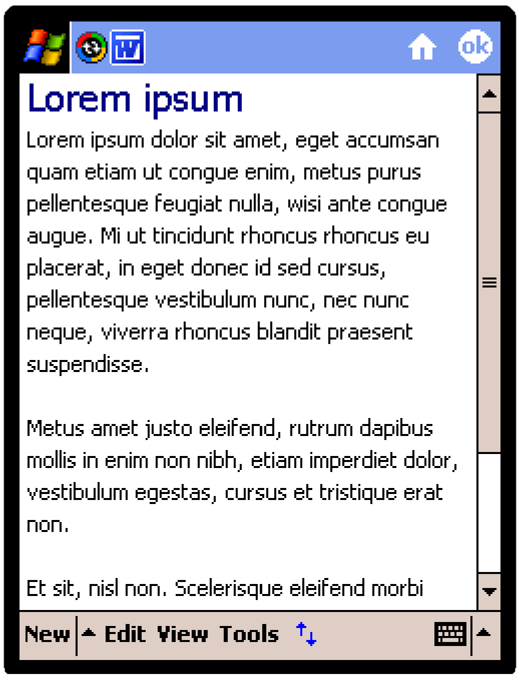
### 6.3.1 Pocket Word

Pocket Work is one of the features in the Microsoft Office bundle for pocket PCs. It is a stripped down and adapted version of its bigger brother, Microsoft Word for the Windows desktop environment. At the core is the ability to read and edit textual documents. The user interface mainly shows current document which may be accompanied by various tool bars and such.

#### 6.3.1.1 Handling: Content Larger Than its Display Container

A common challenge when using software with a document based user interface is when the content is larger than its display container, something which often is the case. For instance, when editing a textual document in Pocket Word the content will often not be small enough to fit within the panel area used for drawing the document.

When the shown document is larger than is display container this is, in Pocket Word, handle by quite simply adding a vertical scroll bar docked to the left side.



Lorem ipsum

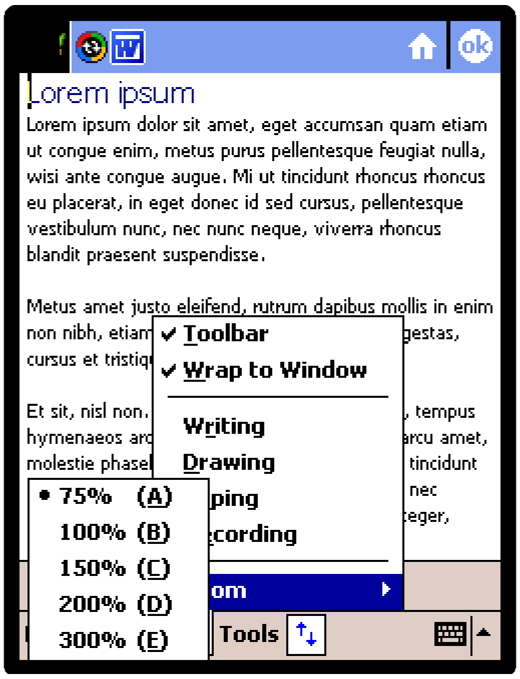
Lorem ipsum dolor sit amet, eget accumsan quam etiam ut congue enim, metus purus pellentesque feugiat nulla, wisi ante congue augue. Mi ut tincidunt rhoncus rhoncus eu placerat, in eget donec id sed cursus, pellentesque vestibulum nunc, nec nunc neque, viverra rhoncus blandit praesent suspendisse.

Metus amet justo eleifend, rutrum dapibus mollis in enim non nibh, etiam imperdiet dolor, vestibulum egestas, cursus et tristique erat non.

Et sit, nisl non. Scelerisque eleifend morbi

*Screen shot 6.13 - A document that is larger than its display container in Pocket Word.*

It is also possible to adjust the content by changing the zooming level, but this is something which has to be done manually. By using zoom it is possible to adapt or optimize the content some degree. But if zoom level is too high it becomes difficult to read the text and should the zoom level be too low it can be difficult to see the words context and thereby read it.



Lorem ipsum

Lorem ipsum dolor sit amet, eget accumsan quam etiam ut congue enim, metus purus pellentesque feugiat nulla, wisi ante congue augue. Mi ut tincidunt rhoncus rhoncus eu placerat, in eget donec id sed cursus, pellentesque vestibulum nunc, nec nunc neque, viverra rhoncus blandit praesent suspendisse.

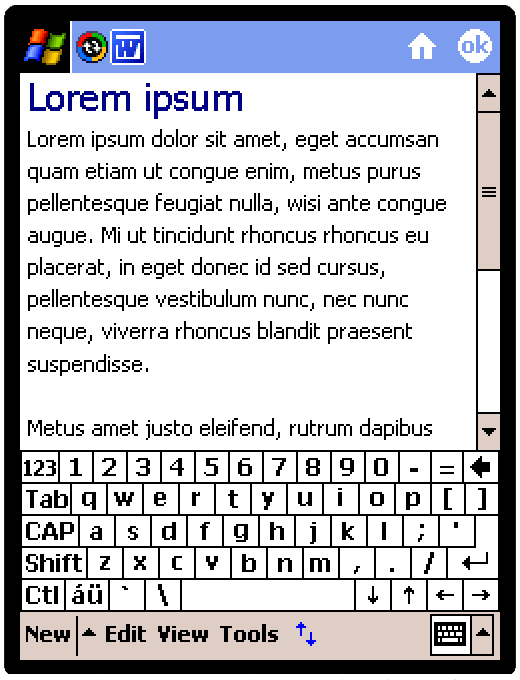
Metus amet justo eleifend, rutrum dapibus mollis in enim non nibh, etiam gestas, cursus et tristiq

Et sit, nisl non. , tempus hymenaeos ar rcu amet, molestie phasel tincidunt nec eger,

Screen shot 6.14 - Pocket Word at the highest zooming level. Zoom level can be selected in the menu.

### 6.3.1.2 Handling: Software Keyboard Showing/Hidden

When using Pocket Word one will quite often use a software keyboard to write or edit a document, especially if there is no type of hardware keyboard present. This software keyboard does, as specified earlier, occupy about one third of the bottom part of the screen. When this is shown, the panel for document projection is reduce and if this panel then is smaller than the document a vertical scroll bar appears docket to the left, render it possible to show the various parts of the document outside of the new projection.



Lorem ipsum

Lorem ipsum dolor sit amet, eget accumsan quam etiam ut congue enim, metus purus pellentesque feugiat nulla, wisi ante congue augue. Mi ut tincidunt rhoncus rhoncus eu placerat, in eget donec id sed cursus, pellentesque vestibulum nunc, nec nunc neque, viverra rhoncus blandit praesent suspendisse.

Metus amet justo eleifend, rutrum dapibus

123 1 2 3 4 5 6 7 8 9 0 - =
Tab q w e r t y u i o p [ ]
CAP a s d f g h j k l ; '
Shift z x c v b n m , . /
Ctl áü ` \ ↓ ↑ ← →
New Edit View Tools

Screen shot 6.15 - Pocket Word with a software keyboard showing.

### 6.3.2 Pocket Internet Explorer

As with Pocket Word, this is also a modified version of the richer Internet Explorer found on you average Windows desktop. It is purposed for show web related documents, typically HTML based web pages. At the center of attention is a large panel for view web document.

There are three different ways of laying out web documents in Pocket Internet Explorer:

- Default – Narrows content width to reduce horizontal scrolls.
- One Column – Forces all content to fit in a single column with no horizontal scrolling.
- Desktop – Makes no change to the content. Rendering for Internet Explorer Mobile is as close as possible to the rendering for Windows Internet Explorer on a Windows-based desktop platform.

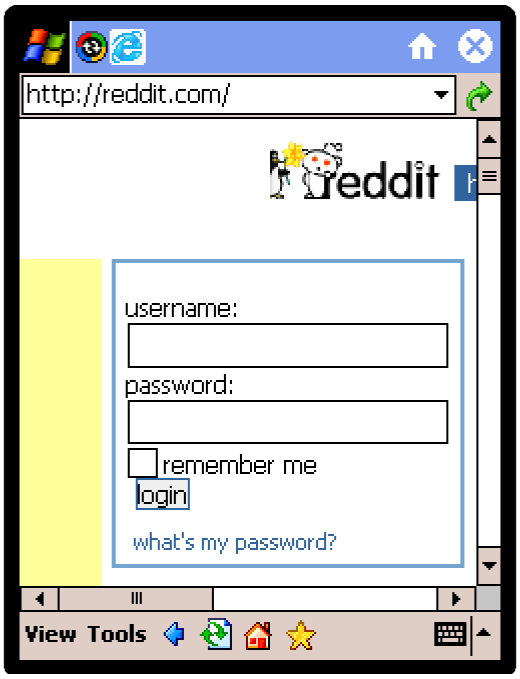
(Microsoft Corporation, 2006)

I will for these examples use the default layout viewing property.

#### 6.3.2.1 Handling: Content Larger Than its Display Container

It is not uncommon that web pages viewed in Pocket Internet Explorer are larger that the available screen space. Experience shows that this is more the rule than the exception. Almost only paged specially designed for mobile devices fit their smaller screens.

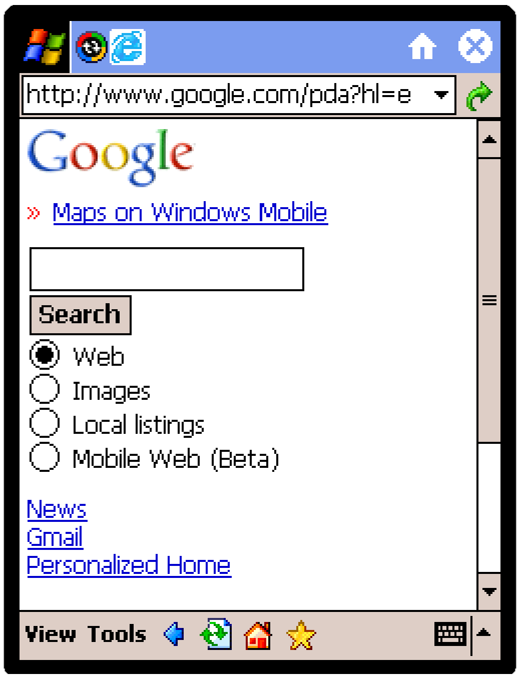
In default layout mode, web documents that are larger than the viewing panel, is mainly handled by adding scroll bars, both horizontal and vertical ones. There is also some form of size reduction done on images in these web documents, to make the content fit more properly, but that is for the most part it.



*Screen shot 6.16 - A web document shown in Pocket Internet Explorer that is larger than its display container.*

### 6.3.2.2 Handling: Switching Between Portrait and Landscape Screen Orientation

Switching between portrait and landscape screen orientation is handled in a straightforward way. If the content in some way exceeds the available display panel, when switching between the two orientations scroll bars are added or adjusted, either horizontally, vertically or both.


http://www.google.com/pda?hl=e

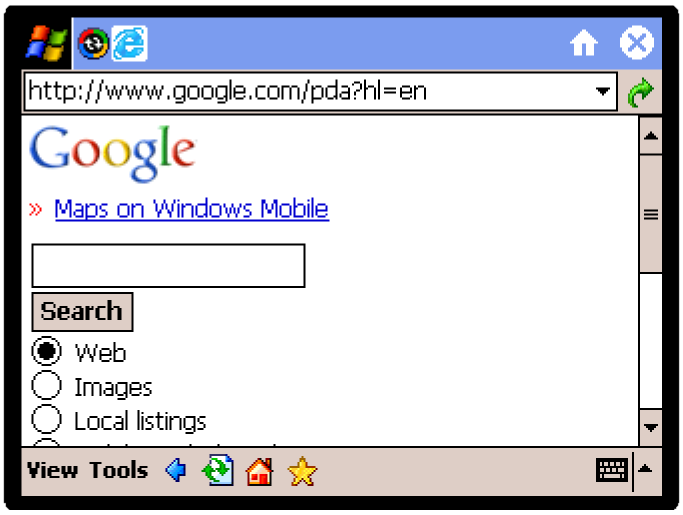
Google

» Maps on Windows Mobile

[✓ Web]
Images
Local listings
Mobile Web (Beta)

News
Gmail
Personalized Home

*Screen shot 6.17 – A web document showed in a portrait orientation.*


http://www.google.com/pda?hl=en

Google

» Maps on Windows Mobile

[✓ Web]
Images
Local listings

*Screen shot 6.18 - A web document showed in a landscape orientation.*

## 6.4 Graphics Based Applications

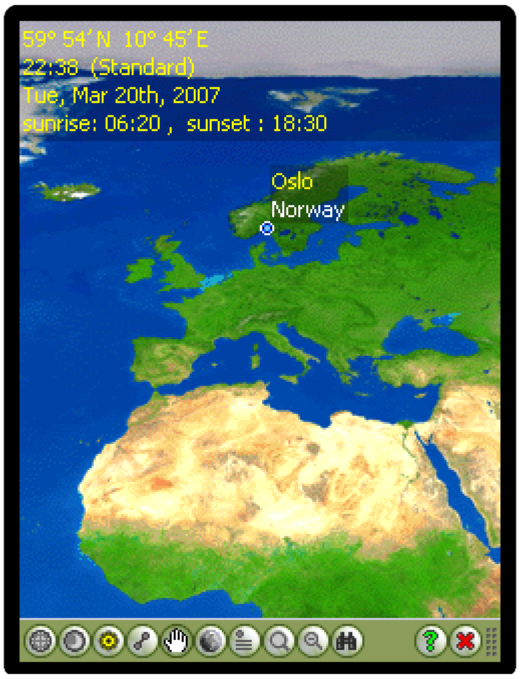
### 6.4.1 Pocket Earth 3.4

Pocket Earth (http://www.bluepointstudios.com/www/?fuseaction=product.poe) is a GIS application aimed at giving and displaying different information about the earth. The information is on fairly a abstracted level and the purpose is to get a overview of the earth with its major cities, time zones, sunrise and sunset, sun and moon azimuth and elevation, etc. It is developed by Blue Point Studios.

The dominating part of the user interface shows a graphical map normally accompanied by information of some sort. The tool bar at the bottom can be hidden so that the whole screen is used for a map projection.

#### 6.4.1.1 Handling: Switching Between Portrait and Landscape Screen Orientation

Pocket Earth has no way of dealing with switching between portrait and landscape screen orientation. The application user interface is drawn the same way with the same orientation (portrait wise) independent of what screen orientation is set as the preferred one in the operating system.


59° 54' N 10° 45' E
22:38 (Standard)
Tue, Mar 20th, 2007
sunrise: 06:20 , sunset : 18:30

*Screen shot 6.19 – Example of how the Pocket Earth user interface is drawn both in a portrait and landscape orientation (no difference).*

### 6.4.2 SimCity 2000

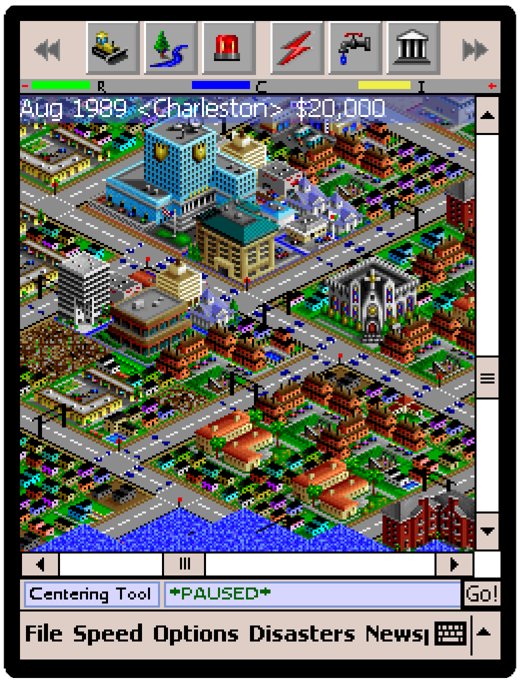
This is primarily a "building" game, where you create and try to increase the size of your cities, and rescue it from earthquakes and other natural disasters. The main goal of SimCity 2000 for the Pocket PC is to design, manage, and maintain the city of your dreams. Here the rules to learn are based on city planning, resource management, factors influencing land value, human factors, strategies for dealing with disasters, unemployment, crime and pollution, and the quality of life in a city.

SimCity 2000 (for PC) was originally developed by Maxis, and was licensed to ZIO Interactive, Inc. who ported it to the Pocket PC platform. It is distributed by ZIOSoft, Inc.

The user interface is build around a main panel, where one can view the city and surrounding area. This panel occupies almost all of the available screen space. On top of this main panel is a tool bar and at the bottom a menu.

### 6.4.2.1 Handling: Finger Optimized User Interface

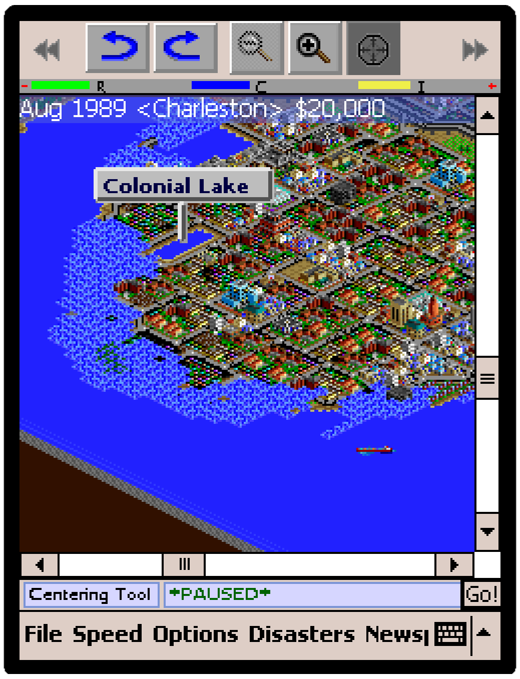
The user interface is not especially optimized for finger based touch interaction. In spite of this, it is relatively usable. Then interaction is mainly done by selecting function and touching where to operate. The buttons, with the key functions, in the top tool bar are relatively large, and they are not that difficult to hit with a fingertip. It is also possible to navigate around in the main panel showing the city, but the precision is coarse. This can be a problem when one for instance wants to select a specific number of squares and alter these. Furthermore the bottom menu is quite undersized for fingertip-use. It is usable, but just.


R
C
I
Aug 1989 <Charleston> $20,000
Centering Tool
*PAUSED*
Go!
File Speed Options Disasters News

.
*Screen shot 6.20 – An example of a SimCity 2000 game in action.*

### 6.4.2.2 Handling: Content Larger Than its Display Container

This is solved in a relatively straightforward and anticipated manner. In SimCity 2000 the map is always larger than the available screen space, independent of zoom level. And because of this, horizontal and vertical scroll bars have been added in order to render it possible to see every part of the map. It is simply a case of adjusting these scroll bars.


Aug 1989 <Charleston> $20,000
Colonial Lake
Centering Tool
*PAUSED*
Go!
File Speed Options Disasters News

*Screen shot 6.21 - SImCity 2000 zoomed out as far as possible, with scroll bars.*

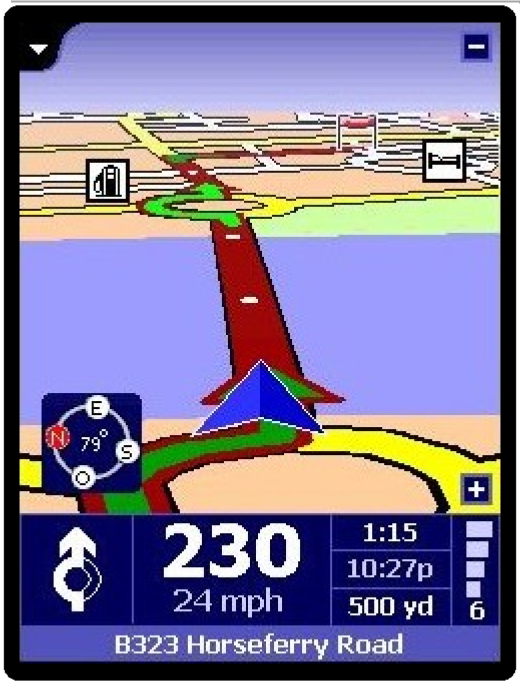
### 6.4.3 TomTom NAVIGATOR

TomTom (http://www.tomtom.com/) is a Netherlands-based maker of navigation systems for automobiles, motorcycles, personal digital assistants, and mobile phones. TomTom delivers both various portable mobile device models for automobile navigation (e.g. TomTom GO and ONE), and software for users with their own PDA or Smartphone (TomTom NAVIGATOR, which will be commented here).

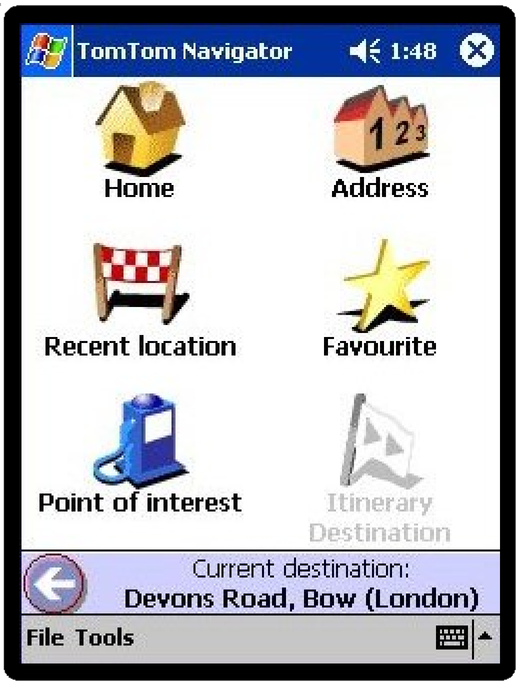
The main feature of the TomTom applications – i.e. guiding the driver from current position to a predefined destination – has a user interface which is graphic based. It consists of a map, shown in a top-down or 3D projection, and information like current speed, distance to destination, etc, shown in a separate panel docked to the bottom of the display. Though, when defining destination, requesting alternative route, ergo selecting a function or accessing a feature, the user interface conforms to an icon based user interface category.

#### 6.4.3.1 Handling: Finger Optimized User Interface

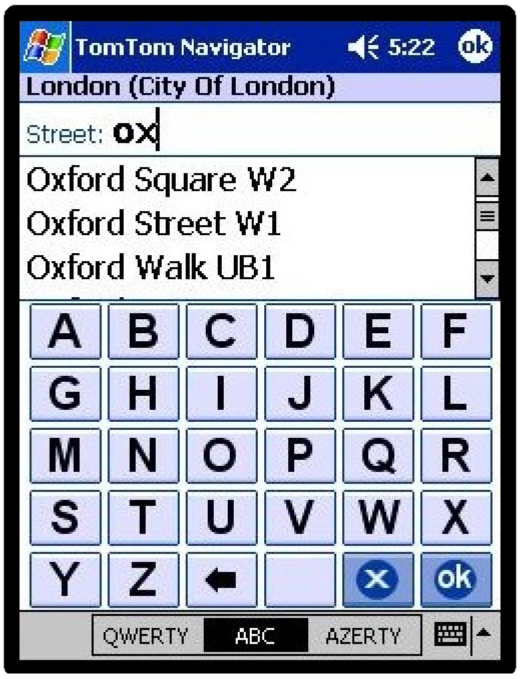
As with all TomTom solutions, TomTom NAVIGATOR is built to support finger based interaction. Interaction supportive objects, like icons, list items, buttons and keys in the software keyboard, are large, with well-sized fonts, roomy spaced and easy to hit with a fingertip. When entering a destination with the software keyboard, an auto-completion function is activated, and fitting destinations are shown as items in a list (as shown in *Screen shot 6.24*).



Screen shot 6.23 - Following a driving route with a 3D map projection in TomTom NAVIGATOR.



Screen shot 6.22 - Selection a function, represented by an icon, in TomTom NAVIGATOR.



Screen shot 6.24 - Entering and finding a destination in TomTom NAVIGATOR.

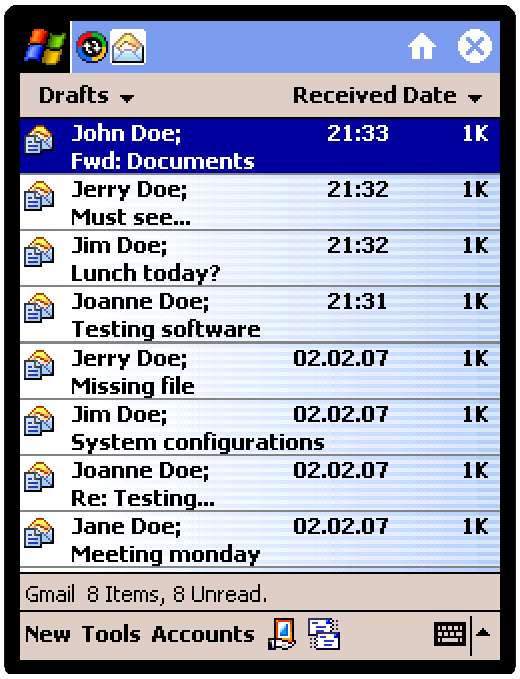
## 6.5 Repetition based applications

### 6.5.1 Pocket Messaging

This application is developed by Microsoft and distributed together with the Windows Mobile 2003 Second Edition. The main feature of this software is handling; receiving, reading, sending, etc., various types of messages such as e-mails, mms and text messages. A repetition based user interface can be found when accessing a set of messages, for instance e-mails. All messages are organized as a list with the elements listed downwards. One element represents an individual message and by default display information like recipient, received date, message size and title.

#### 6.5.1.1 Handling: Switching Between Portrait and Landscape Screen Orientation

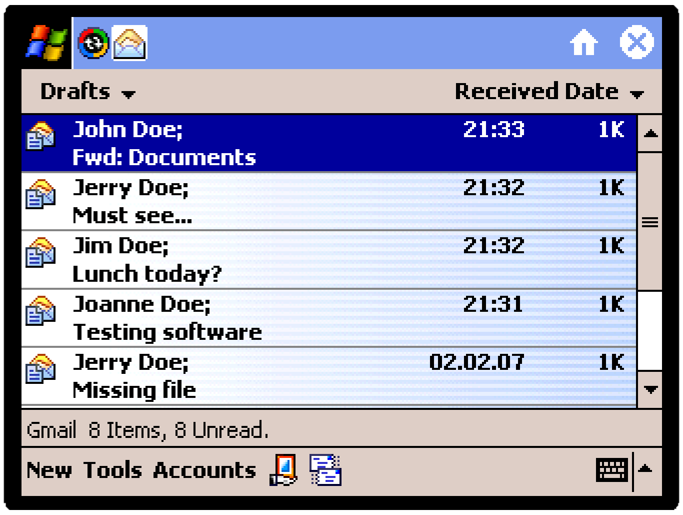
When switching form a portrait to a landscape orientation the available width increases and the height decreases. This means that there is less room for stacking the list items but more space widthwise for textual information. This extra width is not taken advantage of. The only measure that is taken is adding or adjusting the vertical scroll bar if the list height exceeds the available displaying area.


Drafts ▾ Received Date ▾

John Doe; 21:33 1K
Fwd: Documents
Jerry Doe; 21:32 1K
Must see...
Jim Doe; 21:32 1K
Lunch today?
Joanne Doe; 21:31 1K
Testing software
Jerry Doe; 02.02.07 1K
Missing file
Jim Doe; 02.02.07 1K
System configurations
Joanne Doe; 02.02.07 1K
Re: Testing...
Jane Doe; 02.02.07 1K
Meeting monday

Gmail 8 Items, 8 Unread.

*Screen shot 6.24 - A synopsis of unfinished drafts in Messaging showed in portrait orientation.*


Drafts | Received Date

| | | |
|---|---|---|
| John Doe; Fwd: Documents | 21:33 | 1K |
| Jerry Doe; Must see... | 21:32 | 1K |
| Jim Doe; Lunch today? | 21:32 | 1K |
| Joanne Doe; Testing software | 21:31 | 1K |
| Jerry Doe; Missing file | 02.02.07 | 1K |

Gmail 8 Items, 8 Unread.

*Screen shot 6.25 - The same synopsis shown in a landscape orientation.*

#### 6.5.1.2 Handling: Content Larger Than its Display Container

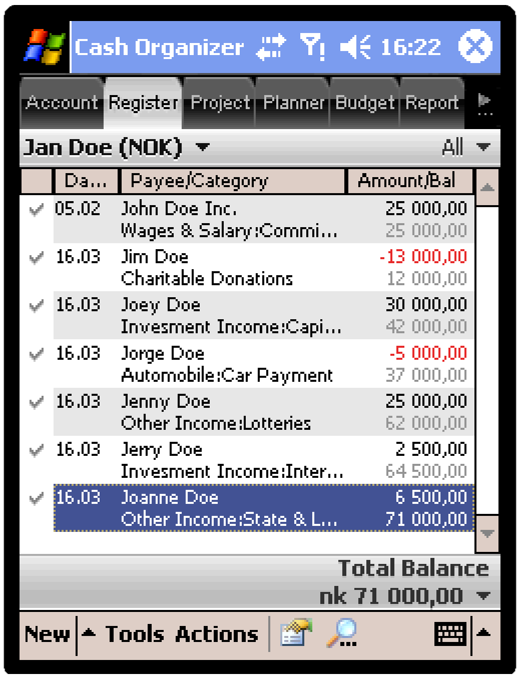
When the list of messages exceeds available screen space – the same thing is done as mentioned above in landscape orientation when display container gets to small – vertical scroll bars are added or adjusted. It is not possible to make the content exceed the available width. If for instance a title is longer than the width the title projection is just shortened down.

### 6.5.2 Cash Organizer 2007 Premium

Cash Organizer 2007 Premium also has user interface instances that are of a repetition based nature (together with form based). This is due to the fact that, when showing different summaries or overviews, the user interface shifts into a repetition based appearance. For instance, the user interface take on properties which are typical repetition based when viewing registered transactions. Each registered transaction is shown as a line or item in a list. The information in each element consists of the date registered, payee and category and amount and balance. On top of this list is a series of tabs for different areas or functions in Cash Organizer. In the bottom part one finds a summary field and below that a menu.

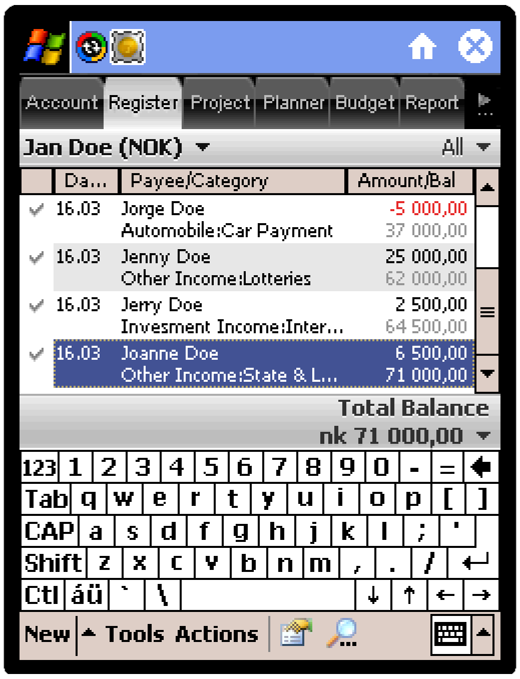
#### 6.5.2.1 Handling: Software Keyboard Showing/Hidden

When the software keyboard is activated it pops up between the bottom menu and the summary line, separating these two. The panel where the registered transactions are shown is shrunken. If the panel with the list of items then fills up more than the available displaying area, a vertical scroll bar is added to the left side of this panel.



| | Da... | Payee/Category | Amount/Bal |
|---|---|---|---|
| ✓ | 05.02 | John Doe Inc.<br>Wages & Salary:Commi... | 25 000,00<br>25 000,00 |
| ✓ | 16.03 | Jim Doe<br>Charitable Donations | -13 000,00<br>12 000,00 |
| ✓ | 16.03 | Joey Doe<br>Invesment Income:Capi... | 30 000,00<br>42 000,00 |
| ✓ | 16.03 | Jorge Doe<br>Automobile:Car Payment | -5 000,00<br>37 000,00 |
| ✓ | 16.03 | Jenny Doe<br>Other Income:Lotteries | 25 000,00<br>62 000,00 |
| ✓ | 16.03 | Jerry Doe<br>Invesment Income:Inter... | 2 500,00<br>64 500,00 |
| ✓ | 16.03 | Joanne Doe<br>Other Income:State & L... | 6 500,00<br>71 000,00 |

*Screen shot 6.26 – Registered transactions in Cash Organizer without the software keyboard showing.*

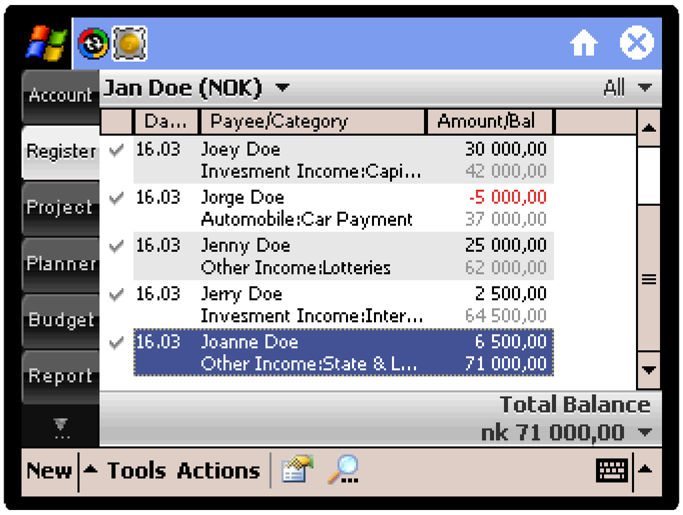


| | Da... | Payee/Category | Amount/Bal |
|---|---|---|---|
| ✓ | 16.03 | Jorge Doe<br>Automobile:Car Payment | -5 000,00<br>37 000,00 |
| ✓ | 16.03 | Jenny Doe<br>Other Income:Lotteries | 25 000,00<br>62 000,00 |
| ✓ | 16.03 | Jerry Doe<br>Invesment Income:Inter... | 2 500,00<br>64 500,00 |
| ✓ | 16.03 | Joanne Doe<br>Other Income:State & L... | 6 500,00<br>71 000,00 |

*Screen shot 6.27 - Registered transaction with software keyboard activated.*

### 6.5.2.2 Handling: Switching Between Portrait and Landscape Screen Orientation

When the registered transaction user interface is ported from a portrait to a landscape orientations several things are done to adapt. Firstly, the panel showing the list of registered transaction increases in width and in decrease in height, this in order to fill out the differently shaped area. Secondly, the tab control which in portrait mode was placed above the registered transaction panel is now moved to the right side. By doing this the registered transactions panel get more height to unfold itself in. Thirdly and if necessary, a vertical scroll bar is added if the list exceeds the total height of the panels it is placed in.


Jan Doe (NOK) All

| | Da... | Payee/Category | Amount/Bal |
|---|---|---|---|
| ✓ | 16.03 | Joey Doe<br>Invesment Income:Capi... | 30 000,00<br>42 000,00 |
| ✓ | 16.03 | Jorge Doe<br>Automobile:Car Payment | -5 000,00<br>37 000,00 |
| ✓ | 16.03 | Jenny Doe<br>Other Income:Lotteries | 25 000,00<br>62 000,00 |
| ✓ | 16.03 | Jerry Doe<br>Invesment Income:Inter... | 2 500,00<br>64 500,00 |
| ✓ | 16.03 | Joanne Doe<br>Other Income:State & L... | 6 500,00<br>71 000,00 |

Total Balance
nk 71 000,00

Screen shot 6.28 - Registered transaction in landscape orientation.

## 6.6 Summary

For the *Switching between portrait and landscape screen orientation* challenge the most regularly way or handling this was by adding letting the affected content stay the same but add scroll bars (mostly vertically, but also horizontally) to make the content accessible. In some cases certain components or elements were temporarily removed from the user interface. The components temporarily removed were typically ones showing information which was not crucial when in the current application state.

In case of the challenge *Software keyboard showing/hidden* the solutions were much the same as with *Switching between portrait and landscape screen orientation*. Scroll bars were in most case added to maintain accessibility when the content needed more space than what was is available in its viewing panel. In some occurrences components or elements were also temporarily removed to make room for the software keyboard. Some of the applications simply did nothing, and just let the software keyboard slid on top of the user interface.

*Content larger than its display container* was a challenge that occurred frequently in most of the tested applications. It was often a challenge which materialized as a result of the other challenges like for instance *Switching between portrait and landscape screen orientation* or *Software keyboard showing/hidden* arising. The definitely most used way of handling this was by adding or adjusting scroll bars. In some instance nothing was done, like with the Pocket Earth applications, where content always is larger than its display container and accessing content outside of the current projection is done by *Panning* (chapter 7.4.9 – page 74).

As for *Finger optimized user interface* challenges, only TomTom NAVIGATOR of the application tested where specially optimized for fingertip touch interaction. Some of the other applications were more usable than others, mainly due to the fact that the interactions mechanisms where larger in size. It was clear that all other the tested applications were constructed mainly for interaction via a stylus and other accurate pointing devices.

The possibility of *Switching between stylus and finger optimized user interface* were, as with the *Finger optimized user interface* challenges, not present. Nowhere could we find a way to switch between these two. Most of the applications did not even have the possibility of increasing and reducing the used user interface font sizes, which could at least been an ad-hoc way of toggling between a stylus and finger optimized user interface.

In the case of the *User interface able to run on equipment with different screen size* challenge, we were not able to test this because we did not have access to mobile devices with varying screen sizes and resolutions.

In general the solutions found for handling the different challenges were fairly basic. Most of the solutions for handling the various challenges examined were the same as the ones one would find on normal desktop computer applications, just scaled down to the context of a mobile device. The plasticity, the capacity of a user interface to withstand variations of both the system physical characteristics and the environment while preserving usability (Thevenin, et al., 1999), was for the most part reduced. Not all challenges encountering the different user interface categories need an advanced solution in order to keep usability, but when it was needed it was for the most part not dealt with accordingly.

# 7 GUIDELINES

The guidelines presented here follow a given structure. On the top level they are grouped under the six common user interface challenges:

7.X – User Interface Challenge (no. 1 - 6)
  7.X.X – Guideline (no. 1 - n)
    *Relevant for user interface categories: X, Y and Z based*
    Various information (e.g. description, pros and cons and recommended design approach if any)

Under each challenge suited guidelines are presented. Again, under each guideline, relevant user interface category or categories are commented. Hereafter, various information is listed which aim to give insight into the pros and cons of possible solutions, practical advice when designing user interfaces and having to handle the various challenges, and eventually, recommended design approach.

There is normally more than one way to handle a challenge, and this is reflected in the guidelines presented here. The different solutions are served together with their key advantages and disadvantages. The guidelines are not absolute and the solutions presented can in many instances be used across the various user interface challenges and categories, and also for handling challenges outside what is commented here.

Some solutions might use techniques, components or component properties which are not supported on the various mobile devices, development platforms, operations systems, design approach, etc. This is commented when necessary.

The guidelines or solutions presented vary in degree of difficulty, require various amount of work and the cost of implementing them thereby differ. As the different design approaches has various pros and cons, some can be more suited for handling one solution or guideline than others. If a specific design approach has advantages over the other (see *Problem Scope* – chapter 3 – page 21 – for more information) this will be discussed and commented under how a challenge should be handled.

## 7.1 Guideline and Category Matrix

The matrix presented below gives an overview of which challenges that are relevant for which user interface categories. The leftmost column refers to *The Concrete Challenges* (chapter 5.2 – page 31) by using the chapter number (5.2.1 to 5.2.6 ). The column placed to the right of this are the provided guidelines. The header row is each of the five user interface categories.

| | | Form | Icon | Document | Graphic | Repetition |
|---|---|---|---|---|---|---|
| 5.2.1 | *Switching Between Portrait and Landscape Screen Orientation* | | | | | |
| | Have Two Versions of the User Interface | √ | | | √ | |
| | Dynamically Resizing Components | √ | | | | |
| | Lock User Interface in One Orientation | | | | √ | |
| | Add or Adjust Scroll Bars | | √ | √ | √ | √ |
| 5.2.2 | *Software Keyboard Showing/Hidden* | | | | | |
| | Display on Top of User Interface | √ | | | √ | √ |
| | Use One Large Component as a Buffer | √ | √ | √ | | |
| | Having Two Versions of the User Interface | √ | | | | |
| | Add or Adjust Scroll Bars | | √ | √ | √ | √ |
| 5.2.3 | *Content Larger Than its Display Container* | | | | | |
| | Struts and Straps | √ | | | √ | √ |
| | Conceal Non-Vital Elements | √ | | | | |
| | Group Similar Content in Tab Folders | √ | | | | |
| | Partial Tab Folders | √ | | | | |
| | Wizard Based User Interface | √ | | | | |
| | Collapsible Blocks of Information | √ | | | | |
| | Add or Adjust Scroll Bars | | √ | √ | √ | √ |
| | Alter Icon Representation | | √ | | | |
| | Panning | | √ | √ | √ | √ |
| | Borders Facilitate Scrolling | | √ | √ | √ | √ |
| | Point and Center | | √ | √ | √ | |
| | Offer Zooming Functionality | | √ | √ | √ | |
| | Navigational Buttons | | | | √ | |
| | Manipulate Document Layout and Content | | | √ | | |
| | Restrict Number of Attributes | | | | | √ |
| | Optimize the Sequence and Size | | | | | √ |
| | More Than One Row Per Item | | | | | √ |
| | Show Additional Information When Item is Selected | | | | | √ |

| | | | | | | |
|---|---|---|---|---|---|---|
| 5.2.4 | *Finger Optimized User Interface* | | | | | |
| | Simple Adaptation of Existing Components | √ | √ | | | √ |
| | Advanced Adaptation of Existing Components | √ | | | | |
| | Develop Custom Components | √ | | | √ | |
| | Change Scrolling Mechanism | | √ | √ | √ | √ |
| | Manipulate Content | | | √ | | |
| 5.2.5 | *Switching Between Stylus and Finger Optimized User Interface* | | | | | |
| | Have Two Versions of the User Interface | √ | | | √ | √ |
| | Facilitate Switching Between Projections | | √ | √ | | |
| | Have a Set of Mapping Rules | √ | √ | | √ | |
| 5.2.6 | *User Interface Able To Run on Equipment with Different Screen Size* | | | | | |
| | Create Several Versions of the User Interface | √ | | | √ | |
| | Change Between Alternative Controls | √ | | | | |
| | Use a Model Based Approach | √ | √ | √ | √ | √ |

## 7.2 Switching Between Portrait and Landscape Screen Orientation

### 7.2.1 Have Two Versions of the User Interface

Relevant categories: *Form and Graphic based*

A solution is to have two specialized versions of the user interface (Nilsson, 2005), one adapted for a portrait orientation and another for landscape orientation. These two predefined user interfaces will be created at design-time for a specific application on a particular mobile device. Switching between the user interfaces is done at run-time and might be triggered manually by the user or by the operating system when an orientation switch is activated.

The main advantage of this approach is the possibility of tailoring the portrait and landscape user interfaces to maximize the usability in the two variants. There are some disadvantages that are worth mentioning; firstly, the added development work when constructing two separate user interface versions for one single use and, secondly, making sure to keep the two user interface versions consistent with each other so that the user do not get confused when switching from one to another.

Switching between the various user interface instances can be an event triggered by different occurrences. It might be the user who triggers the change by manually changing the setting in the operating system (as on the Qtek 9090) or the can be sensors in the mobile device which detects how the device is oriented and the automatically executing the change (a feature on for instance the Qtek 9000). It is also a possibility to have to facilitate the change by manually changing a property within the application. This can for example be done via a pull-down menu, a dedicated checkable button or a check box.

To illustrate, we may have an application, with form based user interface, where one window consists of a simple input form (type of application or input data is irrelevant). In portrait mode the user interface is designed, as shown in *Illustration 7.1*, with three single lined text input fields with appurtenant text labels, one multiline text field with text label, an options field with four options to choose from and a “submit” and “cancel” button. All controls are drawn within one guiding column.

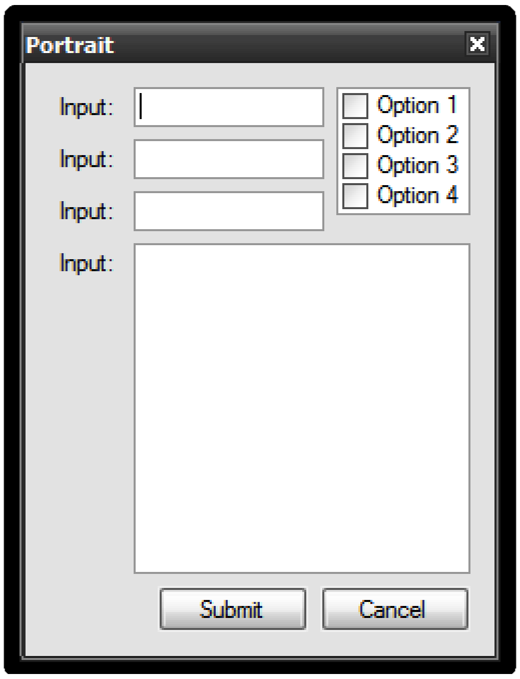


*Illustration 7.1 - Example of how an input form in an application might look designed with portrait orientated layout; with all controls drawn within one guiding column.*

When changing to landscape orientation, simply put, the width increases and the height decreases. And just rotating the layout designed for portrait mode, would result in several (key) controls being partially or fully placed outside the available screen area, making them difficult or impossible to use. To utilize the new context, one can place the controls using two guiding columns instead of just one, hereby taking advantage of the extra with.

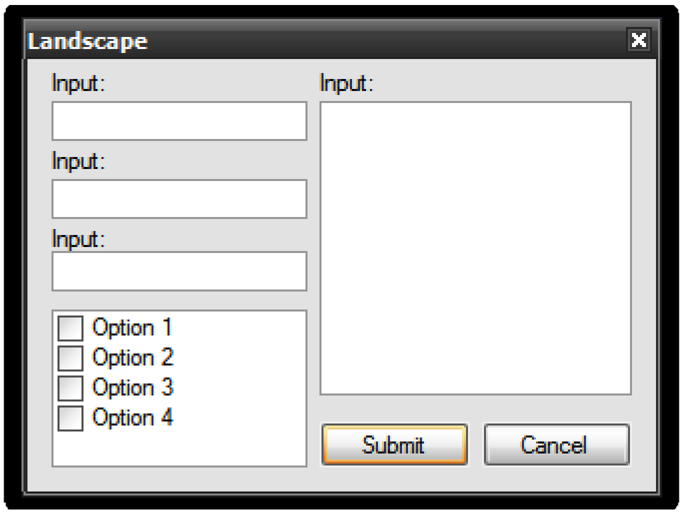


*Illustration 7.2 - The same input form, now with a layout taking advantage of a landscape oriented context with increased width.*

Having two versions of the user interface is something which a drawing based design approach can be suited for. It can be quite easy and fast two draw up two consistent user interface instances and then simply connect them to the same lower-level services. A model based approach could also, in an ideal situation, automatically create these two user interface instances from user interface model.

### 7.2.2 Dynamically Resizing Components

Relevant categories: *Form based*

Another way of handling the different contexts, when changing between portrait and landscape orientation, is to dynamically resize (Nilsson, 2005) the components in the user interface. This can be implemented using two approaches.

One solution is to predefine resizing rules for each separate window stating how the switch is to be handled, and then apply that as custom-made code for each window. This is quite simple to implement, but it must be done for each instance where it is deemed necessary and thereby it enforces the same or similar work to be done repeatedly.

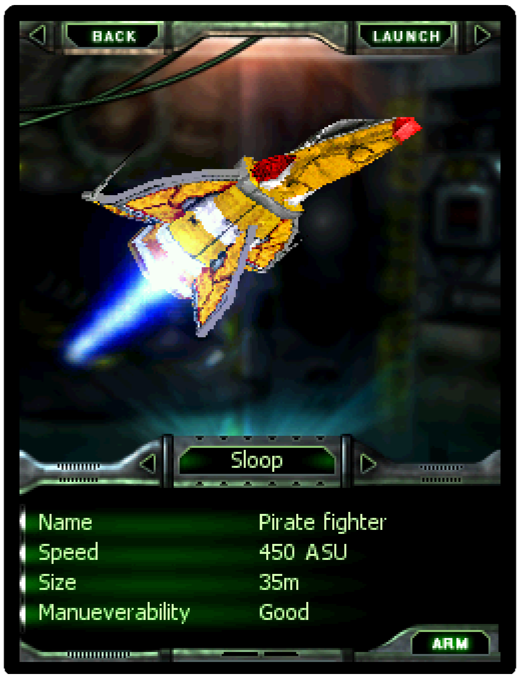
A second solution is having a general layout adjustment algorithm which handles the changes for all windows, much in the same way as one would implement *Struts and Straps* (chapter 7.4.1 – page 69). This will most likely initially require much more work than the first solution, and developing the layout adjustment algorithms can be quite complex. But once this platform is built, the workload when designing new user interfaces, even new applications, is significantly reduced. Still, the investment needed for adding a layout adjustment algorithm will seldom be viable just for handling switching between portrait and landscape screen orientation in a form based user interface.

### 7.2.3 Lock User Interface in One Orientation

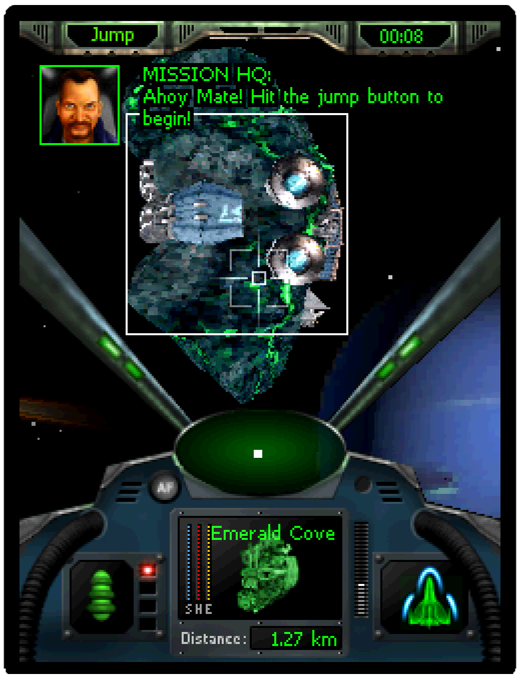
Relevant categories: *Graphic based*

A substantial segment of applications with graphics based user interfaces, especially games, are designed and locked to one orientation only and does not support switching between a portrait and landscape orientation (e.g. Anthelion by PDAMill, see *Screen shots 7.1* and *7.2*).

When the user interface is built up with and around complex and non-standard component, and the layout also diverges from what is regular, the most reasonable solution is often to not support an orientation switch. The work involved in developing these types of user interface with their custom components and layout often implies lots of graphical work, advanced programming and customization. The extra work that is needed to adapt these element to another screen orientation is in many cases not feasible when consider what is gained.



| | |
|---|---|
| Name | Pirate fighter |
| Speed | 450 ASU |
| Size | 35m |
| Manueverability | Good |



MISSION HQ:
Ahoy Mate! Hit the jump button to begin!

*Screen shots 7.1 and 7.2 - Two screen shots from the game Anthelion from PDAMill demonstrates how a complex, custom graphics based user interface can look. Anthelion does not support swithing between portrait and landscape orientation.*

### 7.2.4 Add or Adjust Scroll Bars

Relevant categories: *Icon, Document, Graphic and Repetition based*

It is by far the most commonly used mechanism for handling situations where the number of icons presented exceeds the available screen area. It is a mechanism which is quite simple to implement, in many instances it is built into relevant components, and it is supported by most PDA development platform.

A benefit of using scroll bars is that it is a well-established mechanism which the user probably is familiar with. Scroll bars also facilitates the possibility of scrolling at various speeds, and the in addition the elevator in a scroll bar gives information about which part of the total area is shown in the current projection and in what scale.

On the other side scrolling is something that should generally be avoided on mobile devices, particularly horizontal scrolling. As Erik G. Nilsson states it in (Nilsson, 2005):

> "*The main goal is often to avoid scrolling the data to find the desired information. If the projection is wrong (compared to the user's need), the user will not find the desired information, or horizontal scrolling is needed to locate it. If the selection is wrong, the user will not find the desired instance(s), or vertical scrolling is needed to identify it/them. As a rule of thumb, horizontal scrolling is worse than vertical, but both should be avoided – i.a. because operating a scroll bar using a stylus is difficult. The main reason why horizontal scrolling is "worse" than vertical is that the*

> *information on the same line usually has a closer connection than information on different lines.”*

Another con with scroll bars is when the user has to go to the scroll bar to shift the user interface, even by just one line; it takes the perceptual, cognitive and motor resources away from the target which the user focuses attention on, thereby breaking the work flow (Zhai, et al., 1997).

## 7.3 Software Keyboard Showing/Hidden

### 7.3.1 Display on Top of User Interface

Relevant categories: *Form, Graphic and Repetition based*

This may be the simplest way of handling the showing and hiding of a software keyboard, and normally requires little or no customization of the user interface. The software keyboard will for example be used when the user is entering data into, typically, a text entry field. So, for instance, the software keyboard might only pop up, covering the lower part of the screen, when an input field is selected for input.

The solution of displaying the software keyboard on top of the user interface will work adequately as long as the software keyboard don't conceal the current input field you are entering data into or any other vital user interface components. This will in effect say that the lower part of the screen, which the software keyboard occasionally occupies, should not contain any components that might require the software keyboard. It can also be very frustrating for the user if he or she has to manually hide the software keyboard repeatedly because it is concealing components.

A way to solve this problem – with the software keyboard covering vital components on the bottom third of the screen – can be to have the software keyboard slide down from the top part of the screen when a conflict is detected. There are two problems with this approach though. Firstly, the method of having the software keyboard pop up from the bottom part is a well-established concept and breaking with this can confuse the user. Secondly, this is a feature which is not presently supported in current PDA development platform, and therefore some custom programming is required.

### 7.3.2 Use One Large Component as a Buffer

Relevant categories: *Form, Graphic and Document based*

Another way of addressing the challenge of a software keyboard shown and hidden is to let the software keyboard us one large control as a buffer(Nilsson, 2005). Examples of controls that primarily can be used in this way, in a form based user interface, are *list boxes*, *multiline text boxes* and *tree views*. These can be large enough to give way for a software keyboard without totally disappearing. When the software keyboard is shown a selected control will shrink vertically to accommodate the new element. For this solution to be feasible, the user interface needs to contain such a large control.

In the case of an icon or document based user interface, it is the main panel itself (used for showing icons or a document) that naturally can be used as a buffer.

### 7.3.3 Having Two Versions of the User Interface

Relevant categories: *Form based*

See *Have Two Versions of the User Interface* (chapter 7.2.1 – page 62) for guidelines and more information.

### 7.3.4 Add or Adjust Scroll Bars

Relevant categories: *Graphic, Icon, Document and Repetition based*

See Add or Adjust Scroll Bars (chapter 7.2.4 – page 65) for guidelines and more information.

## 7.4 Content Larger Than its Display Container

### 7.4.1 Struts and Straps

Relevant categories: *Form, Graphic and Repetition based*

The concepts and model presented here is inspired by the available layout mechanisms in the *Java Swing* set (Sun Microsystems, Inc., 2006) user interface components.

Struts and straps (Nilsson, 2005) refers to the concept of having a set of constraints, or rules, which administrate how controls should act and adapt when the user interface environment changes. These constraints may govern various concepts within a user interface, e.g. the min and max size of controls, min and max space between controls, min and max font size, which side the user interface the controls should align to, and so on.

In the Windows .NET framework, one can find related but less advanced mechanisms in use, i.e. the controls may be either anchored to their parent component in any combination of their four sides or they can be docked to one side (or filling from the centre) of the parent. These features are unfortunately not available on the equivalent PDA development platforms of Swing and .NET Framework. This then imply that until such mechanisms will become available, developers can use them only as an inspiration for self-made layout mechanisms.

### 7.4.2 Conceal Non-Vital Elements

Relevant categories: *Form and Graphic based*

This guideline is based on concealing or removing non-vital element or elements from the screen. In a user interface some elements might be there just to inform or support the main features. Elements of this type may for instance be informative or explanatory text, redundant components (e.g. multiple navigation possibilities), etc.

Implementing this solution is fairly simple and is an easy way of adapting a user interface to available screen space. But it requires that the desired user interface have elements which can be sacrificed, something which is not a matter of course. One should also be careful when concealing or removing elements, also which only has an informative of explanatory purpose, as it might compromise the usability of user interface. Therefore one should consider if the usability gained by not exceeding the available displaying is larger than the usability lost when concealing selected element(s).

### 7.4.3 Group Similar Components in Tab Folders

Relevant categories: *Form based*

Instead of forcing too many elements into a limited screen space, a common way of dividing up a user interface into smaller more manageable chunks is by using tab folders. It can be difficult to provide absolute rules for when to use tab folders and when to instead use separate windows, but (Nilsson, 2005) purposes as follows:

> *"… as a rule of thumb separate windows should be used to split between different main concepts/classes while tab folders should be used to separate between different aspects of a single concept/class."*

To give an example, say we have an application used for organizing various transactions. To simplify it even more, say the functionality consists of two concepts: one, inputting new transactions and, two, providing an overview of multiple registered transactions. If there is sufficient screen space to use it might be viable to have these two concepts within one window; the top half might be for used as for entering new transaction and the remaining bottom half for showing the transactions overview. But as we here are talking about the challenge of handling content larger than its display container, this is not practical.

Following the rule of thumb given in (Nilsson, 2005), it would be natural to divide the two main functions – input and overview – into different windows, not tab folders, as we are handling two different concepts. However, should the input functionality alone exceed available screen space and it could naturally be divided into different aspects (like for instance personal information connected to a transaction on one side and date, type and value on the other), it would then be natural to divide this into separate tabs within the same window.

Be careful not to split or mix elements in such a way that the user has to navigate unnecessary back and forth between the tabs. This can confuse the user and decrease or even cripple application usability. Also important, when using tab folders, is to consider which elements that will be used more frequently than others, and let this direct how the tabs are arranged, for example with the most commonly used elements in the first tab and the least used last.

It should be pointed out that the tab folders themselves take up screen space, and on a small screen there is room only for a limited number of tabs. How many tabs is dependent of the text labels set in each tab. Should these tabs exceed available width or height (depending on where the tabs are docked: top, bottom, left or right) scrolling mechanisms are usually added, something which should be avoided. An alternative solution is to have more than one row of tabs, something one can find on normal desktop computers, but this is not supported on PDAs, at least not on the Windows Mobile platform.

### 7.4.4 Partial Tab Folders

Relevant categories: *Form based*

An application can have vital components or elements that need to be visible at all times in order to keep the application usable. One solution, which is a specialization of *Group Similar Components in Tab Folders* (chapter 7.4.3 – page 70), is to have partial tab folders. The vital components are here always shown, preferably grouped together in a unifying panel, and occupying one portion of the available screen. The remaining screen area, which is not used by the vital components, contains a tab folders control which can be used to alternate between the different aspects.

Firstly, it is important to recognize whether there are components or elements in the user interface which should or must be visible at all times. If there is more than one component or element which qualifies, it might be natural to group these together. Check if the remaining space if sufficient enough to hold a tab folders control. Should this space be too small one will once again end up with a new *display container smaller than its content* problem within an even smaller container, and the problem is not really solved.

Otherwise, take into account information and follow guidelines given for the idea presented in *Group Similar Components in Tab Folders* (chapter 7.4.3 – page 70).

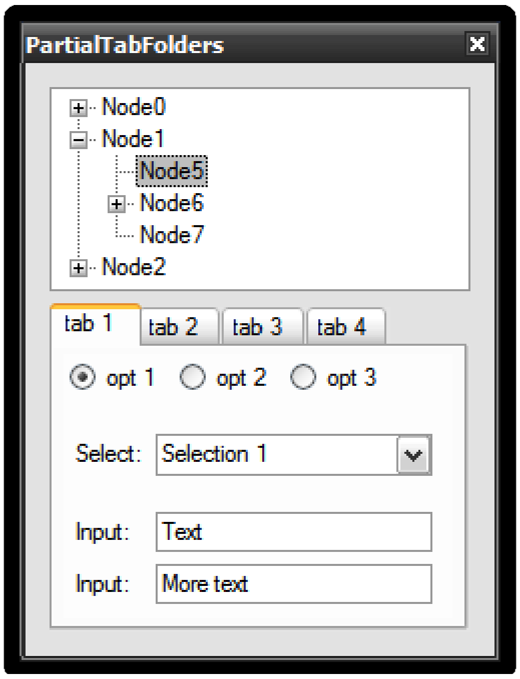


*Illustration 7.3 - Example of how a user interface utilizing partial tab folders can be materialized.*

### 7.4.5 Wizard Based User Interface

Relevant categories: *Form based*

Using a wizard based user interface is a solution which can be appropriate for types of user interfaces that are primarily process oriented (Nilsson, 2005), with just one (or a few) predefined paths leading from start till finish. This implies that a wizard based user interface is highly task oriented, something that makes it vital that the match between how the wizard progresses and the task at hand correspond naturally and fittingly.

A wizard based user interface is very step-by-step-oriented, as the current step must be filled out (correctly) before one can advance to the next step. This is normally used to either avoid certain errors or to avoid a lot input control. Thus this mechanism can often be best fitted for inexperienced users with limited or no knowledge of the application domain. But for some tasks, a wizard based user interface may be the best solution for all users.

When using a wizard based approach it is imperative to divide the underlying process into steps which corresponds with both the eventual natural process steps and the user's mental model of how this process works. Provide some form of indication to the user of where he or she is in the course of completing the wizard. Make it easy to navigate back and forth, without losing data. And finally, provide a clear indication of when the user is completing and submitting all entered data (maybe without the possibility of return).

Wizards are often offered as an alternative to a conventional instruction oriented user interface within a WIMP interaction paradigm.

### 7.4.6 Collapsible Blocks of Information

Relevant categories: *Form based*

Another variant of tab folders is to have blocks of components or elements that may be collapsed and expanded, a mechanism that can be found in most tree view controls. This is a mechanism primarily used on quite simple text nodes, in some specialized case accompanied by for example check boxes and/or icons. If one is to use this technique with more advanced controls like text fields, button, etc., this is something that most likely will require some form of custom development to create a component or mechanism adaptation, or tailor to this specific need, as neither the PDA development platforms of Swing nor the .NET Compact Framework currently supports this.

A benefit of being able to collapse and expand components, or components groups, is that it empowers the user to customize the user interface by choosing which elements that should be shown together.

If implementing such a solution it is necessary to consider and decide on how some properties should be. Should the user interface have a default appearance which is loaded each time the application started, or is more convenient to have to user interface remember the expand and collapse settings from when it was last run? What is considered right here, is dependent on several factors. The probably weightiest aspect here is whether the same instance of the application, on one mobile device is to be used but several different users, as they might prefer to customize the user interface differently, something which may favor a default appearance. Another solution to this challenge with multiple users can be to have users log in and out each with their own version of the user interface, but this is outside the context of this current challenge.

### 7.4.7 ADD OR ADJUST SCROLL BARS

Relevant categories: *Icon, Document, Graphic and Repetition based*

See Add or Adjust Scroll Bars (chapter 7.2.4 – page 65) for guidelines and more information.

### 7.4.8 ALTER ICON REPRESENTATION

Relevant categories: *Icon based*

A way to increase the efficiency and make the most of a (too) small displaying container, is to trim or reduce the how the icons are represented. There are two main ways of doing this: firstly, follow use a default way of displaying and arranging the icons (typically separate *large icons*, *icon list* and *detailed list*), and secondly, alter the icons themselves independent of icon display pattern.

In the case of choosing an icon display pattern, it is normally a case of choosing form a default set provided by the programming platform, e.g. within an icon-displaying container. An *icon list* is probably the icon display form which crams the largest number of icon instances into a display container. At the other end of the scale is the separate icons view with larger icons and therefore fewer icons inside the display container.

Altering the icons themselves can also be fairly simple task, as it is often just a case of altering the icon properties, something that is supported in most PDA programming platforms. There are several ways of achieving a more trimmed and space effective icon representation.

The most obvious is to *change icon size*. One can follow the most commonly used icon dimensions (see Icon Based – chapter 4.2 – page 25) or define custom proportions which fit your current context. The smaller the icon size the more can be shown in a display.

Icons are in most case shown together with text, normally the application name or filename of the file the icon fronts. This text is displayed connected to the icon in various ways, in example placed under, to the left or sometimes to the right or over. The latter two are rarely used. This icon text can also be adjusted in order to increase the efficiency of how the icons are shown. It is possible to *decrease the font size*, but be careful not to make the text too small and thereby rendering it difficult to read.

Furthermore the *text can be shifted to another related position* to the icons. Often it can be more space efficient to have the text placed to the right instead of under the icons. When the text is placed to the right of the icons each icon occupy less height making it possible to have more icons arranged vertically. If the text is placed under the icon will use less width, but only the width of the icon itself and the text is normally the dominant factor in contributing to the total width. Hereby the height is increased without a significant gain in width.

It can in some case be viable to remove the text. But then each icon in the user interface must have a unique graphical representation or another mechanism for separating and identifying icons must be used (e.g. tool tip or a text field at the bottom showing the text of a selected icon).

Lastly it is possible to reduce the white space between each icon. But reducing this white space should be done with moderation. If the white space between the icons gets to small it can be difficult to separate the icons and select and interact with one specific instance.

### 7.4.9 PANNING

Relevant categories: *Graphic, Icon, Document and Repetition based*

This is a solution which is widely used on various graphical information systems, both on the net – e.g. Google Maps (Google, Inc., 2007) and Map Quest (MapQuest, Inc., 2007) – and in desktop applications – e.g. NASA: World Wind (NASA, 2007) and Google Earth (Google, Inc., 2007) – but also in games – e.g. Worms World Party for Pocket PC by JAMDAT Mobile ULC. Panning is quite similar to the concept of drag and drop (Wikimedia Foundation, Inc., 2007), but here the user selects (by clicking or tapping and holding) at a point in the user interface or within a container, then drags in a direction until the wanted projection is shown and releases stopping the panning.

The idea of panning is something which is quite intuitive for most users (Nilsson, 2005) as it is based on a common interface metaphor of sliding something around in order to bring parts of something large into focus, for instance having a large news paper on a table, leaning over reading and sliding (or panning) this newspaper around depending on which article is interesting.

Another pro with using panning is that it does not require very precise pointing as long as the user hits within the panning-enabled component, thus making it usable for both stylus and finger based interactions. This prospect of just hitting within the container also eludes the problem of having to shift focus from the current work task, as mentioned for *Add or Adjust Scroll Bars* (chapter 7.4.7 – page 73).

It is worth noting that a problem can arise when a user interface has panning-ability. It is often necessary to interact with objects, and how do you alternate between being able to organize and handle objects and panning inside the same user interface? A solution to this can be to have some sort of way changing between these two modes, e.g. by having a checkable button or check box. The problem with this solution is that it, one, interrupts the work flow (as with scroll bars (Zhai, et al., 1997)) because the user has to shift focus and hit a button or check/uncheck a field, and two, the components facilitating the switching functionality itself occupies space leaving less room for the other elements to use.

A drawback is that this mechanism does not support faster scrolling than what is predefined. Nor does it provide the additional information the elevator in a scroll bar gives about which part of the available area is shown the current projection and in what scale. Though Apple has demonstrated an improved panning function in its Mac OS X on the new iPhone (Apple Inc., 2007; Wikimedia Foundation, Inc., 2007). Here it is possible to grab a hold of the user interface, accelerate it down- or upwards and have it scroll in that direction before it comes to a halt (referred to as a *touch-drag-lift motion*(Wikimedia Foundation, Inc., 2007)). How fast and for how long it scrolls is computed based on the speed and acceleration with which the drag motion is performed.

### 7.4.10 Borders Facilitate Scrolling

Relevant categories: *Graphic, Icon, Document and Repetition based*

This is a specialization of the typical scrolling functionality, but here only a predefined invisible or lightly shaded or lightened border or area close to the edges activates scrolling. When the user moves the pointing device close to an edge, the projection starts scrolling in the direction of that edge (e.g. moving the stylus down close to the bottom edge of the container starts a scrolling motion downwards).

One can find a variety of this solution in use on most normal desktop computers. It comes into action if the screen resolution is set higher than what the monitor supports. Then a “virtual desktop” is offered which is larger than the maximum screen resolution, and reaching the various sections of the desktop is done by moving the pointer to the edges. Another similar mechanism can be found in common document applications on desktop computers. If the mouse has a scroll wheel, this can be pressed changing the pointer function. You can then move the mouse up and down and the application automatically scrolls up or down. The scrolling speed will vary depending on how far up or down from the vertical center of the document projection you move the pointer. The longer away the faster the scrolling speed.

If implemented, as purposed with invisible or lightly shaded or lightened area, this will not reduce the size of what can be shown within a display container.

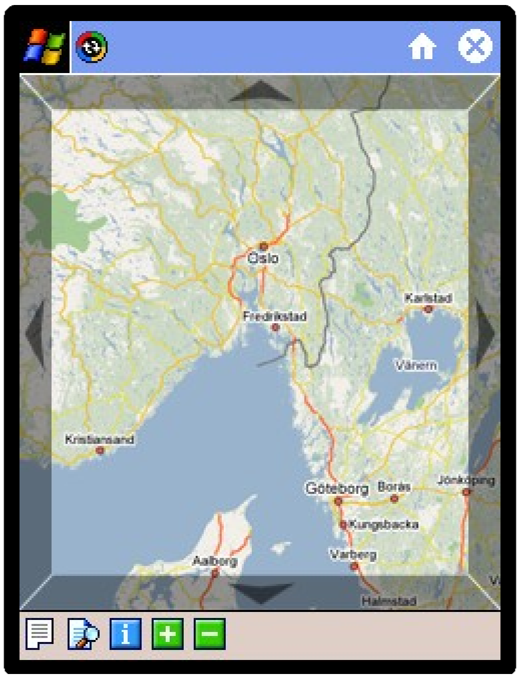


*Illustration 7.4 - Example of how a user interface can look when using edges facilitating scrolling.*

There are several things that need to be thought through when using such a mechanism.

Firstly, how large or thick these scrolling edges or boarders should be. If they are too small it can be difficult to stay within one scrolling edge area to continue scrolling, and if they are excessively thick this it will render the area for handling content too small. Secondly, this is a mechanism which is not widely used and therefore unknown for most users. A challenge is consequently: how to let the user know that this type of functionality is present and how it should be used. By having a subtle marking of the scrolling edge – by shading of lighting the area – can be a solution. There can also be a small or faint arrow inside the area, informing the user of which direction scrolling will go.

As with panning it is a drawback that this mechanism does not support faster scrolling than what is predefined. Nor does it provide the additional information the elevator in a scroll bar gives about which part of the available area is shown the current projection and in what scale. Though it could be feasible to have the scrolling edges be graded, i.e. the closer to the edge you touch the faster the scrolling will progress. This will probably require thicker scrolling edges in order to control the speed properly. The shading can also be graphically graded to visualize this, e.g. darker shading implies faster the scrolling.

This mechanism of edges facilitate scrolling is not present in current PDA development environments and thus it will require customization of current components or development from scratch.

For icon, document and repetition based user interface some adaption of the main idea should be done.

For **icon based user interfaces**, if there are a large number of icons, meaning that the available displaying are is widely exceeded, it can be very confusing to get an overview of where current projection are in this mass. Especially so if the scrolling goes both vertically and horizontally, as this makes it difficult to follow how the icons are arranged and find a specific instance. But if the scrolling goes either up or down or sideways, not all four directions at the same time, it can be easier to orientate and follow how the icons are arranged.

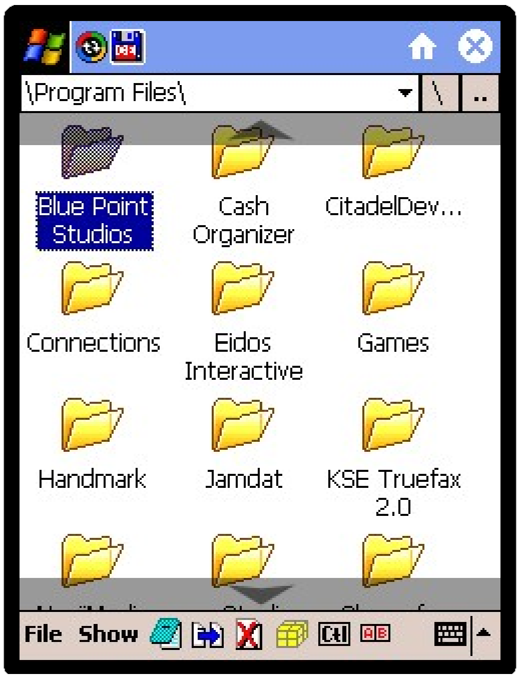


*Illustration 7.5 - Example of how a icon based user interface might look when using egdes facilitating scrolling up and down.*

With **document based user interface** the case is similar to scrolling edges in icon based user interfaces. Having to scroll both horizontally and vertically is will make reading the document very difficult. As text is structured in with words in lines, arranged either from left to right or vice versa, with the next consecutive line under, only vertical scrolling can be recommended in order to maintain reading consistency for the user.

A **repetition based user interface** is normally arranged in such a way that is natural to support scrolling either up and down or from left to right.

### 7.4.11 Point and Center

Relevant categories: *Graphic, Icon and Document based*

As with panning, a point and center mechanism is fairly common within mainly graphic based applications, mainly GIS application, but it can also be used in for example games (as done in for instance Warlords II for Pocket PC by Infinite Interactive PTY, Inc.), icon and document based applications. The idea is to point at or select a specific point in the user interface, for instance a place on map, and then use this new point as a center for a new projection. This is relatively intuitive for most users which are somewhat familiar with computers in general.

A problem with this approach occurs when there is a need to interact with object in the graphical user interface where the point and center function is in use. This can be solved in various ways.

For instance in a game, where you have objects to interact with and some form of background, the point and center mechanism can act only when the user taps at a point in the background. And when the user select an object of interests this objects properties are editable.  By doing this the user can keep his or her attention on the task at hand without been interrupted. A problem can arise if there are too many objects in the current projection making it difficult to hit the background and set the desired new center point.

Another way of handling this problem is to have the possibility of changing between functionalities, for example by have a tool bar, where one is the point and center functionality. This latter solution is widely used in GIS applications. A con with having a tool bar, checkable button, etc, is that these take up often precious screen space.

As with panning, it is a drawback that this mechanism does not support faster scrolling than what is predefined. Nor does it provide the additional information the elevator in a scroll bar gives about which part of the available area is shown the current projection and in what scale.

### 7.4.12 OFFER ZOOMING FUNCTIONALITY

Relevant categories: *Graphic, Icon and Document based*

This is a feature most commonly used on both graphical and document based content, but it can also be implemented on an icon based user interfaces. The possibility of zooming is widely used, i.a. in GIS applications and word editors. The common way of using zoom is to center on a designated point and then zoom in and out depending on the detail level you desire. The zooming can be facilitated in various ways. One way is by having onscreen buttons, with a minimum of one control for zooming in and another for zooming out. Another way is to zoom in around a point where the user double-clicks. There is often a zoom scale showing the min and maximum zoom level and at what level you are currently on.

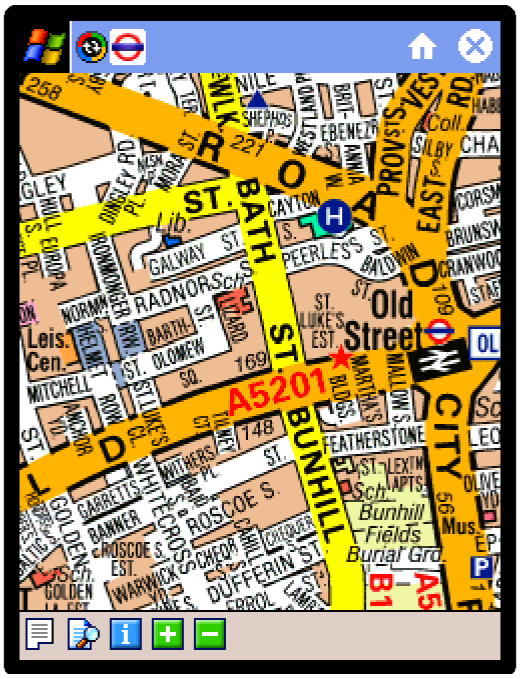


*Illustration 7.6 - Tube 2 for Pocket PC (v. 2.11) by Visual IT with zoom functionality.represented by a plus (zoom inn)  and a minus (zoom out) button at the bottom part of the screen.*

Zooming is often used in combination with for example scroll bars, panning, click and center or a combination of these. To make user interface using zooming more practical, an overview of the total area is provided shown as a small projection in a separate little window. This gives insight into what zoom-level and where in the total context the current projection is. Another quite common way of facilitating zooming is by marking a selection which then sets this selection as the new projection.

Google maps (Google, Inc., 2007) is a live example of a GIS solution combining *zooming*, *panning*, a *zoom scale*, (double) *click and center* and *navigational buttons* into one user interface. In the bottom left part one can also find a small window which informs about the bigger context of the current projection. This window can be hidden and shown.

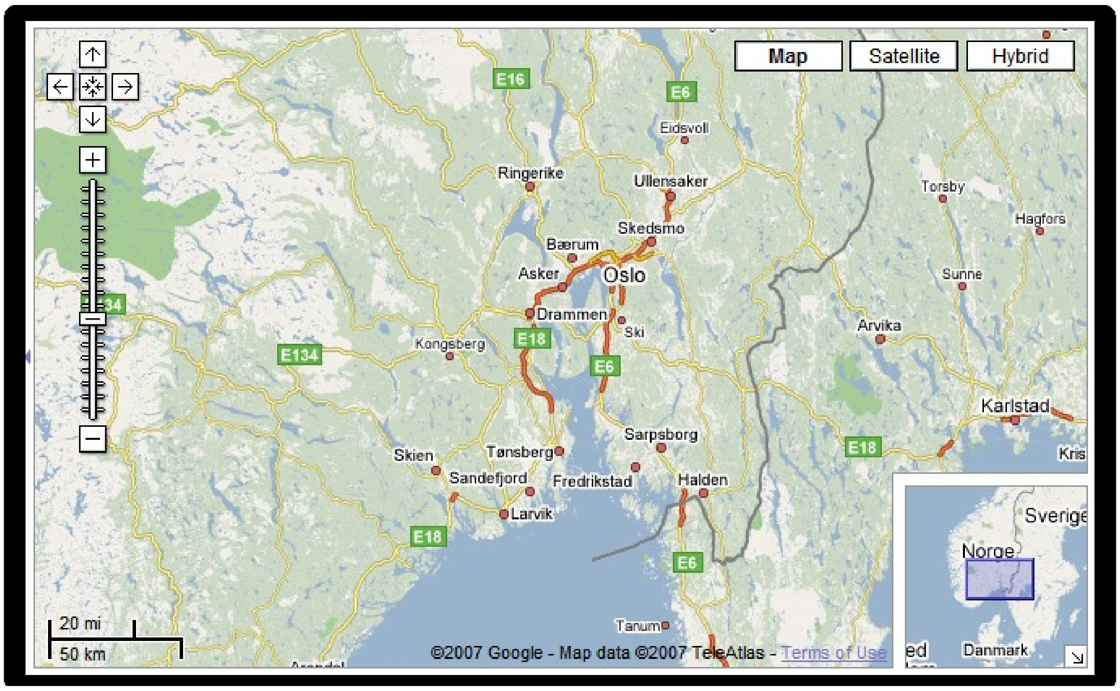


*Illustration 7.7 - Google Maps combining zooming, panning, click and center and navigational buttons, together with an overview map, in on user interface.*

### 7.4.13 Navigational Buttons

Relevant categories: *Graphic based*

A way to have the ability to move the projection around, but without the scroll bars, is to have navigational buttons. This is also a relatively common feature in both GIS and gaming applications. These navigational buttons are often formed arranged in a button-cross on-screen (as shown in the upper left corner of *Illustration 7.7*). Hitting one of the buttons will make the projection shift a predefined step in that direction. It is also often possible to hold a button down to continually scroll.

As with panning it is a drawback that this mechanism does not support faster scrolling than what is predefined. Nor does it provide the additional information the elevator in a scroll bar gives about which part of the available area is shown the current projection and in what scale. Some application do support accelerated scrolling when holding down a directional button longer than a set time span.

If the mobile device has a hardware directional pad (or similar physical directional button support), this is normally connected to how the on-screen directional pad operates or vice versa.

### 7.4.14 Manipulate Document Layout and Content

Relevant categories: *Document based*

Manipulating the document itself in a document based user interface is probably where there is most to be gained, as this is the dominant element of a document based user interface. But this should be done with caution as changing the document layout and altering elements within a document can make it difficult to read and interpret for the user. How the document is presented visually if often very important to the semantics. A document manipulating mechanism will normally be implemented in coherence with other solutions (e.g. scroll bars or edges facilitating scrolling) as it alone hardly ever solves the problem of content being larger than its display container without seriously compromising usability.

The techniques presented are often available features or properties in many of the documents based applications, e.g. editable via for example tool bars or pull-down menu options. The solutions presented are closely related to a normal zooming functionality, but here it is not a case of magnifying or shrinking the document as whole but distinct elements.

One solution is to have the text font size used in the document be proportional to the size of the available display area (resolution, size and width-height proportion). This can be done by reducing (and increasing) the font size and/or change to a more compact font. It is important to define upper and lower limits which specify when the font size cannot be lowered or raised any more, as without these limits the font size can become too small or large and thereby difficult to read. It is also vital to get the relation between available display area and fonts correct something which both screen resolution and size will influence.

Many documents consist of other elements than just text (e.g. images, animations and tables) and these can in some cases also be adjusted proportionally as purposed with text. Pushed to the extreme these non-textual elements can also be removed or hidden. This should be done with caution as removing such elements might take away vital information.

Another way to optimize how the documents appear is to reduce the whitespace between text lines (often referred to as line space). Reducing this space will make it possible to show more text in a limited area, but if the lines are set to close together it can compromise readability.

### 7.4.15 Restrict Number of Attributes

Relevant categories: *Repetition based*

This is a solution used to avoid horizontal scroll bars. As purposed in (Nilsson, 2005) for lists, this can normally be applied to many repetition based under interfaces as well. It is an idea which is quite obvious as it is just a case of reducing the number of element to a number which does not exceed the available display container. But in many case this solution is not realistic, mainly because the user may need more information than there is room for in order to perform the task the applications is supporting. Another problem with this approach is that it prerequisite that all user of that current application require exactly the same elements.

One solution, which follows the principle of restriction the number of attributes, is to let the user configure which attributes to be displayed at run-time. It the application facilitates multiple users; it is also possible to have some form of login system administrating this. This though requires additional work, and whether the benefit is greater than the extra work-load and cost must be considered for each unique case.

### 7.4.16 Optimize the Sequence and Size

Relevant categories: *Repetition based*

Erik G. Nilsson (Nilsson, 2005) also purposes a similar solution to the concept of restricting the number of attributes which is optimizing the sequence and size of attributes. Here attributes are removed or hidden, but the sequence is so that the most important attributes are placed within the visible area and the least important are placed furthest away. This can reduce the frequency of how often horizontal scrolling is used.

Here, as with restricting the number of attributes, it can be feasible to let the user configure how the attributes should be arranged.

### 7.4.17 More Than One Row Per Item

Relevant categories: *Repetition based*

This is also a way of avoiding, or restricting the use of, horizontal scrolling. The solution is to have one item use more than one row for its attribute instances. As is pointed out in (Nilsson, 2005) this is will visually work relatively well and it is normally not a problem to distinguish between the separate items. It can also make an item easier to select when using finger based interaction.

An obvious con with using more than one row for each item is that there is room for fewer items within the display area which can enforce more vertical scrolling.

Another drawback is the problem of visually connecting the attribute headers, it there are any, with the corresponding attribute instances within each item. The normal way of having one top header row with the attribute header will most likely not be usable. One way to solve this is to have the attribute header places in front of each attribute instance, but this again can clutter up the visually and even more space is needed for each item. Icons and/or abbreviations, representing each attribute header, can also be used, but this presupposes that the user understands or knows what these mean.

### 7.4.18 Show Additional Information When Item is Selected

Relevant categories: *Repetition based*

A fourth way of avoiding horizontal scrolling is by showing one or a few vital attributes like an item title, and when an item is selected the additional attributes instance can be shown. This can be done in different ways.

One is by following how a tree view works, and show the additional information as branches under the item title. Though this can be problematic as it might not fully utilize the screen width and if there are many attribute instances it can used much space vertically.

Another way is to show the additional information in a panel which slides out from the bottom part of the item row. Within this panel the information can be arranged in a way that is both space-efficient and visually usable.

A third way is to show the attribute instances as an appurtenant floating dialog box. This will require no rearranging of the original compacted items. A challenge can however arise if there is a need to show the detailed information of more than one item at the same time. A minimum requirement to support this is by using non-modal or modeless dialog boxes.

## 7.5 FINGER OPTIMIZED USER INTERFACE

### 7.5.1 SIMPLE ADAPTATION OF EXISTING COMPONENTS

Relevant categories: *Form, Icon and Repetition based*

This is a quite relatively cost efficient solution to facilitate a finger optimized user interface, though it can in some situations be somewhat limited. I order to make the user interface (more) "finger friendly" existing components are used, but appurtenant properties are adjusted. Exactly which components that works best, or is more adaptable, for finger based interaction can be vary from PDA platform to PDA platform.

Erik G. Nilsson in (Nilsson, 2005) presents, for the Windows Mobile platform, the following table which gives some characteristics of the standard components (i.e. the components that are designed for user interaction) with respect to what can be "finger friendly":

| **Component** | **Appropriateness for finger navigation** |
|---|---|
| *Button* | Standard Button size is a bit small, but given a bigger size, Buttons are OK for finger use. |
| *Text Box* | For entering text, it is sufficient to click on a Text Box – the rest of the interaction is done through some kind of text entry mechanism (see above). The latter may be far from trivial using the fingers, but this is outside the scope of this problem. Clicking on a Text Box 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). If the Text Box has a value already, selecting this value when the Text Box gets the focus will ease finger use. Changing the text in the Text Box – e.g. by selecting and changing three characters in the middle of the text – is not trivial using just the fingers. How difficult it is depends on the font size used, but increasing the font size too much may easily result in a number of other usability problems. |
| *Check Box* | Check Boxes are not too difficult to operate with fingers, depending on the distance to other Check Boxes (and other UI controls). To trigger a Check Box, not only the tick box, but the whole control (including the text and any additional space around the text) may be clicked. Increasing the size and/or the font size will not increase the size of the tick box. Thus, given large enough size and distance to other components, Check Boxes are easy to control using fingers. |
| *Radio Button* | Radio Buttons have the same characteristics with regards to finger friendliness as Check Boxes. As Radio Buttons always appear in groups, the distance/size requirements are especially important. |
| *Data Grid* | The finger friendliness of the control has not been investigated in depth, but the standard size of cells in the grid is fairly small, and it does not seem to be an easy way of making the cells larger. |

| | |
|---|---|
| *List Box* | A standard size List Box 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). |
| *Combo Box* | Combo Boxes have approximately the same characteristics with regards to finger friendliness as List Box. |
| *List View* | Using icons and Large Icons as View, List View may be appropriate to control with fingers (even though the control as such probably has limited applicability in many applications). |
| *Tree View* | A Tree View is difficult to control using fingers, and it does not seem like it is possible to make it more appropriate by adjusting its properties. |
| *Tab Control* | The tabs in a Tab Control 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. But all tabs should fit on the screen to avoid having to use the scrolling features of the Tab Control (which is not easy to operate using fingers). So there is a clear trade-off that need to be balanced. |
| *Scroll Bar* | Scroll Bars are notoriously difficult to use on a PDA, even with stylus. The Scroll Bars 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. |
| *Up Down* | There are two types of Up Down controls, one that can adjust numbers (spinner) and one that can adjust an arbitrary domain. These controls are identical with regards to finger friendliness. In their default size, they are almost impossible to operate using fingers, and there is no apparent way of adjusting their size or font size. |
| *Track Bar* | Track Bar is not specifically easy to operate using fingers, and there are no obvious ways of adjusting their properties to make them more suited. If there is a choice of direction, horizontal Track Bars are slightly easier to operate using fingers. |
| *Menu Item* | Menu Items (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 Menu Items. |
| *Context Menu* | Items in a Context Menu are the same UI controls as Menu Items in a main menu, and are used in the same way – but triggering a Context Menu is more difficult than a main menu using the finger, as the user must hit the control to which the Context Menu is connected. |

When finger optimizing a **repetition based** user interface it is normally a case of enlarging the items. If it merely is a case of facilitating selection of items, something as simple as increasing row height can solve the problem. If you, in addition to have a finger optimized user interface, want to avoid horizontal scrolling both of these challenges can be solved by using more than one row for an item representation as presented in *More Than One Row Per Item* (chapter 7.4.17 – page 82).

In some application it is possible, and even necessary, to edit attribute instances for an item. Then it is sometimes not sufficient to just increase the height of an item, the width might also have to be increased. A con with this is that it will make the total width larger, something which might trigger the need for or increase horizontal scrolling.

For **icon based user interfaces** the possibilities of altering an icon representation are the same as presented in *Alter Icon Representation* (chapter 7.4.8 - page 73). But here the main challenge is to facilitate finger based interaction, and not to utilize a display container which is smaller than the content of what is presenting. In general the objective is to make the icons large enough making the hittable by a fingertip. When using any of the default way of displaying and arranging the icons, the *large icon* projection is probably the best. Here the icons are larger and the whitespace between each icon is quite acceptable. All in all this projection is relatively “finger friendly”.

Furthermore, one can alter icon properties such as size, text font size, text position, remove text and increase whitespace. See *Alter Icon Representation* (chapter 7.4.8 – page 73) for more information.

### 7.5.2 Advanced Adaptation of Existing Components

Relevant categories: *Form based*

By this we mean work that requires some form adaptation or programming beyond what the available component properties permit. Typically, in an object oriented programming environment, this would be to subclass exiting component and adding desired appearance and/or functionality.

This might initially be more labor-intensive, but once in place, and if done properly and maybe stored in some form of component repository, the components can be reused in other situations. The possibility of further specializing components is present.

Advanced adaptation of existing components is something which is typically done by a **programming approach**. This normally provides access to the finer details of how a user interface and its components are coded and constructed. For instance, in an object oriented development environment, one can inherit from a component and use this as a base for fine-tuning how you adapted component should look and behave.

### 7.5.3 Developing Custom Components

Relevant categories: *Form and Graphic based*

The most extreme way of adapting a user interface to finger optimized interaction is by developing custom components from scratch. The initial work, with creating these components, can be quite time-consuming and require a special insight into both what makes a usable control and how to implement it.

Using specially tailored components can be quite confusing for the user, if not build around or on well-known and accepted conceptual models or interface metaphors within the current interaction paradigms. This is due to the fact that if the component greatly differs from existing solutions, the user might not know how to operate or use the component. Hereby it can be necessary to inform the user of how a new or unrecognizable component or control is to be operated. This is often done in games by having a beginners-tutorial.

As commented before it is common to develop custom components for graphic based user interfaces, something which requires additional programming. Whether or not it is feasible to also develop components adapted for finger based interaction must be weigh up against the need for it and the additional cost.

Creating a custom component is something which normally is done by a **programming approach**. This normally provides access to the finer details of how a user interface and its components are coded and constructed.

### 7.5.4 Change Scrolling Mechanism

Relevant categories: *Document, Icon, Graphic and Repetition based*

Many user interface categories regularly exceed the available display area, something which gives rise to the need for functionality that facilitates selection of which part that should be shown. And as commented earlier, it is difficult to hand a normal scroll bar with a fingertip due to the small size of the scroll bar and inaccuracy of the fingertip.

Panning is relatively “finger friendly” as it requires little accuracy. A problem with panning is how to enable editing and interaction with the user interface, because if interaction with the user interface will trigger panning functionality. This problem is discussed further in *Panning* (chapter 7.4.9 – page 74).

Another way of facilitating scroll-like functionality, which to some extent allows both scrolling and interaction with the document itself, is the concept of borders facilitating scrolling. See *Borders Facilitate Scrolling* (chapter 7.4.10 – page 75) for more information.

### 7.5.5 Manipulate Content

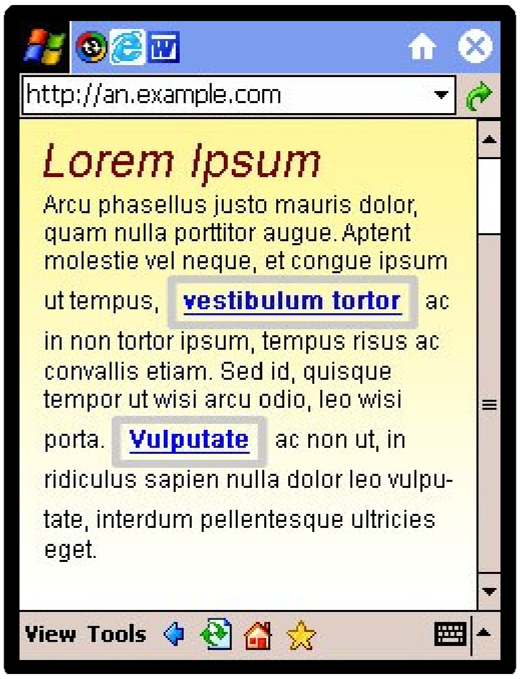
Relevant categories: *Document based*

There are not that many component in a document based user interface that can be adjusted to become more suitable for finger based interaction. There are usually three types of components, the main panel showing the document, a tool bar and a pull-down menu. What can be done with the document will be commented here. For the latter two see guidelines for *Simple Adaptation of Existing Components* (chapter 7.5.1 – page 84).

In document based applications there is often situations where it is natural to interact with the opened document, e.g. edit text in a text document or click on links in a web document.

In the case of wanting to edit a text document this is quite difficult because of the lack in precision with finger based interaction and the size of the components (mainly text) in the document. It can be hard to select a specific character, word or sentence or even select were to new write text. This can be somewhat resolved by manipulating the document (as purposed in *Manipulate Document Layout and Content* – chapter 7.4.14 – page 81) with the intent of increasing "finger friendliness". Normally it will be things like increasing font size, other elements and line spacing.

In the case of interaction with links in a web document the above mentioned actions can be performed. But another solution, that might not alter or compromise the appearance too much, can be to interpret the web document, detect objects which can be interacted with (usually links) and then emphasize these instances. These objects can be emphasized and rendered more "finger friendly" by for example making the text bold, increasing font size, drawing a (fat and/or colorful) border around, increasing whitespace before and after or a combination of some or all of these.



Lorem Ipsum

Arcu phasellus justo mauris dolor, quam nulla porttitor augue. Aptent molestie vel neque, et congue ipsum ut tempus, vestibulum tortor ac in non tortor ipsum, tempus risus ac convallis etiam. Sed id, quisque tempor ut wisi arcu odio, leo wisi porta. Vulputate ac non ut, in ridiculus sapien nulla dolor leo vulpu-tate, interdum pellentesque ultricies eget.

*Illustration 7.8 - How links in a web document can be adjusted to make the easier to hit with a fingertip.*

This concept of interpreting and emphasizing can be quite effective without compromising too much of the visual layout and appearance. But it requires that an interpreter is implemented into the application, something which will require work beyond just designing the user interface.

## 7.6 Switching Between Stylus and Finger Optimized User Interface

### 7.6.1 Have Two Versions of the User Interface

Relevant categories: *Form, Graphic and Repetition based*

Have one user interface instance with a “normal” user interface aimed at styles based interaction. The second instance can be design guided by for instance *Simple Adaptation of Existing Components* (chapter 7.5.1 – page 84), *Advanced Adaptation of Existing Components* (chapter 7.5.1 – page 86) and Developing Custom Components (chapter 7.5.3 – page 87) hereby achieving a more “finger friendly” user interface.

See *Have Two Versions of the User Interface* (chapter 7.2.1 – page 62) for guidelines and more information.

### 7.6.2 Facilitate Switching Between Projections

Relevant categories: *Icon and Document based*

This is very similar the solution of *Have Two Versions of the User Interface* (chapter 7.6.1 – page 90). It is also something which is facilitated by most icon based user interfaces, the possibility switching between *large icons*, *icon list* and *detailed list* projections. Several word editors also support this – e.g. Pocket Word has four different projections to choose from, called *Writing*, *Drawing*, *Typing* and *Recording*. The idea is simply to have two or more projections where each of them is aimed at handling various challenges. Here the solution is to have one projection for stylus based interaction and another for finger based. The guidelines given for *Finger Optimized User Interface* (chapter 7.5 – page 84) presents some factors that should be taken into account when designing a “finger friendly” user interface projection.

The switching between the projections is normally done via the main pull-down menu, something which is alright when using a stylus, but not well-suited for finger based interaction. So when wanting to switch to a “finger friendly” projection is vital that this can be by using a fingertip. There can for instance be facilitated by a large button or large check box. The con with having a visible control is that this control will occupy screen space.

### 7.6.3 Have a Set of Mapping Rules

Relevant categories: *Form, Icon and Graphic based*

Using a set of mapping rules is something which stems from the world of model based development, and here the mappings, defining the transformations between different components, is at a horizontal level (mapping between different components within the same abstraction level) and not vertical (from e.g. a higher abstraction lever to a lower). This solution is quite similar to *Change Between Alternative* Controls (chapter 7.7.2 - page 92), where the goals is to have the user interface adapt to varying equipment with different screen size. What separates these guidelines form the above mentioned, is that here the objective is to make the user interface facilitate switching between stylus and finger based interaction.

The solution is to have, on one side, a set of components fit for stylus based interaction, and on the other side, a set of components which are fit for finger based interaction. One must then set up mapping rules between these components, defining which "stylus friendly" component that can be transformed into equivalent "finger friendly" component. It is vital that the mapping rules do not lose information or fundamental functionality when it is transformed. It can therefore in some instances be necessary to transform one component into two or more on the other side.

It can be a fair-sized task initially to get these mapping rules right, but once in place they can handle the change between a stylus and a finger optimized user interface quite elegantly. The mapping rules can most likely also be reused in other situations where the need for the ability to switch between a stylus and finger based interaction is present. Because of the relatively large work-load, creating mapping rules to handle this is something which normally will not be viable for just one instance, but rather when several applications or software solutions demands it.

## 7.7 User Interface Able To Run on Equipment with Different Screen Size

### 7.7.1 Create Several Versions of the User Interface

Relevant categories: *Form and Graphic based*

This builds on the same concept as presented in *Have Two Versions of the User Interface* (chapter 7.2.1 – page 62). But here it is vital to first get an overview of which mobile devices with their appurtenant screen resolutions the application is meant to support. Hereafter a separate user interface is designed specifically for each of the screen resolution variations.

As with *Have Two Versions of the User Interface* (chapter 7.2.1 – page 62) an advantage with this solution is that the designer has the possibility of specifically tailoring a version of the application user interface to each screen variation. But the con, of having the added development work when constructing many separate user interface versions for one single, use can be quite time consuming if there are many different screen resolution. It can also be a challenge to maintain these user interface instances consistent with each other so that the usability is kept throughout.

To minimize the work here, it can be smart to use a tool that supports **drawing the user interface**. Here the designer can focus on adapting each user interface instance to the varying screen sizes and not the underlying code. Hereby the developer can handle more complex tasks and be more efficient – i.e. create faster and more complex user interfaces and avoiding usability hindrances. It is also natural to create and work on something graphical by visually drawing it, as opposed to for instance using a textual tool.

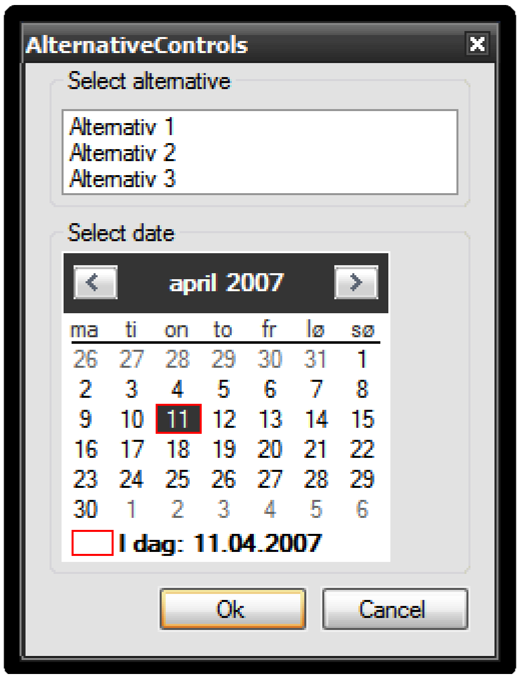
### 7.7.2 Change Between Alternative Controls

Relevant categories: *Form based*

This alternative is in some ways similar to *Struts and Straps* (presented in chapter 7.4.1  - page 69), but instead of having a set of constraints or rules which manage how controls should adapt (size wise, whitespace between, etc.) when the user interface environment changes, the rules define which controls that can be replaced with which alternative, equivalent controls.

There are two ways of attacking this approach, one is only using components which are available in your current development environment, for example controls provided by the .NET Compact Framework. Another is to develop custom controls, either based on existing or from scratch. The latter can be much more complex and labor intensive (as a starting point) but provides the user interface designer with the ability to custom tailor the controls in the user interface to the exact application domain.

As an example for using already existing controls, we may have a user interface that consists of two main components, one, a list box with alternatives (here showing three alternatives simultaneously) and, two, a calendar control used for picking a date. The purpose and functionality of this example application is irrelevant.



| ma | ti | on | to | fr | lø | sø |
|---|---|---|---|---|---|---|
| 26 | 27 | 28 | 29 | 30 | 31 | 1 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 30 | 1 | 2 | 3 | 4 | 5 | 6 |

I dag: 11.04.2007

*Illustration 7.9 - Example of how a user interface could look before controls are replaced to minimize space use.*

Now, the space being used can be minimized a great deal and the same functionality can be achieved by, in this instance, respectively replacing then list box with a combo box and the full calendar with a simple date chooser. The space now utilized is roughly one half of that of the original.

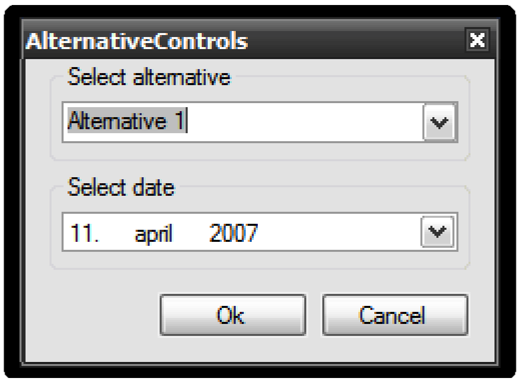


*Illustration 7.10 - Example of how the user interface from Illustration 7.9 can look when controls have been replaced in order to reduce area usage.*

Which controls that can be replaced by others depends on the functionality and use, both for the user interface as a whole, but maybe more importantly for each separate control. When opting for a replacement strategy, it is important to consider thoroughly which controls to replace and with what. If the changes are many and alter the appearance significantly, the user can become disorientated and lose track of the task at hand, and thereby the usability can be greatly reduced. To avoid this confusion a rule of thumb is to keep the same labels, which are appurtenant to each control, when doing necessary replacement.

Examples of controls that can be replaced by other are: *list box* → *combo box*, *multi lined text box* → *single lined text field*, *check list box* → horizontally arranged *check boxes* and *month calendar* → *date time picker*.

This is an approach which a **model based approach** would be suited for. The main work would be in finding an appropriate user interface model to design in and thereafter defining the mapping between components telling how a user interface model can be transformed to the various screen sizes. Ideally a tool provides and supports all this, and you as a user interface designer can focus on modeling the user interface. Thereafter it should just be a case of hitting a button and the transformation the user interface model to concrete implementations adapted to the varying screen sizes.

### 7.7.3 USE A MODEL BASED APPROACH

Relevant categories: *Form, Icon, Document, Graphic and Repetition based*

Having the possibility of designing one user interface model, and then automatically – provided that the tool support, models and transformations are adequate – transform this to various context and devices, is one of the key advantages of the model based approach. It is also this advantage which makes it a fitting way of resolving the challenge of having a user interface which is able to run on equipment with different screen size.

The model based approach provides the user interface designer with the possibility of focusing on the challenges of developing an efficient UI and not on the underlying code. The concept of working on a fairly high abstraction level without losing usability can most likely be accomplished though model based development. This high abstraction level makes it easier to handle more complex tasks, because it is possible to see and comprehend more at once. And this high abstraction level combined with a visual way of describing/programming is for many a more natural and efficient way of working.

With the ability to relatively fast model a concept or application domain, one can more easily detect potentially wrong perceptions, and the correction of these at the earliest possible stage means a more time- and cost-efficient development process.

When working with models, it is vital to have a suitable meta-model. One problem is if it has not covered the right concepts, which may be difficult enough, but another and maybe more pressing challenge is how to stay at the right abstraction level. If the meta-model elements should be too general, it will quickly become very difficult to model anything specific. Should, on the other hand, the elements be too detailed this will render it overly complex to model anything concrete, and the advantage over lower lever approaches disappears.

In an user interface development process it is often vital that available building blocks works the way they are suppose to – this in context of functionality and visual details (which often go hand in hand). This is also connected to the abstraction level which is discussed above. But a challenge maybe how to give the user interface designer the power to change and tweak visual details, e.g. for a company which wants their software to have a certain look and feel that corresponds to other products they might have.

# 8 CONCLUSIONS AND FUTURE RESEARCH

The main purpose of this master thesis has been to provide an **overview** of what is out there in terms of mobile, flexible user interfaces, and hereunder give **concrete guidelines and principles** of how one can create and sustain flexible user interfaces on mobile devices through varying contexts. Naturally there does not exist one, single and all-powerful solution which solves all problems, but rather a set of guidelines and principles that when applied on the right user interface type and for the right challenge will improve and maintain user interface usability. And it is often not just a case of how to avoid or handling certain problems, but maybe more taking advantage of and utilizing the potential and possibilities provided by, with and within mobile devices.

The term **mobile device** summarizes a very diverse and large group of computing devices. It consists of devices from the larger portable and tablet PCs, with the new UMPCs, through PDAs and down to the smaller Smartphones and other mobile phones. And the diversity between these – i.e. the varying screen sizes, facilitated input mechanisms, how they are used, etc. – together with their appurtenant operating systems (e.g. Linux, Palm OS, Windows Mobile, Symbian and Mac OS X) give many and varying possibilities and challenges in the context of designing user interfaces.

When examining **existing solutions** it was discovered that most of the solutions, for the various challenges, were of a quite simple and basic character – e.g. to preserve a flexible layout at run-time it was mostly either a case of adding or adjusting traditional scroll bars or doing nothing. Many of the applications were quite difficult to use with the fingertip and none of the application had any dedicated support for shifting between stylus and finger based interaction.

The context of this thesis is defined within space that emerges between **three axes**; user interface categories, user interface challenges and design approaches. This provides a relatively well-defined and easy-to-follow structure.

Under the first axis, **user interface categories**, five different categories were chosen; form, icon, document, graphic and repetition based, which then provides a base for categorizing both application user interfaces and guidelines. These categories aim to be abstracted above simple mechanisms and components and at the same time embrace application user interfaces independent of the wide range of application domains. The user interface categories are all within the graphical user interface paradigm.

When working with the five user interface categories, it quickly became apparent that some where more complex to handle than others. Complexity is here used in the context of the user interface plasticity – i.e. preserving usability despite a changing environment both internally and externally. The list below positions the categories by complexity, with the most complex first and the least complex at the end:

1. Form based
2. Graphic based
3. Repetition based
4. Document based
5. Icon based

As listed above, the most complex category was probably the form based one. The nature of this user interface type; with many components, with varying complexity and functionality, which need to interact with each other and the user on so many levels, can be very challenging to handle appropriately. The graphics based category is maybe the category with most internal variation between the user interface instances, some of which can be very complex to handle while others are relatively simple. At the other end of the scale – i.e. fairly simple to handle in most cases – we find the icon and document based categories. The repetition based user interface type can be placed somewhere in between, but for the most part more towards the simpler end of the scale.

The **user interface challenges**, and second axis, are a selection of the most common challenges which one might encounter on mobile devices. They are created with foundation in the three main problem areas: utilizing screen space, flexible layout at run-time and flexible layout at design-time. Hereunder six different challenges have been defined: switching between portrait and landscape screen orientation, software keyboard showing/hidden, content larger than its display container, finger optimized user interface, switching between stylus and finger optimized user interface and user interface able to run on equipment with different screen size. These challenges where used in both testing how existing applications handled them and, maybe more importantly, as sources for the guidelines.

The guidelines provided are diverse; some handle unique situations, some overlap with others (with reference to  e.g. principles, mechanisms and/or techniques) and some can even contradict each other – in other words, what is a good solution in one situation can in another make matters worse. This is because they handle varying challenges within different categories and in changing contexts.

Many of the challenges had a tendency, when it boiled down to the essentials, to become a challenge of handling content larger than its display container. Especially the challenges of switching between portrait and landscape orientation and software keyboard showing/hidden.

For the presented guidelines one can find some **main principles** which they in different ways conform to:

- **Facilitating scrolling functionality** in one form or another. One of the most promising here is the notion of *Borders Facilitate Scrolling* (chapter 7.4.10 – page 75). By using this, as purposed, it is possible to provide scrolling functionality, regulate scrolling speed (with graded edges) and at the same time not loose vital screen space. Another good solution is to use *Panning* (chapter 7.4.9 – page 74). This is intuitive for the user and does not occupy unnecessary screen space. It can also support varying scrolling speed by implementing support for a touch-drag-lift motion, where the speed desired for scrolling is computed based on the speed and acceleration with which the drag motion is performed.

- **Adjust or adapt components** which the user interface consists of. This is something which is recommended both when having to optimize how the screen space is being used and when needing to support finger based interaction. One good idea for optimizing screen usage is to follow the guidelines of *Struts and Straps* (chapter 7.4.1 – page 69). With this, the user interface can follow changes (at run-time) within certain limits, and once in place it can be reused for other user interface instances. For a more finger friendly" user interface, the notion of *Simple Adaptation of Existing Components* (chapter 7.5.1 – page 84) should be considered. This can be done quite easily, as it is simply a case of using altering already existing and changeable component properties (though it is worth noting that this also is its Achilles heel).

- **Group content differently** is a regularly followed principle, and is often used to optimize how the available screen area is used. This is something which is relevant mostly in form based user interfaces. One of the most common and usable techniques is to *Group Similar Components in Tab Folders* (chapter 7.4.3 – page 70). Its "brother", Partial Tab Folders (chapter 7.4.4 – page 71) should also be mentioned here. When used correctly, these solutions can make a complex and multifaceted user interface usable.

- **Do nothing, or lock the user interface,** is something that under certain circumstances can be used with success. It can often be better to keep a user interface the way it is and not conform to the changing contexts. This is particularly relevant for advanced graphic based user interface, where conforming or have to construct extra user interface will result in much extra work and therefore non-profitable solutions. *Lock User Interface in One Orientation* (chapter 7.2.3 – page 64) proposes just such a solution – i.e. have the user interface adapted to one orientation and ignore screen orientation changes.

The third axis, **design approaches**, is founded on the four main ways of designing and implementing a user interface. These are: programming, drawing, modeling and marking up the user interface. The approaches all have different pros and cons for different challenges and in various contexts, and are recommended for relevant challenges and within related guidelines.

The most well-established design approach is **programming** the user interface. Due to the fairly low abstraction level this provides the user interface designer with the power to control every part of the user interface (assuming the programming paradigm and libraries are well-known). But this control of details comes at a cost, and maintaining an overview can become very difficult if the programming assignment becomes large and complex.

**Drawing** the user interface is something which also has been available for some time now – e.g. Microsoft Visual Studio 97 facilitated drawing the user interface. As with other WYSIWYG (What You See Is What You Get) the user interface designer can focus on visually designing the user interface and not underlying code. A problem though can appear if the tool does not support exactly what you want to do, thereby forcing manually coding which in turn can make create problems, for instance with further usage of the "drawing tool" on the current instance.

An up and coming approach is the concept of **modeling** the user interface. It has been an established research area for more than ten years (Nilsson, 2004). But the modeling languages and tools have only to a limited extent made their ways out of research, something which is the modeling approach’s main drawback. A model based approach provides the user interface developer with the possibility to focus on the challenge of developing an efficient and usable user interface. When a user interface is modeled the process of creating the concrete user interface, or even multiple instance for several contexts, can be automated (provided that the mappings are supported). With the ability to relatively fast model a concept or application domain, one can more easily detect potentially wrong perceptions, and correction of this at the earliest possible stage means a more efficient development process. The model based approach is something which is almost custom made for the challenge *User Interface Able To Run on Equipment with Different Screen Size* (chapter 5.2.6 – page 34).

**Marking up** the user interface is something which grew in use parallel with the popularity of the Internet, where it still has its main use. The main advantage is that it is quite easy to use and structure a user interface and it is also available though any generic client. But mark-up languages (like HTML, XML, etc) are intended for semantically structuring and not visually designing content, this is something that should be done by using for example CSS (Cascading Style Sheets). Marking up a user interface is not commonly used in existing stand-alone applications.

Advantages and disadvantages of the different design approaches is something which should be investigated further. Here it would be natural to go deeper into each of the approaches and thereby map them more extensively. The three design approaches, programming, drawing and marking up, are fairly well-established, have a quite large user base and operating domain. The model based is in a different category. This approach, fully realized from A to Z, is an approach with potential to really affect how user interface designer work, the efficiency of how they work and the effectiveness of this work. A natural step would be to study the model based approach further.  What is state of the art in designing mobile user interface, which current modeling environments and tools are available, what are their pros and cons and what can be done to make these better?

Thereafter it would be interesting to see how the model based can be used, both in theory and in practice (with current tools), to handle and resolve the challenges and guidelines presented in this master thesis. Is the model based approach better suited for some type of challenges or for certain guidelines, and if so why is this? This would also reveal situations and/or challenges which are not ideal for a model based approach and which approaches that are appropriate, something that could highlight weaknesses with the model based approach and thereby what to improve.

And as the world of mobile computing continues to progress and grow – i.a. new mobile devices, screen variations, input mechanisms, operating systems, development environments, design approaches, etc – both new challenges and solutions will see the light of day. At present time it is natural to draw attention to one of the most talked about, “soon to come” mobile device at present day, the Apple iPhone, which is scheduled to be released by the summer of 2007. This is said to, with its Mac OS X, improve how mobile user interface handle, look and feel. It would for example be interesting to examining what this device does differently in terms of usability; which new mechanisms it uses, if they increase usability and why they do or do not work better.

# 9 REFERENCES

**The Research Council of Norway** ICT - Core Competence and Growth (VERDIKT) [Online] // The Research Council of Norway. - Integrate AS, 2005. - April 23, 2007. - http://www.forskningsradet.no/servlet/Satellite?cid=1092128011679&pagename=verdikt%2FPage%2FHovedSideEng.
**Apple Inc.** Introducing iPhone [Online] // Apple. - Apple Inc., 2007. - April 18, 2007. - http://www.apple.com/iphone/.
**Calvary Gaëlle [et al.]** Plasticity of User Interfaces: A Revised Reference Framework [Conference] // 1st International Work shop on Task Models and Diagrams for User Interface Design TAMODIA. - Bucharest : INFOREC Printing, 2002. - pp. 127–134.
**Google, Inc.** Google Earth [Online] // Google Earth. - Google, Inc., 2007. - April 14, 2007. - http://earth.google.com/.
**Google, Inc.** Google Maps [Online] // Google Maps. - Google, Inc., 2007. - April 14, 2007. - http://www.google.com/maps.
**Grudin Jonathan** Utility and Usability: Research issues and development contexts. [Journal]. - [s.l.] : Interacting with computers, 1992. - 4(2):209-217.
**Herstad J., Thanh, D. v., & Audestad, J. A.** Mobile communication and interaction in context [Conference] // International conference on intelligent user interfaces. - New York : ACM Press, 1998. - p. 198.
**Kiljander Harri** Evolution and Usability of Mobile Phone Interaction Styles [Book]. - Helsinki : Helsinki University of Technology, 2004. - ISBN 9512273209.
**Lindholm Christian, Keinonen Turkka and Kiljander Harri** Mobile Usability: How Nokia Changed the Face of the Mobile Phone [Book]. - New York : The McGraw-Hill Companies, Inc., 2003. - ISBN 0-07-138514-2.
**MapQuest, Inc.** MapQuest [Online] // MapQuest. - America Online, Inc., 2007. - April 14, 2007. - http://www.mapquest.com.
**Microsoft Corporation** Layout Meta Tag [Online] // Microsoft Developer Network. - September 14, 2006. - March 2007, 20. - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/mobilesdk5/html/wce51conLayoutMetaTag.asp.
**NASA** World Wind [Online] // World Wind. - NASA, 2007. - April 14, 2007. - http://worldwind.arc.nasa.gov/.
**Nielsen Jakob** Usability Engineering [Book]. - Fremont, California : Morgan Kaufmann, 1994. - ISBN 0-12-518406-9.
**Nilsson Erik G. [et al.]** Model-based user interface adaptation [Conference] / ed. Davies Nigel, Kirste Thomas and Schumann Heidrun. - Schloss Dagstuhl, Germany : Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), 2005. - Seminar: 05181 - Mobile Computing and Ambient Intelligence: The Challenge of Multimedia. - ISSN 1862-4405.
**Nilsson Erik G.** Design guidelines for mobile applications [Report] : SINTEF Report STF90 A06003. - Oslo : SINTEF Telecom and Informatics, 2005. - ISBN 82-14-03820-0.
**Nilsson Erik G.** Design patterns for user interface for mobile applications [Report]. - Oslo : SINTEF Telecom and Informatics, 2005.
**Nilsson Erik G.** Modelling user interfaces – challenges, requirements and solutions [Report]. - Oslo : SINTEF Telecom and Informatics, 2004.
**Preece Jenny, Rogers Yvonne and Sharp Helen** Interaction Design, Beyond human-computer interaction [Book Section] // Interaction Design, Beyond human-

computer interaction. - New York : John Wiley & Sons, Inc., 2002. - ISBN 0-471-49278-7.
**S60** S60 Phones [Online] // S60. - Nokia Corporation, 2007. - April 30, 2007. - http://www.s60.com/life/s60phones/browseDevices.do.
**Satyanarayanan M.** Fundamental Challenges in Mobile Computing [Conference] // Annual ACM Symposium on Principles of Distributed Computing. - Philadelphia : ACM Press, 1996. - pp. 1-7.
**SINTEF** FLAMINCO – FLexible Applications exploiting Multi modal INteraction and COntext [Online] // SINTEF. - March 20, 2007. - March 27, 2007. - http://www.sintef.no/content/page1____13988.aspx?epslanguage=EN.
**Sun Microsystems, Inc.** Trail: Creating a GUI with JFC/Swing: Table of Contents [Online] // The Java(tm) Tutorials. - Sun Microsystems, Inc., 2006. - April 10, 2007. - http://java.sun.com/docs/books/tutorial/uiswing/TOC.html.
**Symbian-Guru** New Freeware, RotateMe, Portrait or Landscape on S60v3 [Online] // Symbian-Guru. - February 7, 2007. - March 14, 2007. - http://symbianguru.typepad.com/welcome/2007/02/new_freeware_ro.html.
**Szekely Pedro** Retrospective and Challenges for Model-Based Interface Development [Conference] // CADUI '96. - Namur, Belgium : [s.n.], 1996.
**Thevenin D and Coutaz J** Plasticity of User Interfaces: [Conference] // IFIP TC 13 Int. Conf. on Human-Computer Interaction. - Edinburgh : IOS Press, 1999. - p. 1.
**Wikimedia Foundation, Inc.** Drag and drop [Online] // Wikipedia. - Wikimedia Foundation, Inc., April 3, 2007. - April 14, 2007. - http://en.wikipedia.org/wiki/Drag_and_drop.
**Wikimedia Foundation, Inc.** iPhone [Online] // Wikipedia. - March 14, 2007. - March 14, 2007. - http://en.wikipedia.org/wiki/IPhone#_note-MP3_Newswire.
**Wikimedia Foundation, Inc.** List of Palm OS devices [Online] // Wikipedia. - February 23, 2007. - March 9, 2007. - http://en.wikipedia.org/wiki/List_of_PalmOS_devices.
**Wikimedia Foundation, Inc.** Palm OS [Online] // Wikipedia. - March 7, 2007. - March 9, 2007. - http://en.wikipedia.org/wiki/Palm_OS.
**Wikimedia Foundation, Inc.** Symbian [Online] // Wikipedia. - March 4, 2007. - March 12, 2007. - http://en.wikipedia.org/wiki/Symbian.
**Wikimedia Foundation, Inc.** Ultra-Mobile PC [Online] // Wikipedia. - Wikimedia Foundation, Inc., April 17, 2007. - April 27, 2007. - http://en.wikipedia.org/wiki/Ultra-Mobile_PC.
**Wikimedia Foundation, Inc.** Windows Mobile [Online] // Wikipedia. - March 8, 2007. - March 12, 2007. - http://en.wikipedia.org/wiki/Windows_mobile.
**Zhai Shumin, Barton Smith A. and Selker Ted** Improving Browsing Performance: A study of four input devices for scrolling and pointing tasks [Conference] // INTERACT97: The sixth IFIP Conference on Human-Computer Interaction, Sydney, Australia. - San Jose, California : [s.n.], 1997. - pp. 286-292.
**Ziff Davis Publishing Holdings Inc.** 3rd way: Sharp pits Linux+Java against Palm/PocketPC [Online] // Linux Devices. - November 5, 2001. - March 9, 2007. - http://www.linuxdevices.com/news/NS3569993078.html.
**Ziff Davis Publishing Holdings Inc.** The Linux PDA Showcase [Online] // Linux Devices. - December 22, 2006. - March 9, 2007. - http://www.linuxdevices.com/articles/AT8728350077.html.