A Window in your Pocket

Some Small Patterns for User Interfaces

Introduction

Pocket computers (smartphones, palmtops, sub-sub-notebooks, organisers and communicators) are becoming more common, and are real alternatives to PCs for many purposes. Unfortunately, because they are physically smaller, user interfaces for pocket computers are much harder to design than those for larger machines. This paper presents some patterns for designing user interfaces for pocket computers. Programmers and designers can use these patterns to design interfaces that will better suit small computers, resulting in applications that are more usable.

Overview

In this paper we describe a collection of patterns for designing user interfaces for pocket computers. The techniques described by these patterns are not new; rather they are already embedded in the design of successful and usable pocket computers. Perhaps adherence to these patterns is one reason why some designs are more successful than others.

The Patterns

Please, to use an immortal phrase, "see Figure 1" for the structure of the patterns in this paper. The lines in the figure link related patterns.

First, the **ONE TRUE WINDOW** pattern generates the large-scale structure of your application. Then, the **HIDE AND SEEK** and **DIAL H FOR HELP** patterns describe how to design common facilities across your **ONE TRUE WINDOW**. **SCROLLING**, **CATEGORIES**, and **CHAIN OF DIALOGS** all describe how you can present more information to users than will fit into the **ONE TRUE WINDOW**. **BIG THUMB** and **A PICTURE IS SMALLER THAN 1000 WORDS** describe how you should design controls that users can interact with, **CUP OF TEA TEST** how you can ensure than when users go for a cup of tea or climb onto a bus they will not advertently destroy their data, and **USER CUSTOMISATION** describes how you can customize your users so that you get maximum sales for your product.

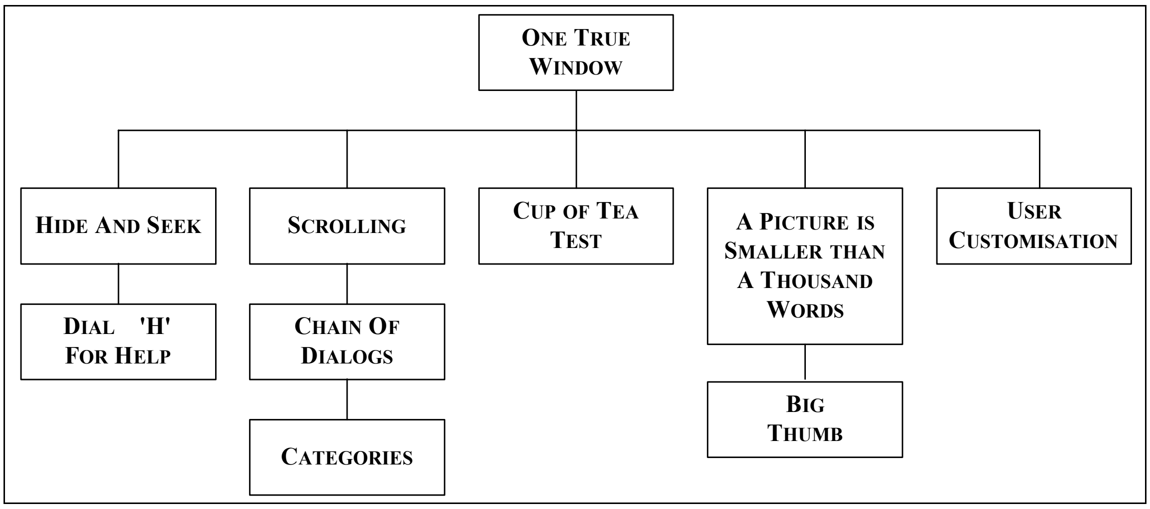


Figure 1: Pattern Structure

The art and science of designing user interfaces to small devices is a fast-developing field. There is no canon of best practice; UI designers are still learning what works best for users of small machines – and at least one recommendation has changed in the two years since we wrote the first version of this paper[1]. As new kinds of devices are produced, so the rules and patterns needed will change. Although there is some structure to this pattern collection, we don't claim that they form a complete pattern language for designing the interface to such machines. Instead it may be that some of these patterns may one day form a core of such a language.

Forces

The patterns in this paper resolve a number of forces. On the side of the Jedi, we are interested in maximizing the *ease of use,* that is, the *usability* of the programs we will write for our pocket computers. As is well known from many years of research into human-computer interaction [Kraftwerk], this is based on the *consistency* of the interfaces; the *visibility* of commands and data so users can see what they are doing and what they are able to do; the *speed* (or efficiency) of the interface, how *intuitive* or *easy to learn* it is, whether the interface is *modal* (restricting what users can do at any given time) and so on.

On the dark side, we have technical constraints, principally the *size of the display screen* (also known as the screen's *real estate)*: one of the main distinctions between a pocket computer and a desktop computer is that the pocket computer has a much smaller screen, and much less real estate. Many of these patterns are about balancing all the other forces off against this one.

These two sets of forces constrain the *solution,* but there are also forces that derive from the nature of the *problem.* These forces include the *amount of information* users needs to access; and the number of distinctly different *tasks* users need to perform with the system (basically the *number of applications* the system will support).

The Pattern Form

The patterns are written in our favourite pattern form, and spelt in English (not American); we hope that you honour our behaviour. As in Christopher Alexander's pattern format [Alexander], each pattern highlights and discusses a problem, then concludes: '**Therefore'.** This is followed by a specific recommendation and a discussion of how to implement this recommendation. Further sections then describe consequences and known uses.

Throughout the pattern descriptions we refer to other patterns using **PATTERN NAMES IN BOLD.** Perhaps the most vital discussion for a pattern is of the *forces*, or considerations and consequences to help readers to decide when to use that pattern rather than another. We've highlighted the main forces in the problem selection by printing them in *italics* (we also use italics for emphasis). Then in the Consequences section, we identify the pattern's other implications, positive and negative, separated by the bold phrase '**However**:' .

The principles we describe here are so familiar that it's unnecessary to provide a special section spelling out a list of 'known uses'; in most of the patterns such examples are listed in the text as we discuss why they are implemented in a specific way.

Most of these principles apply equally in some measure to all GUI design, rather than just very small devices. So to avoid the need to illustrate all the less familiar known uses with pictures, we've taken some of our examples from familiar MS Windows applications.

We've illustrated the patterns with screenshots from two of the most widely found mobile environments: Palm OS, which is probably the best known PDA environment and is now found on a few phones; and the Nokia Series 60, based on Symbian OS, which is currently the most widely found smartphone environment, used in many phone models by various manufacturers (Nokia, Siemens, Samsung, Sendo, etc.). We've also included a few illustrations from UIQ, another Symbian OS environment currently used mainly by SonyEricsson.

[1] The 'Context Window', which requires two equal windows on the screen, is now rarely if ever used in small UI designs.

One True Window

How many applications should you have on screen at once?

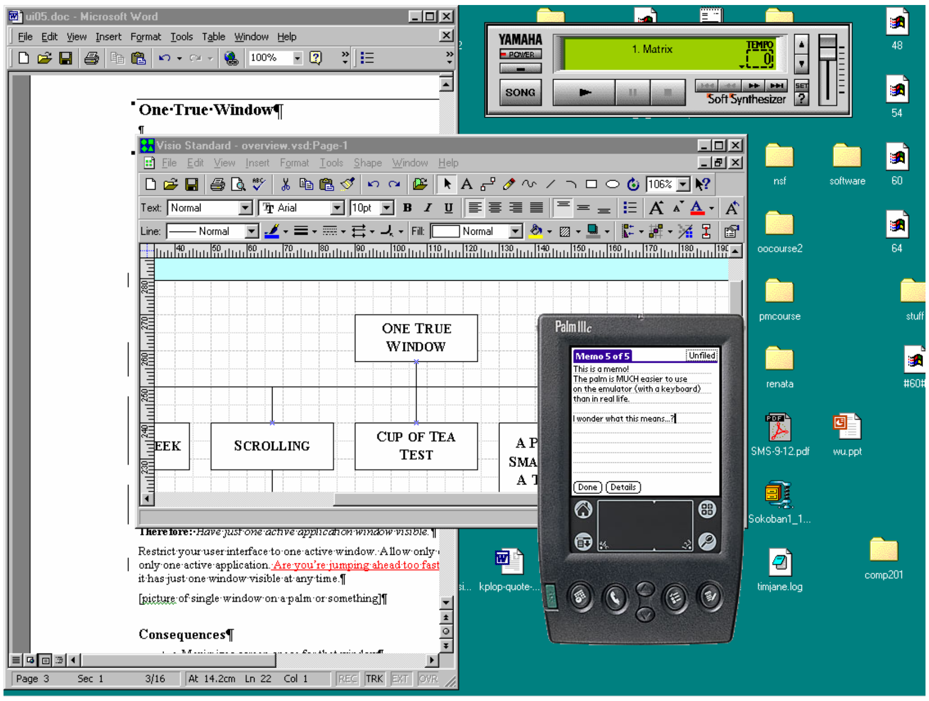
Multiple windows are a key advantage of desktop windowing software, coming from the research work started at Xerox PARC [Tesler]. Multiple windows have many advantages:

- Users can perform several task at one time, because several applications appear to be active simultaneously
- Interaction is modeless because users can choose a different window at any time
- Multiple windows can belong to single or multiple applications
- Multiple windows support lots of desk accessories (clocks, calculators, puzzles) — each in their own window
- Multiple windows leave space for a desktop background that can support access to files and additional menus.

They also have several important disadvantages, especially for systems with little real estate:

- Multiple windows require lots of screen space for window decorations that are used to manage the windows (especially old Macs & Windows)
- Since users can resize the multiple windows, you have to design window layouts that will work in many different sizes.
- The user becomes overloaded with information. Millers' rule [Miller] suggests that a typical human can handle only around seven items of information at a time.

On a large screen the advantages usually outweigh the disadvantages, although even relatively large screens can become cluttered with many applications, and lots of space is occupied by toolboxes and toolbars for many applications:


One·True·Window¶

Therefore:·Have·just·one·active·application·window·visible.¶

Restrict·your·user·interface·to·one·active·window.·Allow·only·
only·one·active·application.·Are·you're·jumping·ahead·too·fast
it·has·just·one·window·visible·at·any·time.¶

[picture·of·single·window·on·a·palm·or·something]¶

Consequences¶

Memo 5 of 5 Unfiled

This is a memo!
The palm is MUCH easier to use on the emulator (with a keyboard) than in real life.

I wonder what this means...?

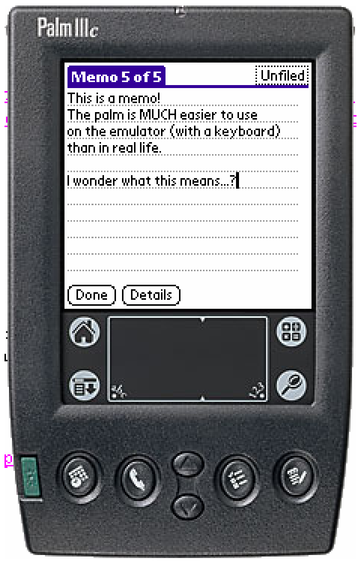
Figure 2: Many Windows on a Screen

On a small screen, this problem would be much worse!

Therefore: *Have just one active application window visible.*

Restrict your user interface to one active window. A design with one window is simpler: it has just one window visible at any time. This means that users will be able to interact with only one application at one time (unless you can split that window across multiple applications). Furthermore, because you only have one window, users have no need to resize it, so your design is simplified (one size fits all) and you can dispense with the controls needed to resize and manage multiple windows.

Here's a single memo pad application running on a Palm Pilot, and a Calendar (diary) application on a Nokia Series 60 (Symbian OS) phone.



This is a memo!
The palm is MUCH easier to use
on the emulator (with a keyboard)
than in real life.

I wonder what this means...?

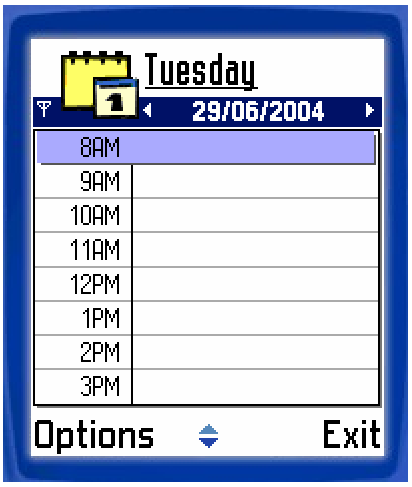


Figure 3: Palm Pilot Memo **Figure 4: Nokia Series 60 Calendar**

Virtually all significant pocket device UIs implement the 'One True Window':

- Microsoft Pocket PC and Smartphone environments.
- All the Symbian OS UI designs: The keyboard-only Series 60, the Palm-like UIQ, the Nokia Series 80 and the Nokia Series 90.

Consequences

This approach maximizes the screen space for that window. It makes UI design and implementation easier, since the size of each window is fixed. There's no need to modify screen layouts to handle different window sizes. And the user doesn't have to deal with too much information at a time.

However: It makes it harder to use "desk accessories". The user needs to swap windows to change applications. And the system will have to provide other support for modelessness (functions available to all applications).

Implementation

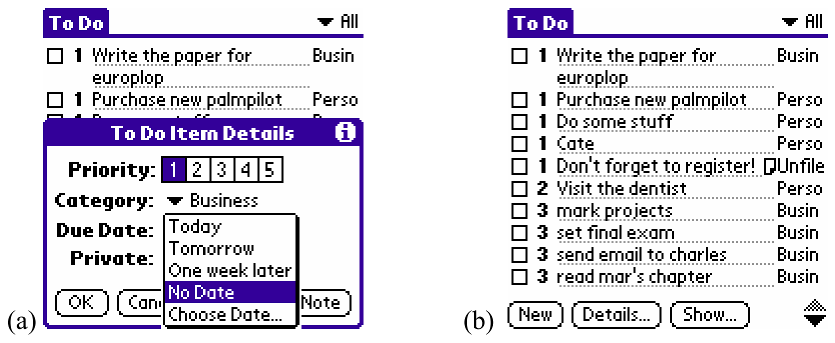
There are two classic UI implementation of this pattern depending on whether an invisible application has state.

Stateless Implementation

In the DOS, early Macintosh, PalmOs and Symbian Quartz environments, only one application has any active state at a given time – and this application's window is on the screen. The operating system provides 'application switcher' mechanisms to changes between the applications. This is sometimes done to save memory (see 'Application Switching', [Noble&Weir 2000]), but – as in the case of Symbian's UIQ – may also be simply to aid usability.

When the user moves to another application, any temporary user state (dialog box showing active, menus showing etc.) in the previous application is lost. For example, if you're editing a memo, and have the editing cursor in the middle of some text, then you switch to the Email application and back to the Memo pad, the cursor will have moved to a safe position at the end of the memo. Similarly, if you have a dialog box or menu popped up it will be closed when you return.

For example, Figure 5 shows a Palm with a menu popped up from a dialog (a). Switching to another application and back again will cancel the ‘To Do Item details’ dialog, leaving the original state (b)[2].


(a)

To Do ▾ All

☐ 1 Write the paper for europlop Busin
☐ 1 Purchase new palmpilot Perso

To Do Item Details

Priority: 1 2 3 4 5
Category: ▾ Business
Due Date:
Private:

Today
Tomorrow
One week later
No Date
Choose Date...

OK Can Note

(b)

To Do ▾ All

☐	1	Write the paper for europlop	Busin
☐	1	Purchase new palmpilot	Perso
☐	1	Do some stuff	Perso
☐	1	Cate	Perso
☐	1	Don't forget to register!	Unfile
☐	2	Visit the dentist	Perso
☐	3	mark projects	Busin
☐	3	set final exam	Busin
☐	3	send email to charles	Busin
☐	3	read mar's chapter	Busin

New Details... Show...

Figure 5: Result of a PalmOS Task Switch

Stateful Implementation

In Symbian’s Eikon environment (best known on the Psion Series 5), each application still shows only one single window, but each application still retains user state even when another application is running. The operating system provides ‘task switching’ mechanisms for the user to switch between them. If you did the same switch away from the equivalent To-Do application and back again, the dialogs would still be present.

Hide and Seek

How should you design menus, scrollbars and toolbars?

The application's menu bars, toolbars, and scrollbars represent commands that the user can perform, so these are very important parts of an applications’ interface design:

- These controls show what the application can do, so they need to be clearly visible (Visibility Principle).
- The Look & Feel of the controls should be similar across all applications (Consistency)
- Unfortunately, they take up lots of screen space

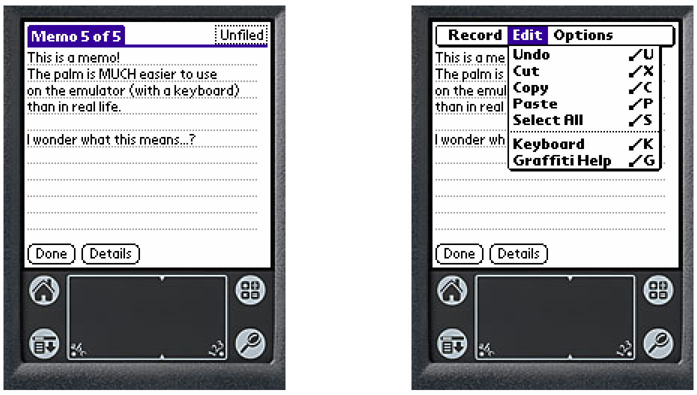
How can you keep the benefits of visibility and consistency, without requiring large amounts of precious screen space?

Therefore: *hide them, but have a standard way of bringing them back again*

The mechanism to bring back or toggle such items is part of the ‘Look and Feel’ of the application. So for an environment to be usable, it’s vital that it’s consistent across all applications.

Palm OS hides its menus, and provides a ‘menu’ button (in the recogniser area) that automatically displays the menu: menus can also be displayed by clicking on the window’s title bar.

[2] Note – strictly speaking the dialog shown here fails the ‘cup of tea test’, since it is not clear which To-do item is being edited.


Memo 5 of 5

This is a memo!
The palm is MUCH easier to use
on the emulator (with a keyboard)
than in real life.

I wonder what this means...?

Figure 6: PalmOS Menus

In Series 60, the menu is the main form of navigation, so the system reserves the two buttons by the screen for menu features. Thus in the Messages application, the Options button brings up the menu:

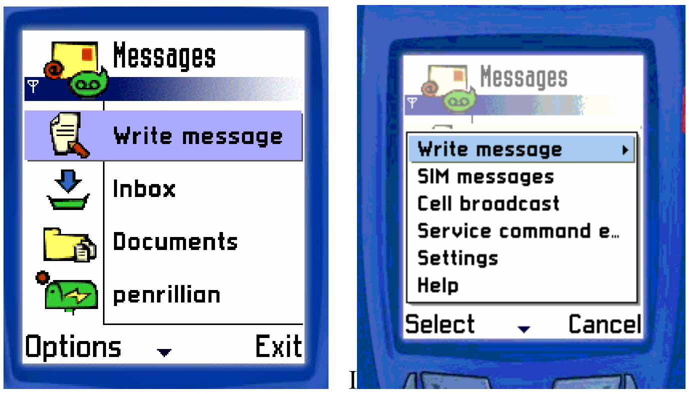


Figure 7: Series 60 Menu

Consequences

Hiding the controls saves screen space. And the standard 'reveal' mechanism maintains consistency between applications.

However: The approach is less intuitive for users, who need to know the incantation that makes the menu appear. It's less visible to users, as they cannot see at a glance what the options are. The menu is slower and more difficult to use, because it must first be made to appear before users can give menu commands. And once on display, the menus hide the underlying application data, making it less clear what the effect of each command may be.

Implementation

It is very important to users to follow the standard Look & Feel in arranging functionality between menus. For example in Series 60, the right hand menu button should always implement some kind of 'back' functionality (see **CHAINOFDIALOGS**). In Symbian OS UIQ and PalmOS the top right hand menu selects the current **CATEGORY.** There will be more sophisticated and subtle requirements – consult the Look & Feel guidelines for your current platform.

In general there is little problem with implementing this pattern in software – the application framework provides all the functionality and it would take more work to provide other mechanisms than to 'go with the flow'

Dial ‘H’ for Help

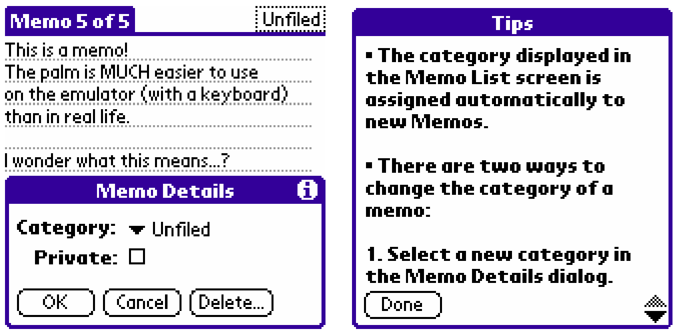
How should you provide help to the user without wasting lots of screen space?

- Help text, such as captions for dialog boxes that really explain what the options are, take up a lot of screen space
- Only inexperienced users really need these explanations
- But we must make it easy for inexperienced users to get tasks done.

Therefore: *keep the text on the screen to a minimum, but have a standard way to get detailed information about every item.*

The key to this pattern is that you must have standard ways of invoking the hidden help. Perhaps your **Hide and Seek** menu contains a “Help” menu item. Perhaps your dialog design always includes a help icon to give help on that dialog (again, as on the Palm).

In this Palm dialog below, the circled “i” in the top left corner will bring up a help screen. On the Nokia Series 60, the Help Screen is a standard menu item[3]:


Memo 5 of 5 Unfiled

This is a memo!
The palm is MUCH easier to use
on the emulator (with a keyboard)
than in real life.

I wonder what this means...?

Memo Details

Category: Unfiled
Private: ☐

OK Cancel Delete...

Tips

• The category displayed in the Memo List screen is assigned automatically to new Memos.

• There are two ways to change the category of a memo:

1. Select a new category in the Memo Details dialog.

Done

Figure 8: PalmOS Help

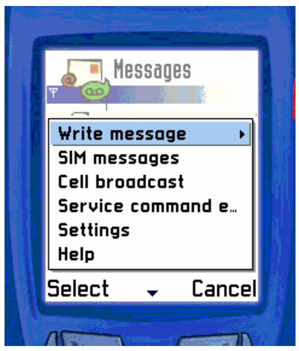


Figure 9: Series 60 Help Menu Item

Consequences

Popup help saves a lot of space on the screen. Expert users aren’t distracted by lots of additional information and can therefore work more effectively. All users has help available when they need it. And the help information is loaded only when required; it doesn’t take up valuable RAM memory when it’s not needed.

However: This can be harder to use than placing help directly onto the screen: users must first work out how to invoke the help mechanism. On the other hand, physical “help” keys can be *easier* to use. It requires operating system infrastructure to support this: typically a Help application and mechanisms to invoke it. It also requires help code in each application, with corresponding testing and support. And when supporting multiple environments, the developer needs to internationalise the help texts as well as the application’s main strings.

Implementation

There are two common forms of Help support, a separate Help Screen, and Help Tips:

Help Screen

The user command invokes a screen, separate from the main application. This screen explains the current application context, but typically may also allow the user to find out about, say, other parts of the application.

Such screens may use quite complicated text and graphics, and may even link to information available on the Internet.

[3] Except for the Nokia 7650, which doesn’t provide help.

Help Tips

An action by the user shows a short item of text related directly to a specific control or item. This may appear as a balloon, or a pop-up dialog, but it is strictly transient.

On some environments, notably MS Windows, the user invokes a help tip simply by holding the mouse over the relevant control. However this approach doesn't work on devices with a touch screen or keyboard.

Scrolling

What do you do when there's an unconstrained amount of data to display?

You can control the design of your application windows and dialogs. But the amount of data you have to display is out of your control: it belongs to users!

- Many applications manage a linear file of data (such as a 'document')
- But there's no simple way to separate it into single units of one screenfull.
- We don't want the screen size to constrain the maximum data size
- The user wants to access all of the data.

Therefore: *Use scrolling to make all of the data visible*

Have a concept of 'the current location' in the data. Display as much data as will fit on the screen starting from that location onwards. Allow the user to view or modify the data on the screen, and to move the current location. Typical movement commands are up or down a small amount, up or down a screenfull, and to the start and finish. Scrollbars also allow the user to select a relative position within the data.

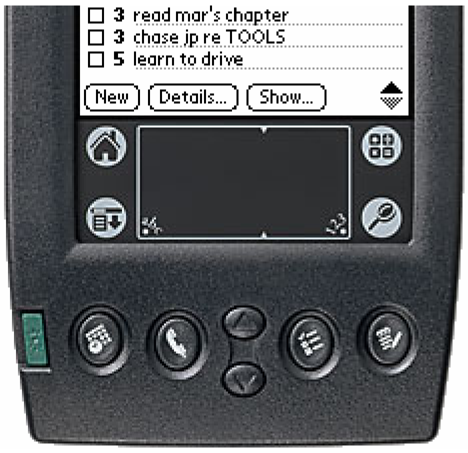


Figure 10: PalmOS Scrolling

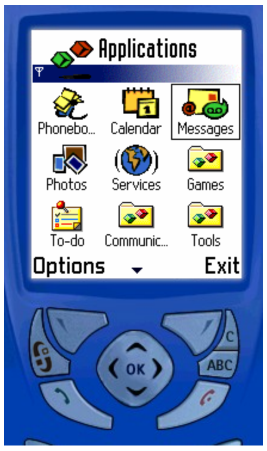


Figure 11: Series 60 Scrolling

These pictures of the bottom half of a Palm and of a Series 60 application picker both show the physical scroll buttons (in the bottom centre) and the small scroll button controls in the display (bottom right of screen on Palm; bottom middle on Series 60).

The main purpose of the small Palm controls is to show that this display *can actually* be scrolled; they are too small to be used easily in practice!

Consequences

The user can access all the data. The idiom is very common; anyone computer-literate will know it.

However: Managing the scrolling and scrolling displays can be complicated, and is best with support from the GUI environment. Scrolling through large amounts of data can be very slow, especially on a small screen or with a slow CPU. And most scroll controls are imprecise, making it difficult to scroll to any particular part of the document.

Scrolling is such an essential part of today's user interfaces that we tend to forget its importance – it was unheard-of in the early days of mainframe processing.

Virtually all word processors and text editors use scrolling. Eikon's 'Database' application makes its dialog boxes scrollable if the user specifies more fields in the database than will fit on a single screen. Most file manager applications use scrolling to handle the arbitrary number of files in a directory.

Implementation Notes

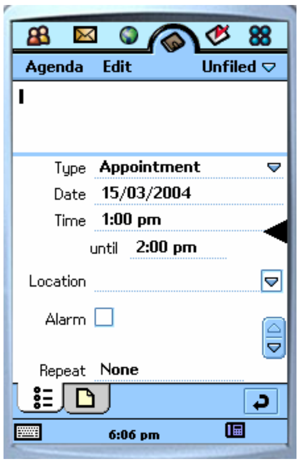
Mechanisms

There are several ways to control scrolling:

Scrollbars	These use up screen real-estate, but provide a visible feedback of the position in the scrolling range.
Hardware keys	Scrolling is such a key part of interfaces that almost all systems incorporate support for it into the hardware. Most keyboards have up, down, page up, page down, home and end keys. Almost all handheld devices now provide some kind of Up/down key (sometimes as part of a four-way key or joystick) – like the Palm Pilot illustrated above. A few, such as the SonyEricsson P800 and P900 and certain mice have a scrolling wheel, which makes easy to scroll a certain number of 'items'.
Grab&Drag	Here the user clicks or taps with a pointing device and drags, and the screen accordingly. This isn't a typical Microsoft idiom, so it's less familiar, but it is useful for viewing read-only data. For example, Adobe's PDF viewer for MS Windows provides this functionality.

What to Scroll

Some environments even allow programmer-specified dialogs to be scrollable. This works best when there's some easy mechanism leading the user from field to field. For example, in Symbian OS UIQ, the dialog boxes can be scrollable – see Figure 12, where the little control above the 'go back' arrow on the bottom right is the scrollbar; When you're entering data the keyboard supports 'next field' and 'previous field' buttons.



Type Appointment
Date 15/03/2004
Time 1:00 pm
until 2:00 pm
Location
Alarm
Repeat None

Figure 12: UIQ Dialog Scrolling

Some UI experts consider scrolling dialogs very undesirable, preferring alternatives such as **CHAIN OF DIALOGS** that offer more precise control over positioning.

Chain of Dialogs

How can you design a dialog that's too big to fit on one screen?

- Most applications at some time need to display a form of data with several related fields for the user to enter.
- The conventional GUI mechanism for this is a 'dialog box', or pop-up screen containing controls for each. The user enters new values, and may then activate them ('OK') or discard them ('Cancel'). Sophisticated dialog box implementations may validate the data in each field as the user enters it.
- Often there are more possible items to enter than can sensibly fit on a single screen, even if we cut out all extra text, menus and explanations using the patterns above.

Therefore: *split the dialog up into multiple windows, and allow the user to switch to each one.*

Divide the dialog into sets of related items, each set small enough to appear on a single screen. Allow the user to switch between them. Ensure the user always knows where she is in the chain of dialogs.

Consequences

You can create arbitrarily complicated dialogs. You can fit your dialogs into small screens. And you don't have to scroll.

However: It requires more programming effort, and more support from the GUI infrastructure than a simple dialog. The system must define a consistent Look & Feel for users navigating around dialogs. It's more complicated for the user to understand than a single-screen dialog. And UI design becomes more vital and more difficult, to ensure that the user can find the features they want to change quickly and easily.

Implementation

There are several approaches to implementing Chain of Dialogs:

'Tabbed' Dialogs

Here a dialog box has a series of different 'pages', or 'tabs'. Each page shows a different set of settings; however all the pages are logically part of the same dialog, and are invoked or cancelled with a single operation.

MS-Windows and Symbian UIQ applications use buttons along the top of the dialog (made to look like index tabs for files in a filing cabinet). Palm OS uses a popup menu button to select dialog panes (see below):

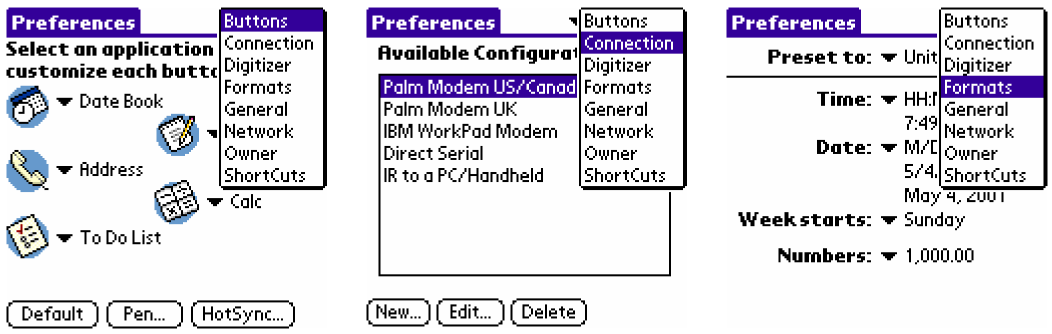


Figure 13: Palm OS 'Tabbed' Dialogs

Figure 13 shows three of the preference sub-dialogs on the Palm Pilot.

Nested Dialogs

Here the most important, or most commonly used, items appear as a main dialog; the remainder go into 'advanced features dialogs'. The main dialog has buttons on it that lead the user to each of the advanced dialogs; on quitting each advanced dialog, the screen returns to the main dialog. MS Windows applications use this approach, as do many Psion Series 5 applications. Note that 'activating' the advanced dialogs should not cause their settings to be activated until the main dialog is activated (some applications are inconsistent on this).

This is the chosen idiom for most Microsoft Windows, Pocket PC and Smartphone applications.

Sequence of Dialogs

Here the user is led from one dialog to the next: each dialog adds settings to the previous one, and the user has a clear Forward-Backward path.

This idiom is at the heart of the Nokia Series 60 Look & Feel. This has a keyboard-only API and a fairly small screen, with two menu buttons to provide navigation. Figure 14 shows the first three Dialogs on the Series 60 for configuring Internet access points.

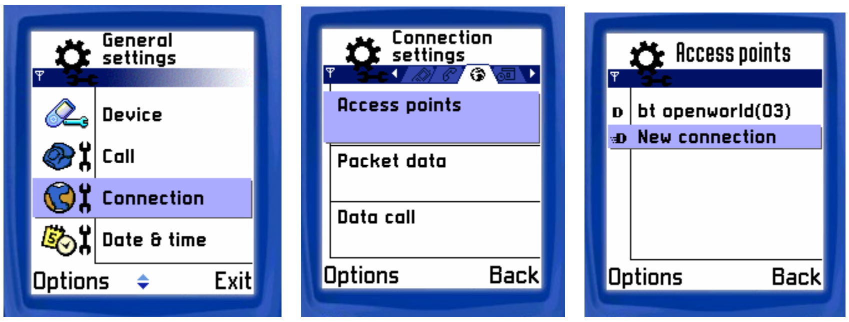


Figure 14: Series 60 Chain of Dialogs

It is also used in 'wizards' for setting up applications and systems in very many environments.

Categories

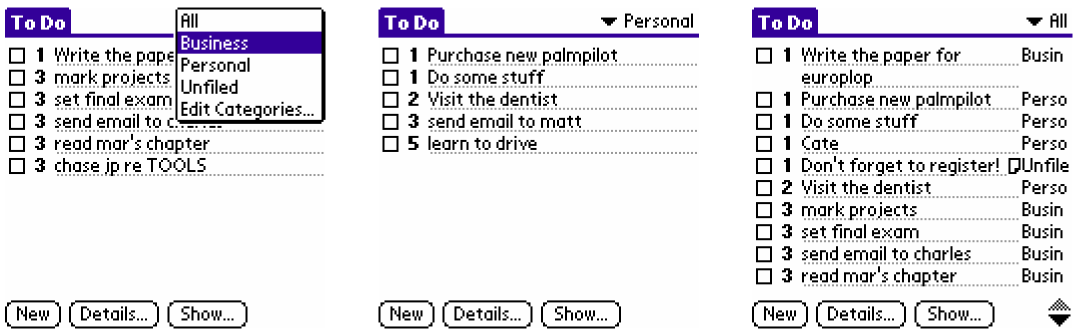
How can you organise the user's data so it's easy to find?

You can chop up your dialogs across multiple screens; and you can use scrolling to get around long linear documents. But how can you organise users' documents themselves, so that they are easy to find?

- The user has lots of data, arranged as separate named items.
- Users need to find individual items, but there's too much to fit in a single screen
- Users find hierarchical file systems difficult to manage and understand.
- File system browsers take up lots of screen space.

Therefore: *make each application provide single-level categories to organise its data*

PalmOS and the UIQ's SonyEricsson P800 both organise stuff (memos, addresses, to-do items) into categories (like one level directories) then uses a combo box to choose which category to display, one of which is "all". Most applications provide these categories.


To Do

☐ 1 Write the pape
☐ 3 mark projects
☐ 3 set final exam
☐ 3 send email to c
☐ 3 read mar's chapter
☐ 3 chase jp re TOOLS

All
Business
Personal
Unfiled
Edit Categories...

New | Details... | Show...

To Do ▾ Personal

☐ 1 Purchase new palmpilot
☐ 1 Do some stuff
☐ 2 Visit the dentist
☐ 3 send email to matt
☐ 5 learn to drive

New | Details... | Show...

To Do ▾ All

☐ 1	Write the paper for europlop	Busin
☐ 1	Purchase new palmpilot	Perso
☐ 1	Do some stuff	Perso
☐ 1	Cate	Perso
☐ 1	Don't forget to register!	Unfile
☐ 2	Visit the dentist	Perso
☐ 3	mark projects	Busin
☐ 3	set final exam	Busin
☐ 3	send email to charles	Busin
☐ 3	read mar's chapter	Busin

New | Details... | Show...

Figure 15: Palm Categories

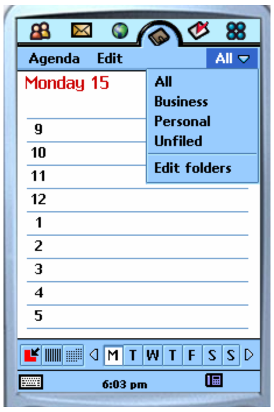


Figure 16: UIQ categories

Single level categories are easy to understand, and don't need lots of space or complex navigation to get around. The current category can be just one word on a popup menu.

Consequences

This pattern reduces screen real estate needed to display all items. It also gives reasonably fast access to items (when you know the category)

However: Categories make it hard to search for something when you don't know which category it is in. This approach is only easy to use with a maximum of about 7 categories. It requires more complexity than a flat structure, needing some way to create and delete categories, and to move items between categories. And the approach doesn't generalise easily to a hierarchical scheme.

Cup of Tea Test

What happens when the user goes away for a cup of tea?

Typically the users of pocket devices expect to be interrupted more in their tasks, since they can use their systems anywhere. They're not at a desk; they get off a bus and have to put the computer away!

- What happens if they were in the middle of a complex operation when they turned the machine off?
- Desktop applications have a lot of screen real estate devoted to showing the user context (title bar, toolbars, status bars etc).
- Pocket system cannot afford this real estate.

Therefore: *all screens must make the state of the system clear at all times.*

For any screen or dialog, imagine the user leaves it to go away for a cup of tea, and comes back having forgotten what she was doing. Would it still make sense? If not, add status information into the screen so that users will be clear what they were doing when they return from the interruption.

This is surprisingly hard to do in the constraints of a small screen, and it is credit to the UI designers of the most popular small screen systems that they achieve it most of the time.

Consequences

Users can be interrupted safely. And a user's data won't accidentally corrupt their data by completing some operation incorrectly.

However: It's hard to get right. The extra context information will be unnecessary most of the time. And temporary information may be lost (but users had probably forgot about it anyway)

Implementation

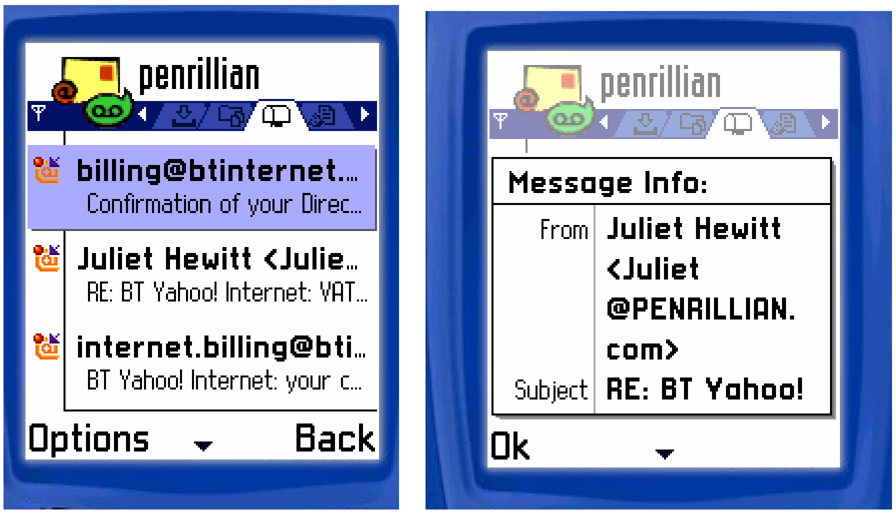
Showing Context

The main mechanism for showing the current operation the user is doing is the Title Bar at the top of the screen of the dialog.

For example, in Nokia Series 60 devices, the top eighth of the screen is taken up with a title bar; this shows the user the current application, or the current dialog within that application.

Often, though, the title bar isn't enough to show the context of a specific dialog (which Contact are you editing, for example, or the details of which message?). In this case, a good approach is to ensure that the dialog fields contain enough information for the user to be able to find out. In the Series 60 Message Details dialog, for example, there is a 'Subject' field – it's redundant, since the user will know the subject from the previous view, but is there so the dialog passes the 'Cup of Tea' test.

Thus in Figure 17, in the main messaging window (left), the icon shows that it's the Messaging application, and the caption 'penrillian' shows the name of the current message service. And the message details dialog, right, includes a redundant 'Subject' field.



penrillian

billing@btinternet...
Confirmation of your Direc...

Juliet Hewitt <Julie...
RE: BT Yahoo! Internet: VAT...

internet.billing@bti...
BT Yahoo! Internet: your c...

Options Back

penrillian

Message Info:

From	Juliet Hewitt <Juliet @PENRILLIAN. com>
Subject	RE: BT Yahoo!

Ok

Figure 17: Showing Context in the Series 60 Messaging Application

A picture is smaller than 1000 words

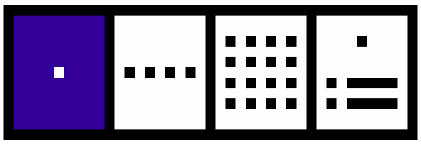
How should you represent common controls and items?

- Common controls can be represented by words (repeated on every screen)
- Words can take up lots of space
- Words need to be translated for international interfaces

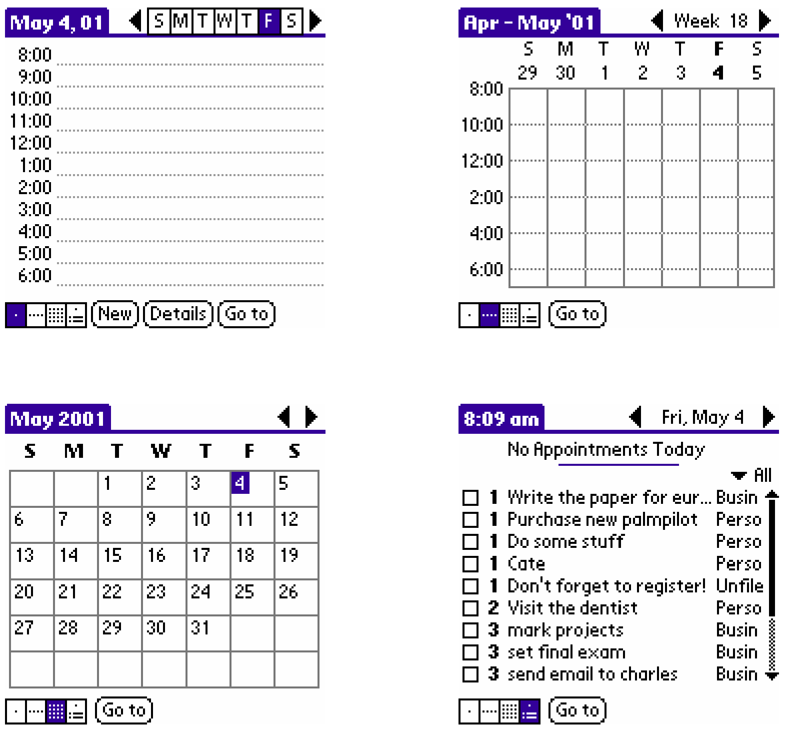
Therefore: *Use pictures!*

By replacing text with *well-chosen* images, you can provide just as much information as the text, in much less space. For example, the Palm diary application has four views, which are chosen using simple icons at the bottom left of the screen. The association from views to icons is obvious in the design of the icons, which are miniaturised versions of each view.

Here is an enlargement of the icons:



And here are the four views (one day/one week/one month/day plus to-do items):



S	M	T	W	T	F	S
		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	31		

☐	1	Write the paper for eur…	Busin
☐	1	Purchase new palmpilot	Perso
☐	1	Do some stuff	Perso
☐	1	Cate	Perso
☐	1	Don't forget to register!	Unfile
☐	2	Visit the dentist	Perso
☐	3	mark projects	Busin
☐	3	set final exam	Busin
☐	3	send email to charles	Busin

Figure 18: Calendar Views

UIQ uses similar icons for its calendar furniture; the top left icons select views, the right select days of the week. And on the shared task bar, the keyboard icon brings up the on-screen keyboard and the phone icon brings up the phone application.

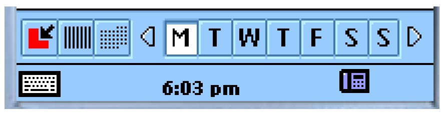


Figure 19: Bottom of the UIQ Calendar application

Consequences

You have pictures. People like pictures, especially in colour. The icons are generally much smaller than the words they replace.

However: Icons are often less intuitive than the words the represent. Users will need to learn what the icons mean. And obscure icons for uncommon commands will never be learned.

Big Thumb

How big should you make common controls on the screen

- Many pocket computers have touch screens

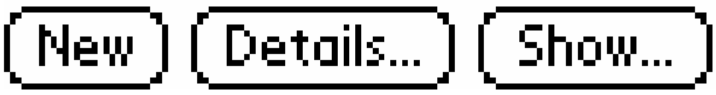
Controls for applications are important, but they take up space on the screen

- Most people use styluses to use the computer
- But perhaps they don't, e.g. one handed use

Therefore: *make important controls big enough to press with your thumb!*

Making important controls large enough to be used directly also makes them easier to select with a stylus, because Fitts law states the time to select something on screen is inversely proportional to its area [Fitts].

Thus, for example, in the Palm pilot To-do application, the buttons across the bottom of the screen are large enough to be used by the pad of a thumb or forefinger (or the back of the nail).



Consequences

Big controls are faster to use than small ones. You will be able to use your pocket computer one handed. You will be able to use your pocket computer when you've lost your stylus between seats in economy class.

However: Big controls take up more space than small ones! And people may not use the machine one-handed anyway, because it makes the screen too grubby.

BIG THUMB has no value in keyboard-only UIs, so is never used in the Series 60 look and feel.

User Customisation

How do you make a small window display as useful as possible to the user?

- A small window makes it difficult to display lots of information for the user
- But users don't want to have to navigate through many different windows to find what they want.
- If we want the window display to satisfy all possible users it must be generic
- Pocket computers typically live in just one person's pocket!

Therefore: *Tailor your display to show the information most likely to appeal to the current user.*

On a 'personal' device like a hand-held, we can identify the user, and use this to make navigation easier. So store information about specific users: what options they've chosen in the past; whether they're novice or expert; perhaps even age and gender. Tailor the display to show only functionality most likely to be used by the current user, but ensure there are mechanisms to get to other data if necessary.

Consequences

Your application will show only the data of interest to the user. Users don't need to scroll or select information to see what is of interest.

However: Users have to customize their displays, or we need extra functionality to do it for them. Users can appear to loose information if they customize so that it's not displayed and forget they've done so. It's hard to use someone else's machine if the user customisation settings are different. It's hard to write documentation or help systems when the behaviour of the system depends on its customisation.

Implementation

There are two main implementation rules for this pattern:

- The default setting should display all the information users require, and
- It must be easy for users to set the software back to a sensible state!

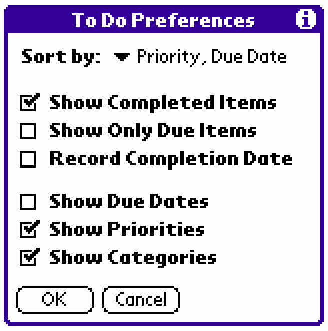
There are two common ways of collecting the customisation information:

Explicit User Settings (Configuration)

The user 'configures' the application to behave in the most appropriate way for their needs. This approach leaves the user feeling 'in control', but has the disadvantage that inexperienced users may not find or use the configuration settings.

For example all MS Office applications store which toolbars and scrollbars the user has chosen.

Often the data that appear on the screen can depend on the user's settings. Palm does this well, for example, its To-Do list has options for sort order, whether to show completed items, whether to show categories, and these are controlled by a separate preferences dialog.


To Do Preferences

Sort by: Priority, Due Date

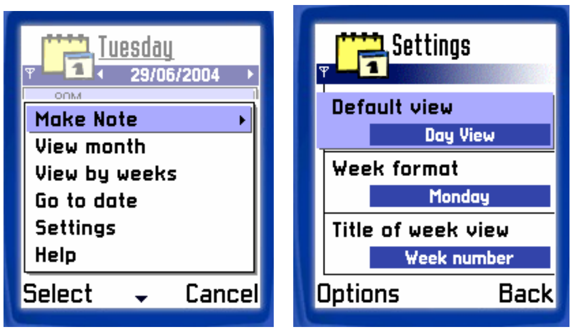
☑ Show Completed Items
☐ Show Only Due Items
☐ Record Completion Date

☐ Show Due Dates
☑ Show Priorities
☑ Show Categories

Figure 20: Palm OS Preferences

Palm also displays in a choice of regular or large fonts; its address book allows users to choose their preferred sort orders, all Palm applications that use **CATEGORIES** can restart showing the category they were last displaying (also, note the **DIAL 'H' FOR HELP** button in the top right again, and the **BIG THUMB** OK and Cancel buttons).

Many Series 60 applications also allow user configuration, normally accessed from a Settings menu item. Here's the configuration for the Calendar application:



Default view: Day View
Week format: Monday
Title of week view: Week number

Figure 21: Series 60 Calendar Settings

Another example is WAP, a small-screen version of the HTML standard. Like HTML, WAP supports 'cookies' so that the server can store information related to the user on the device. Many WAP sites use these to allow the user to set up personal settings. Kizoom, a UK railway timetable service, allows the user to set up their daily commute stations, a home station, and preferences about timing of connections and the like.

Implicit User Settings (Learning)

Here the system learns from the user choices, and sets its defaults according to previous choices.

One of the most successful examples of this is Amazon, which configures its home screen according to the user's previous purchases, and remembers previously-entered addresses and payment details [Amazon].

Another example is the menu system in MS Windows 2000 (as below); this will remember which menu options were selected recently by the user, and initially display only these options (as below). An 'extra' menu item displays the remainder, as does waiting a few seconds.

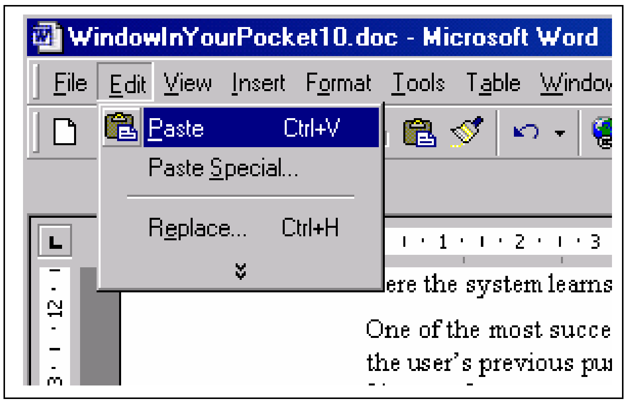


Figure 22: MS Windows 'Learning' Menus

Acknowledgements

Many thanks to our EuroPLoP shepherd, Dragos Manolescu, who provided valuable support getting this paper to an initial form. Thanks, of course, to our workshop group; we've learned a lot from thinking through their comments.

References

[Alexander] *A Pattern Language. Towns Buildings Construction.* Christopher Alexander et al. 1977 Oxford University Press.

[Amazon] http://www.amazon.com

[Fitts] Fitts,P.M. (1954). *The information capacity of the human motor system in controlling the amplitude of movement.* Journal of Experimental Psychology, 47, 381-391. Also at http://www.well.com/user/smalin/miller.html

[Gamma] *Design Patterns* by Gamma, Helm, Johnson, and Vlissides, Addison-Wesley 1994 0-201-63361-2

[Kizoom] http://www.kizoom.co.uk

[Kraftwerk] *Die Mensch-Machine*. Capitol Records.

[Noble&Weir] *Small Memory Software* by James Noble and Charles Weir. Addison-Wesley 2000. See also http://www.smallmemory.com/

[Miller] *The Magical Number Seven, Plus or Minus Two*, George A. Miller, The Psychological Review, 1956, vol. 63, pp. 81-97

[Tesler] Tesler, Larry. The Smalltalk Enrionment. *Byte Magazine*. August 1981. pp90-147 .