

Creating a touch-based task switcher

The task bar re-evolved using tabletop direct touch

Bas Peschier

Department of Computer and Systems Sciences

Degree project 30 credits

Degree subject: Human-Computer Interaction

Degree project at the Master level

Spring 2011

Advisor: Annika Waern

Reviewer: Patrik Hernwall



Sometimes the best thesis partner is the one you already see every day

Creating a touch-based task switcher

The task bar re-evolved using tabletop direct touch

Bas Peschier

Abstract

After researching using touch as a primary input method for personal computers, a case was made for a touchable task bar above the keyboard. This exploratory thesis contains a literature background and design for such a task bar, incorporating both the location and the manipulation and display options that location affords.

The task bar was designed with both task management and touch manipulation background in mind, building problem areas from literature and creating solutions using the possibilities of the location and touch-screen. The focal points are, next to the location and manipulation, window grouping and the use of more elaborate window representations for notification and simple interactions.

This design was tested with a prototype, built on an Apple iPad and observed five participants in a combined contextual inquiry and prototype test. The results both confirm the hypotheses about the usefulness of the features, as well as confirm the problem areas which based the design. They also, however, uncovered areas where further work needs to be done to get a clearer results, such as the used semantics of the window grouping and the implications of designing for accessibility.

Keywords

task switching, window management, direct-touch interfaces, notifications

Contents

Part I: Introduction and background

1	Introduction	3
1.1	Bringing touch into the equation	3
1.2	Research description	3
1.3	Goals	5
1.4	Methods and structure	5
2	Background	7
2.1	Task management	7
2.1.1	Terminology	7
2.1.2	Tasks and computers	8
2.1.3	Causes for switching	11
2.2	Touch manipulation in the horizontal plane	11
2.2.1	Orientation and perception	12
2.2.2	Occlusion	12
2.2.3	Direct manipulation	12
3	Current task switching implementations	15
3.1	Major desktop operating systems	15
3.1.1	Task launchers	15
3.1.2	Task switchers	16
3.1.3	Notification	17
3.2	User-space window managers	17
3.3	Mobile platforms	18
3.3.1	The illusion of multi-tasking	18
3.3.2	Window management and flow	18
3.3.3	Notification	19
3.4	Academic projects	19

Part II: Design of a touchable task bar

4	What is wrong with my current computer?	25
4.1	Problem areas	25
4.1.1	Getting around: switching costs	26
4.1.2	Picking up where you left off: resuming costs	26
4.1.3	Keeping updated: notification	27
4.2	Working towards possible solutions	27
4.2.1	Window grouping	28
4.2.2	Location	28
4.2.3	Richer window representation	28
4.2.4	Direct and more complex manipulation	29
5	Principles for switcher design	31
5.1	Place the switcher on top of the keyboard	31
5.2	Use direct and complex manipulation	31
5.3	Make windows groupable into tasks	32
5.4	Use rich window representations for notification and simple interactions	33
6	Prototype design	35
6.1	Window design	35
6.1.1	Size and placement	35
6.1.2	Interaction	36

6.1.3	Implemented representations	36
6.2	Task design	38
6.2.1	Mechanics of window grouping	38
6.3	Technical background	38
6.3.1	Touch operation limitations	39
Part III: Putting the design to the test		
7	Study design	43
7.1	Participants	43
7.2	Prototype study setup	44
7.2.1	Part 1: Getting a baseline	44
7.2.2	Part 2: Acclimatisation	45
7.2.3	Part 3: Main tests	45
8	Results	47
8.1	Validity of problem areas	47
8.1.1	Switching costs	47
8.1.2	Suspend and resume	48
8.1.3	Notification	48
8.2	Effect of solutions	49
8.2.1	Notification	49
8.2.2	Grouping	50
8.2.3	Ease of access and gestures	50
8.3	The perceived usefulness	51
Part IV: Wrapping up		
9	Discussion	55
9.1	The semantics of window grouping	55
9.2	Limitations of the prototype	55
9.2.1	Required physical space	55
9.2.2	Launching and resuming	56
9.3	Location of the prototype	56
9.4	Learning curves and "natural" interaction	56
9.5	Accessibility	56
9.6	Validity of study methods	57
9.7	Who is the user?	58
9.7.1	The philosophical foundations of motivation and appropriation	58
9.7.2	Appropriation as iterative experiences	58
9.7.3	The user as a practical problem	59
9.7.4	The user as a knowledge question	59
10	Summary and conclusions	61
10.1	Design and prototype	61
10.2	Limitations and scope of study	61
10.3	Conclusions from the studies	61
10.4	Overall conclusions: answers and solutions	62
10.5	Future work	62
Appendix: Study structure		65

Acknowledgements

This thesis was not possible without a couple of people who helped me in the course of not only the project, but also the steps before. Adnan, Tian-lin and Jinyi were instrumental in creating a previous design for the UIST conference, fuelling ideas and sketches. Of course the five participants in the studies showing me my ideas were valid, but also steered towards what needed to be done. Also thanks to Tessa and Tian-lin for giving me the opportunity to structure my thoughts while listening to my endless rants, while Patrik provided an objective and good first review with excellent points to work with, both in implementation and thought. Finally, my supervisor Annika and the thesis "support" group for taking up a foreign idea and convincing me I was on the right track in our fruitful and fun therapy sessions.

The touch gesture in figure 6.3 is based on the *Touch gesture reference guide* by Craig Villamor, Dan Willis, and Luke Wroblewski. It can be found at <http://www.lukew.com/ff/entry.asp?1071>.

Part I: Introduction and background

Personal computers have become a part of everyday life; most people cannot imagine a world without them. The platform is highly flexible and the machines are used for consumption and production, both at home and at work. Although the general characteristics of the platform have not changed that much, developments in both hardware and software are changing how we all interact with them every day.

A very basic example is how one switches between tasks on this multifunctional device; this is a very natural occurrence: we read articles on our browsers and switch to make notes, we change projects at work by sending off an e-mail to a co-worker with the latest news and switch to the window needed for the next task. What we are not aware of is the fact we organise ourselves so we can make the best use of the artefact we are using; an artefact which might not be doing the same job of organising itself to support you.

This does not mean the artefact is unusable – far from it – but it does leave room for improvement. One possibility of improving the interactions for switching is by placing it in a different location, somewhere we have easy access to it with touch-based input mechanisms.

To understand what we can do to improve the experience of working on multiple tasks on a computer in such a situation, one first needs to understand how people organise tasks and how they can use touch systems...

This introductory part includes the general introduction to the work and its goals, followed by chapters on the related backgrounds of task management and manipulation, ending with a look into how current systems tackle the problem.

1. Introduction

Task switching operations are mostly window manipulation operations in our current overlapping window-based systems and thus the *window manager* dictates how the user can layout their work. But is just switching windows actually what people do to switch between tasks?

When asking oneself such a question, it quickly dawns that switching tasks is about more than just windows and can be seen as part of a larger process with an external source. If one then looks back at the methods we use for managing that process on the computer, it becomes clear that there is some difference between how the system structures activities and tasks and how the user structures them.

There is room for improvement, improvement which caters to both how we use switchers and how we operate them. To find a place for improvement, we either need to suggest from literature about the nature of task switching or we need a suggestion for a better mechanism; one such mechanism might be using touch-enabled hardware: we already use keys for quick input, why not extend that to more complex operations?

1.1 Bringing touch into the equation

When Peschier and Majid (2010) looked at the current setup of our computers and set out to explore the option of using multi-touch and the option for direct manipulation for interaction with a day-to-day personal computer, they found out the top area of their setup was suitable for a "switcher"; a component for switching between windows.

Their setup was based around removing the keyboard and mouse entirely and replacing it with a large touch-panel which could discern between multiple inputs; this was their solution-based problem investigation: *what kind of interfaces does this setup afford?* By looking at the affordances of the location and input method and comparing that with a list of current systems' interface component categories and their attributes – how do these interface components operate – they made a further work suggestion for a "task bar".

The suggestion of the "task bar" replacement on the top of the setup was based on the compatibility of switching interface components with their setup: the possibility for direct switching by touching – mimicking the direct nature of keystroke-based switchers and the needs of launcher-type components, the top location which was not obstructed while used and in peripheral vision which matched with notification components not unlike current switchers such as the Windows task bar and Apple OSX Dock, as well as provide a neutral location for displaying running applications *and* launcher-type interface components.

All these arguments led to the suggestion of creating a window switcher in the region just above ones keyboard and is depicted in figure 1.1. The location and input method is a *given* in this research and used as a starting point for a design-oriented research into task switching.

1.2 Research description

This document is mainly structured and implemented around the methodology and perspective of *design science*. This concept was popularised by Hevner et al. (2004) and concerns the relation between knowledge and practice and especially the conviction designing useful things can yield scientific knowledge. Wieringa (2009) expands Hevner et al.'s framework methodologically by discussing the concepts of practical *problems* and knowledge *questions* and their impact on how research is done.

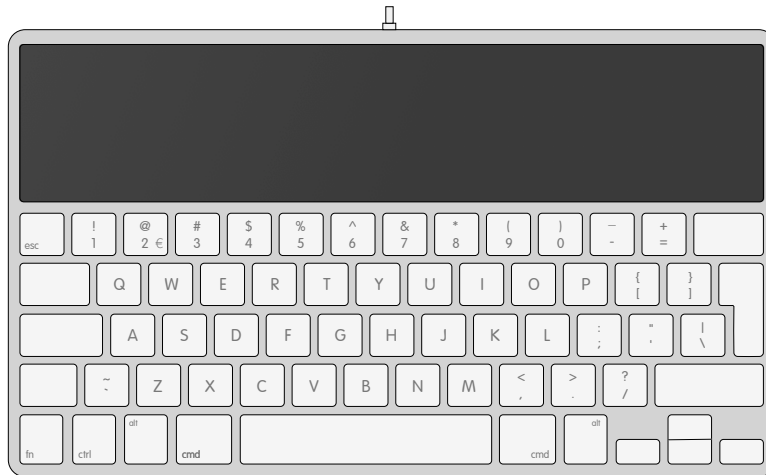


Figure 1.1: The suggested location for a task switcher by Peschier and Majid: a touch panel in the top area of the keyboard

In design science, all work begins with a practical problem, which in this case is: *How would a task switcher using the location on top of a keyboard in peripheral vision with direct manipulation interactions look like?* To give a context for the concepts in this question, the following background is used:

Peripheral vision is the outer part of a vision system, which in case of human vision is the range outside of the focal area of the gaze. Colour and shape are less distinguishable in peripheral vision, while it is sensitive to motion.

Direct manipulation is a form of manipulation where the physical location of the manipulator and manipulated object are the same; this in contrast with indirect manipulation like a computer mouse where the manipulated object (a window for example) is at a different location than the manipulator (the mouse).

This question is by no means a knowledge question, but it is surrounded by knowledge questions both influencing the design *and* its impact. Design science is science because of the contributions along the way, which require problem relevance and solid methods to validate findings.

First off all, to create a design, the problem calling for that design has to be defined and explored starting off with the open knowledge question: *What problems are there in task switching?*, followed by *What benefits and limitations do direct manipulation bring?* Answering these questions is a practical problem in itself and will touch upon both backgrounds of the concepts behind the design: *task switching on computers* and *direct manipulation in the horizontal plane*. This background thus is the first part in this work; this work reflects the practical in the way the background is used as a literature study with the design characteristics in mind, providing the foundations for the design.

Having identified its background, the problem area can be described and design comes back on the table when coming up with solutions. This part starts off with a problem investigation both *goal-driven*, since we are investigating an area which might not be broken but our goals require further investigation, and *solution-driven*, since the design is based on a pre-set solution looking for a problem – the touch-screen – and reasoned from that point. Wieringa, the proposer of this classification, does not put iron fences around the classes and discusses the two classes in light of "solving a problem which is not experienced yet" and how a design can solve this by "changing the world". The context in this work brings up an interesting philosophical concept diving deeper into that expression: *who* determines our current situation is broken? The *experience* of a problem, especially in such a highly customisable environment as our *personal* computers, is by no means set in stone.

The resulting design again brings us back in the domain of scientific knowledge: *does the design work?* This is both done by internal validation – the arguments used to make design choices and cover the entire problem space – and by means of an implementation; a prototype, which can be studied and tested with users.

1.3 Goals

The first goal of this thesis is to explore the possibilities of using touch in the suggested location near the hands of the user. To achieve this, a design will be argued for from both literature and quick user testing. The literature review will look into both task management and the specifics of the proposed platform.

The design of this window management tool is implemented as a prototype which is also used to test the design. It uses the background study for the location and touch-enabled characteristics, as well as theories from task management on computers as a basis. The process will lead to a prototype of such a system which can be tested with users.

Second, the goal is to evaluate the aspects of the prototype with users to see whether or not the solutions actually provide the improvements hypothesised. It will test whether or not the user experiences improvements with notification and lay-outing windows, as well as experiencing less costs to suspend and resume tasks or to switch between them.

In the end, this work results in a practical design surrounded by answers and discussions on knowledge questions leading to the design and concerning the evaluation of the design.

1.4 Methods and structure

To reach the goals denoted above and the answers to the questions described in the research description, a couple of steps must be taken. First, a case will be built in the remainder of this part based on background literature covering theories and experiences on related physical setups and task management on computers in general. This background is partly based on the previous work by Peschier and Majid (2010) which led to the suggestion of the task bar and also incorporates a look into other task management solutions in the public domain and academia.

In part II that background will be brought into the design of a prototype task bar. The prototype is designed in an iterative fashion with short development cycles followed by a "quick-and-dirty" evaluation. The final design is described in chapter 6. The platform used is an Apple iPad for the touch area with a platform-agnostic service with implementations controlling a Apple OSX computer. The method was chosen for its flexibility and the ability to get feedback quick while keeping end-users involved during the design process.

Part III is concerned with a small prototype study focusing mainly on qualitative aspects of the interaction. Because of the open nature of the problem and implementation nature of the prototype, as well as limitations of the platform, there are no real reliable quantitative metrics to measure. The only baseline available is a qualitative one, which is obtained by doing a contextual inquiry: an interview and observation on location with the participant into their current behaviours and motivations. This part of the study will also yield information about the validity of the problem area reasoned and provide the opportunity to discuss the underlying perception of the problem by users.

This qualitative method fits in multiple ways: since attitudes towards users' systems and possible new interactions with the prototype are highly subjective and personal, a contextual inquiry will provide the most reliable baseline for analysing their reactions to the prototype's interactions and possibilities. The same goes for the users' opinion on how the prototype functions or could be functioning for their own system and habits. Besides that, a lot of information about the ease of learning the interface can be extracted by pure observation with a qualitative baseline as context.

Finally, chapter 8 presents the results from the studies, which will be discussed in the final part IV, along with the final summary and conclusions.

2. Background

The design of the task switcher relies on two major factors: the theory of how users use computers to perform their tasks and the effects of placing the switcher near our hands using a touch screen. In the first section, an overview is given on task management on computers; this involves getting out a list of terms to clear the sky of misconceptions, while being followed up with perceptions on task management both from a human organising their tasks, as well as interruption of those tasks. The second section covers effects of the physical location by looking at similar systems: tabletops. Issues like occlusion of information and perception under angle are two of the subjects touched upon.

2.1 Task management

In starting to understand what improvements can be made, one needs to take a good look at how people manage tasks on their computer. The term "task management" in context of computers is a vague one, however. Before it can even be explained what task management on a computer is, we should start with defining concepts.

2.1.1 Terminology

Most confusing elements of task management on computers come from inconsistent terminology of vendors; if one takes Microsoft Windows as an example, terms like windows, tasks, processes and applications are used almost interchangeably at least for what the end-user is seeing. The "task *bar*" lets us switch between windows and the "task *manager*" is a hodgepodge of windows, processes and other system information.

To keep the terminology clean in the rest of this document, concise concepts will be used for each of them. They are ranked in order of probable awareness by the user, starting with the omnipresent *window*.

Window A window is a single, atomic, view of an application and is a dedicated separate part of the screen. In most modern system these windows can overlap, which is often called "the desktop metaphor" Myers (1988) since it mimics the interactions of papers on a desk. Basic windows operations are handled by the window manager of the operating system and not by the application. For all intents and purposes, this is the smallest cog in the task switching system.

Application An application is a computer program with a user interface as presented to the user; they exist in the user's perspective of the system. They control the content of the windows associated with them. Applications range from elemental single-window calculators to enormous office automation software. It is often possible to have multiple instances of an application running; a prime example of this is multiple documents opened by a word processor. Conceptually this seems to be all one application, but interpretations vary based on platform and understanding of the inner workings of systems. These conceptual differences will be addressed in chapter 3 on page 15.

Process Processes usually do not surface often on the radar of end-users. These are the logic and operations underlying applications and can be seen as the system's perspective of applications. Processes are mainly dealt with when applications go haywire and the underlying process needs to be monitored or even shut down.

Activity An activity is a part of a task and are the operations needed to complete a task. This concept is not directly transferable to a window or applications, since both could be used for an activity or even a combination of different windows and tasks. A simple example is the calculator; it is a utility, but it still is a full-blown application. Doing calculations will in most cases be an activity within a larger task.

Task Tasks are the semantic high-level groupings of activities as used by humans. Examples are "working on my thesis" or "designing a product". There is no direct system-level equivalent in most operating systems at the moment and this is catered for by the user by managing windows. Tasks can be small – containing only a handful of activities – or large – being divided into subtasks containing their own activities.

2.1.2 Tasks and computers

Bannon et al. investigated in 1983 how "users of computer systems" manage to work on multiple activities at the same time. These interfaces were still text-based, but already featured mechanisms to suspend and resume applications to switch between them. Although this categorisation and analysis was done 28 years ago in a time where the pre-cursors to our current windowing systems like the Apple Lisa were just surfacing, the guidelines are defined on an overarching level focusing on the human perception of task switching on computers. This makes their findings relevant even today and Bannon et al. are thus still referenced in new work created.

They coin the term *workspace* to signify an environment which allows easy manipulation of *activities* to achieve a goal or a related set of goals. To achieve this, the system should contain semantic information about the activities translated from the intentions and structure the user has for them. Activities in this text-based context can often be translated to windows in current major operating systems.

Their observations yield a set of coordination issues when it comes to switching between the activities and they argue for the following guidelines, which are discussed and put in contrast with the context of current computing contexts:

Reduce mental load when switching tasks This boils down to keeping switching easy; for the user the switch is relatively quick and the user wants to input on the new task directly; for the system the switch might entail more complex operations, some of which might be offloaded to the user like keeping track of what task is where. Ideally, the switch should be as lean as possible.

Easy switching becomes extra important when the task switch itself is very short; if you get interrupted during a task for something really simple – either by someone else or when you suddenly remember something, you are only going to be in the other task for a couple of seconds by making a note or something similar. If this switch requires you to handle all sorts of interactions, this strains the experience of effectively handling the situation.

Suspending and resuming tasks Bannon et al. observe that most complex tasks almost never get completed before the user makes a switch. It can be argued that this could well have become "worse" in our current interconnected world. Like above, people get interrupted or their schedule demands they switch. The effort required to pick up the task when they have time to work on it again is directly related to how people left their work (Nagata; 2003) and how the state of that task is presented when picking up. The system should provide methods for easily suspending tasks and resuming them instead of having to force users to spend a lot of time preparing the switch due to the system removing state; people will always try to leave the state in something *they themselves* will remember on an abstract level beyond the knowledge of the system (Czerwinski et al.; 2004).

Maintaining records of activities Activities can be as small as operations - especially in the console-based interfaces which Bannon et al. investigated - and maintaining a history of previous commands to quickly re-use them improves usage. Nowadays this is mostly featured in the undo/redo commands, but this can be seen as a memory of operations which can be re-applied when needed. This operation-memory does not contain state, so re-applying an operation by no means implies the same result; for example, if one re-applies the rotate 90 degrees function on an image, the picture will be rotated 180 degrees after a second go.

It must be noted that such mechanics seem to fit better with text-based interactions and require some thought to transfer broadly into the realm of windowed systems. This does not, however, take away from the usefulness of such interactions.

Functional groupings of activities Activities are often related and used to achieve a single goal and thus the windows and applications representing that activity can be grouped as well. A system can for example assist a user by providing mechanisms to keep track of these relations. In this way, the system keeps track of the view the user has on how windows are related, not only on how they relate from a system's view where windows are only related to their underlying application. This functionality can be implemented in a variety of ways, but most boil down to a form of *virtual workspace* in which windows get grouped together in a single space.

Multiple perspectives on the workspace Since humans are not machines, they lose track of what they were doing. The "messy desk" analogy is a good example of how lack of structure helps this process where the user gets lost because of all of the open windows. One way to help a user resolve such issues is to not only create structures, but let the user choose between multiple structures. These multiple perspectives on the same activities could solve this for the user, by for example switching to temporal or goal-based ordering.

In a sense, the different ways one is able to navigate our windows, such as visual recognition switchers as Apple's Exposé or the ordered list of the task bar, are an implementation of this guideline, but the structure in which the workspace is presented is often a system's perspective.

Interdependency Except for activities being related to each other for some task, activities could also be related to multiple tasks. An example for this is when a programmer works on multiple projects – tasks – which use the same documentation for the framework they use on all major projects. The view into the documentation could be shared across workspaces, if it would be deemed needed. Bannon et al. thus suggest it should be possible for activities to have multiple instances which are linked.

This behaviour is by far the least implemented guideline; it is often possible in third-party window managers to mimic such behaviour by creating a sticky window, but the window is simply transferred from workspace to workspace and not linked. This means the window cannot be fit in to the workspace, with its own position and size.

Since these guidelines target the abstract task switching level, they are still very valid for new interactions and flow well into the windowed world of today from the text-based console days of the eighties. Multiple of these guidelines hint at giving the system a mechanism for maintaining the state the user has instead of just the logical connections made by the system. It seems to make sense that users can – with minimal administration – convey *their* perspective of *their* tasks onto the windows and applications.

Bannon et al. touch upon subjects which look like backbones for problems in current systems: they talk about the costs to switch between windows, how suspending and resuming tasks impact task management, as well as the possibility for grouping windows to make switching context-aware. In our eyes, these areas are either partly solved or not solved at all. These all tie back to the context of *workspace*.

Workspaces

The term workspace in computing still is an often used mechanism for maintaining those relations. *Workspaces* or *virtual workspaces* are environments in which users have multiple virtual versions of their desktop. Each desktop contains a set of windows and represents a view the user chooses for it. Some people put all their communication on one workspace and others create a workspace per project they work on. Although workspaces do not force any pre-set method of use, they often are used to portray tasks (Ringel; 2003). In this sense, each virtual workspace is either a task or a subtask with the windows belonging to that task. The other end of the use-spectrum is to assign *one* application to each virtual workspace, but this is often done because the workspace manager has easier mechanisms to switch between them than the operating system can provide for single applications.

Tasks are also often grouped together in a specific order, which is also seen in multiple-monitor setups. Grudin's paper on the use of multiple monitors describe clear use cases of them; one screen (or workspace) is used for "primary" applications resembling the more important tasks, while applications for secondary tasks like messaging and browsing are on the second screen. There is however a big difference between multiple visible workspaces and virtual ones, since multiple visible workspaces can be seen as one larger workspace with interchangeable parts and different visibility. The second screen often lies in the peripheral vision of the user where information can be glanced at and notifications are less intrusive: not all information is equal (Grudin; 2001). The two workspaces can also be closely connected, where for example the second screen holds the documentation of the application on the main screen.

Systems with multiple workspaces actually have two types of switching: between the spaces and between the windows in the space. But for example, Apple's OSX actually gets this "wrong" at the moment: while the workspaces in Apple's Spaces can divide up the different windows, the window switchers switch between all windows across all spaces, instead of within the specific workspace. This does not improve the costs of switching and instead burdens the user even more with remembering on which workspace applications are. These kinds of interactions make users devise their *own* methods to manage their windows.

Appropriation and habits

Throughout the literature and background cases are made for certain implementations; a couple of them can be found in chapter 3. What can be concluded from them is that one has to keep in mind that people try to make their current situation work. People game systems they often work with to configure them in a way that suits their methods and work habits.

The example of people using workspaces for single windows is an excellent example where the use of the tool is neither a system's view or a user's view, but a mechanism which makes the task of task switching easier. This points straight back at the first guideline by Bannon et al.: reducing mental load when switching.

This development is often called *appropriation*, the method of making an artefact your own. Appropriation has no definitive definition, but hinges around the concept that bringing an artefact into one's life is not the end of a story; one uses it and learns how to use it, while in the mean time adapting one's own stance and use of it. This process is a loop where new experiences continue to feed the system and people adapt further or adapt the artefact to mirror their needs of it.

In the complex customisable systems of our computers we all use every day, this mechanism diversifies us from – in essence – the same starting point and creates our own personal stance towards and habits while using the system. It can be very hard to break these habits or this image of the system since it is worn into our routines. Explaining qualitative feedback is thus not a trivial task and the complete context has to be taken in account, calling for some baseline before trying to do so.

2.1.3 Causes for switching

Dealing with multiple tasks within the area of human-computer interaction is often approached via the mechanism of interruption. While a user, in the course of using a computing device, often switches between tasks for self-inflicted and planned reasons, a significant part of all switches are forced by external influences (Czerwinski et al.; 2004; Nagata; 2003; Iqbal and Horvitz; 2007; Spink et al.; 2009). Examples are incoming e-mails or IM messages, but also phone calls and other non-computer sources. Not all interruption is completely external since sudden realisations and other flashes of memory can interrupt a user just as well as somebody else.

These interruptions might cause errors when picking the task back up, depending on the state the user left the workspace in – something Bannon et al. also surveyed, but not all interruptions are equal. Users treat interruptions with different levels of importance; they often finish up their current task before switching if the interruption allows it, based on both the type of interruption and the importance (Nagata; 2003). This way, they leave their work in a more stable situation so picking up is easier.

The second aspect is the only aspect missing from Bannon et al.'s guidelines: *notification*. While related to the suspending and resuming guideline, the actual method of interruption is not included in the discussion. This is however important for the user as it provides a level of control over the decision to be interrupted. In these days of interconnectedness even when in transit, the method of delivery and the payload of the notification heavily influence their usefulness.

This second aspect of interruption is thus the way users are notified of the state of the system and applications. Users also keep a lot of windows open just to be able to glance and see what the state is (Hutchings and Stasko; 2004b), both for keeping an eye on when a task switch is needed, but also for resuming tasks when interrupted (Czerwinski et al.; 2004). For now, most operating systems have a notification area, but, due to the size of the area, this is a very limited way of conveying state since it only tells you something has changed, thus making the decision how to act on the notification harder. Also, a lot of notifications are temporary like the toast pop-ups¹, requiring the user to shift attention or miss it.

Control over these interruptions is thus important for users, since delaying them makes it easier for them to pick up their work (Iqbal and Horvitz; 2007). Choice over the time of reaction to an interruption also makes it possible to coordinate switching tasks like taking a break. Control is a major psychological factor in interruptions (Nagata; 2003; Iqbal and Horvitz; 2007), but it is equally important to guide users back to what they were doing (Czerwinski et al.; 2004). An example of these situations is when people choose to finish a paragraph before answering an incoming e-mail. If the notification includes enough information to make a decision whether or not to answer directly, in a short while or when the user is done with the activity, users can plan and make the switch on their own terms.

2.2 Touch manipulation in the horizontal plane

The location of a task switcher in the horizontal plane of the keyboard leaves the question of how well that area is actually suited for more complex interactions not only with touch input, but also with dynamic visual information like notifications displayed in that area. Since keyboards with screens are nowadays prototypes at best with the Microsoft Adaptive Keyboard² as a prime example, information and theories about working with screens in the horizontal plane come from research into tabletop computing.

Tabletops in this context are table-sized interfaces, often employing touch surfaces for main input and using the surface of the table for screen output. Observations with direct touch tabletops have been summarised by Ryall et al. (2006). They claimed the design of such systems is still in their infancy and observed the benefits and problems of a couple of systems.

¹A toast pop-up is a small pop-up sliding into view just like a toaster does when bread is toasted

²See <http://www.microsoft.com/appliedsciences/content/projects/uist.aspx>

These systems are all large coffee table sized systems and are used for interactions with more than one person. They categorised their observations into *physical setup* and the *content and layout of information*. Their findings range from problems with text input to physical attributes as leaning on such a large area. Since the context often is not the same as our context not all points are valid, but because of the similarities in physical layout and use, quite a lot of them are.

The aspects which remain applicable are the aspects concerning the physical setup and implications for basic use, which were the basis for the suggestion of the task bar by Peschier and Majid: issues with the *orientation, occlusion* of the hands and effects of *direct manipulation*.

2.2.1 Orientation and perception

The horizontal plane makes for a couple of coordination and perception concerns. In the case of tabletops this is mainly caused by multiple persons using one tabletop, where everyone has their own unique physical perspective on the table and they need to coordinate access in case of obstructions (Ryall et al.; 2006). When looking at inherent perception concerns for a single user, the orientation still is of concern.

This is because, while the coordination issues are not relevant in a single-user setup, the angle and orientation towards the device is still relevant for how the user perceives the information. Orientation disturbs the perception on a horizontal plane Wigdor et al. (2007); angles get distorted and the user cannot discern between angled information: the interface should not use angled information if it is relevant.

On top of that, when comparing items on a horizontal plane, it is better to keep them at the same distance from the user. This is because one cannot correctly assess the differences at different distances (Wigdor et al.; 2007). A solution is to keep horizontal bars or bands to keep things at the same distance.

These two issues were the original argument for the use of "bands or bars" by Peschier and Majid: strokes of non-angled information at the same distance. The ramifications run further in case of looking at the task switcher itself; although it is already a single band, information within the switcher should not be angled unless it is orthogonal.

2.2.2 Occlusion

A problem of a different kind is occlusion. Because of the physical properties of the tabletop and the input device, being your arm, one occludes the part of the display below ones arm (Ryall et al.; 2006; Brandl et al.; 2008). Information important to the interaction with the system should therefore not be placed in such a location on the lower half of the tabletop during those interactions.

When translating this to the realm of the keyboard, one can see that occlusion is relevant for beginners; but because on a normal keyboard the *information does not change*, occlusion is no longer a problem when one knows which key is under one's finger. With a dynamic section – the task bar – added, this problem is more permanent due to the dynamic nature: information can change at any time giving no certainty about the information.

Two ways, we argue, which can combat the problems of occlusion are to either make use of motor memory by keeping information on the same location or, alternatively or used together, to make sure the interactions used to operate the system do not occlude the relevant information while operating.

2.2.3 Direct manipulation

Multi-touch tabletop systems such as our suggested switcher are examples of direct manipulation environments. These systems consist of a single mixed input and output surface of the same dimension, allowing direct manipulation of elements displayed (Ryall et al.; 2006). Because it allows for multiple inputs simultaneously, it can be used by multiple users at the same time. This

is a much researched characteristic of these systems, which is however the opposite of our use case: multiple inputs by a single user.

In addition to just allowing for multiple users, the multiple input points also allow for high bandwidth input, since you have ten fingers to work with and the system can capture all of them. This gives you more degrees of freedom in creating complex, but "natural" interactions with the user and because of the direct nature, literature argues, it also requires less attention (Brandl et al.; 2008).

Direct manipulation is seen as superior to indirect manipulation – like the computer mouse – in interactions which model physical operations. Because the interactions based on them are directly linked to their location on the surface, users can easily perform these tasks (Po et al.; 2004; Wigdor et al.; 2007; Ryall et al.; 2006). The manipulation of pictures is a recurring example in this, but this interaction could be transferred into the domain of manipulating task-related objects such as windows.

3. Current task switching implementations

Before designing any task bar it is logical to look at what current implementations of window managers use to switch between windows or other window manager constructs like workspaces. There are many window managers available for all of the major operating systems, both desktop and mobile, as well as on the newer mobile smart-phone platforms. There are also a couple of managers developed in academia to fight certain problems with those found in the operating systems. All of them follow certain principles, some seem flawed to specific users, others enlighten with innovative insights. This chapter looks at them both for inspiration as well as traces of the theoretical background and how they are implemented.

3.1 Major desktop operating systems

Peschier and Majid looked into interface components of currently used operating systems like Apple OSX 10.6, Windows XP, Windows 7 and Linux desktop environments like Gnome and KDE to investigate what kind of interface components were suitable for their large touch-screen setup and noticed everyone favours a slightly different default setup of components (Peschier and Majid; 2010). However, they identified several useful task-switching related components within the different setups: task launchers, task switchers and notification areas.

3.1.1 Task launchers

Task launchers can be divided into three subcategories: *Icon-based*, *Menu-based* and *Text-based* launchers. Launcher are not strictly part of the window switching, but do play a major role in task switching since they start new tasks – or add windows to the current task. This means they are not immediately covered by Bannon et al.'s guidelines, but provide insight in the methods to switch *tasks*.

Icon-based launchers are just the icons of the application you want to start. They vary in size, which in all cases is customisable, and are in most cases placed close to a task switcher component. In some cases, like the OSX Dock and the Windows 7 task bar as seen in figure 3.1, these characteristics are combined into one single icon. Currently running applications are visually marked as running. The one-click nature of these icons makes them suitable for a direct touch interface. The entire component is a list-type component, which allows for the interaction of scanning and selecting.

Menu-based launchers differ from their icon-based brethren by being slightly more indirect. They use the hierarchical nature of a menu. The most used example of such a menu is the start menu of Windows operating systems which has been available since Windows 95 into the latest Windows product, Windows 7. Tasks are placed behind a single node and a user can drill down the menus. Most operating systems employ a similar system, with two exceptions: Apple OSX and Gnome. Apple's OSX does not include a separate menu-based launcher, it only allows you to browse to a list of applications. This leaves the icon-based Dock as the main launcher. Gnome on the other side does not use a single root-node operated menu; it has the main categories of the hierarchy directly on the top bar.

Text-based launchers are a less common launcher in current graphical user interfaces. The shell-based terminals can be seen as a text-based launcher, but our experience with them is that

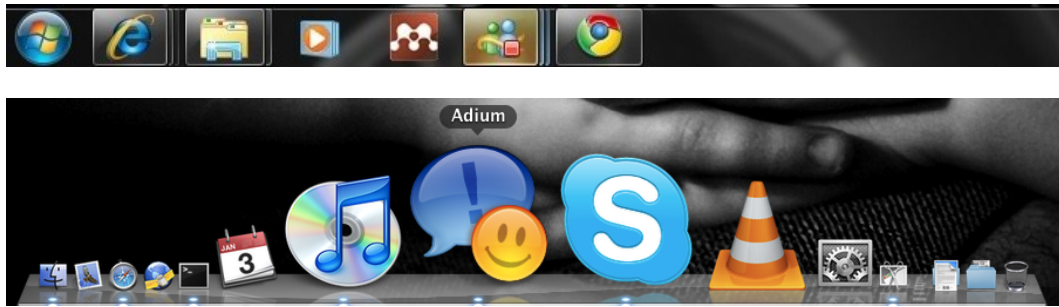


Figure 3.1: Task-launchers in Windows 7 (task bar) and Apple OSX (dock)



Figure 3.2: Notification in Windows 7 and Apple OSX

only power-users really use them. Document-indexing systems like Apple's Spotlight are bringing text-based launchers back in a different manifestation. Users can search for applications just like they can search for documents and launch them. The whole operation is text-based. But since text operations are mostly done with the complete focal attention of the user, our setup is no better instrument for these.

Our setup does seem to prefer icon-based or *visual representations* of applications as they provide easy recognition, as well as be easier to manipulate on a touch surface.

3.1.2 Task switchers

Task switching is another basic interaction found in operating system graphical user interfaces. Switchers can be sub-categorised into *keystroke operated switchers* and *visual switchers*.

Visual switching is switching between visual representations of running tasks. All operating systems employ a "task bar". Apple's OSX and Windows 7 use an icon-only representation, while the older Windows XP and the Linux desktop environments also include the name of the window. The former make use of larger icons. Visual switchers are list-style components. These task bars allow for direct manipulation and are suitable for use in our setup.

Keystroke operated switching is mostly known as "alt-tab"-ing, being the prevalent keystroke for switching between tasks. All viewed operating systems employ it and is an easy and short keystroke. Variations include the option to use the cursor to select tasks within the popped-up list. In these cases it can be seen as a keystroke to show a visual task switching component while held, reducing it to a visual switching option. This interaction is also used in Apple's Exposé, which uses a keystroke to show all open windows, thumbnail-sized and selectable.

Although the visual switching might be more suitable, the setup has the possibility to combine these two and thus combine the extreme immediacy of the keystroke switching with the richness of the visual switchers. This would both benefit the costs to switch, as well as provide a base for handling interruption.

3.1.3 Notification

Notification areas are used for portraying the status of a task or to give some sort of information - more complex than a simple status overview - to the user.

Statuses of tasks are mostly displayed in the system tray or "systray" like seen in figure 3.2. This consists of an array of icons depicting the current status of that task. The Windows operating system limits this information to just an icon, the rest of the systems allow a slightly more complex format. The primary information is still conveyed with an icon, but it can be accompanied by text. The user can use this area to monitor current tasks and attention is mainly attracted by animation of the icon; it does not have a built-in attraction routine like the visual task switchers like OSX's Dock and the task bar from Windows.

A more recent development is the usage of "Widgets" or "Gadgets". These visually rich components are used to convey any kind of information snippet to the user. Examples are the current weather or whether or not your e-mail inbox contains new mails. There is a different approach to these components; Apple uses a full-screen semitransparent overlay, while KDE and Windows 7 keep them on the desktop. Gnome has no such system built-in, but 3rd-parties have already stepped in. Any information can be displayed and this can be a monitor-view of a running application. The earlier mentioned e-mail component is an example of that.

3.2 User-space window managers

Outside of the mainstream platforms – and especially on open source platforms – there are a lot of user-space window managers available. It is hard to put numbers or even rough indications on their categorisations, but it seems fair to say most follow the generic windowing metaphor. More popular examples are XFCE and fluxbox, which are created for the X Windowing system available on most platforms.

These window managers often employ virtual workspaces to extend the desktop area and include extra operations on windows. Notable examples are the option to group windows together by tabbing their window titles in fluxbox or the option to minimise a window to just its title. This gives users the option to structure their desktop a bit more. Another method for grouping is creating constraints for window sizes and location. Badros et al. (2001) created SCWM, which employs simple rules to group windows together; the user has total control over how these constraints are set up.

The major effect of these tools is structuring and de-cluttering the desktop aside from the windows' presence in window switchers. It is therefore debatable how useful these techniques are for incorporating into a switcher. They seem to focus on making it easier to use the desktop itself as the primary source of what options one has to switch between, which reduces the costs to switch; another explanation, in the case of minimising windows to their title, could be it allows the user to "suspend" a task while still being able to see it on the current list of activities.

Not all user-space window managers follow the overlapping window system discussed until now; there are a reasonable amount of tiling window managers available. These managers work by dividing the entire workspace up in sections where windows can be placed: there is often no unused space available when there are open windows. This method thus lowers the amount of effort needed to place windows on the screen since the user does not have to use the mouse to meticulously place windows, which is often further empowered by having keyboard short-cuts for most management functions. All tiling window managers looked at (Awesome, ion3, ratpoison) also implement virtual workspaces and window grouping within a tile. Combined with scripting languages which allow users to program their own interactions, these environments can be very powerful when the user spends time to dive into it.

It is safe to assume these managers grow out of the interest of a single person or small group of people, they implement a need perceived. In some sense, it is the ultimate form of appropriating

ones workspace. Like all appropriation, it can be a matter of taste, but in most cases it seems it is grounded in a need for a certain interaction with windows.

3.3 Mobile platforms

Mobile platforms do not have the computing resources that desktops or laptops have and that has had its influence on the principles for task management. Other hardware influences like screen size and input method both limit the platform in their options, like single-window operation, *but* give it also options "normal" computers do not have, such as orientation changes and direct manipulation.

3.3.1 The illusion of multi-tasking

Because of the limited memory in smart-phones and because computing gravely affects battery life, the idea of multitasking has become different on mobile platforms. Since the limited screen size has forced the vendors to allow interactions with a single window, all other windows have no visual presence while backgrounded.

This allows the vendors to seriously limit the amount of resources used by the backgrounded windows or just close them entirely. Mobile platforms like Apple's iOS and Google's Android implement features to exactly store where the user left a window when he or she switched to another window. This gives the illusion the application was never closed by the operating system when the user reloads the window. By allowing limited background operations such as playing music or using the GPS for location-based operations, this illusion is further strengthened.

This development is not only beneficial for the battery life, but poses interesting implications not only limited to mobile platforms: since the state of an application is saved and background operations are allowed in a structured manner, users can pick up where they left off. If one scales this up to groups of applications, entire tasks could be just backgrounded and recalled without the user noticing or having to prepare the workspace before continuing.

Another interesting consequence is seen on Apple's iOS platform where, because of the greyed definition of a running application, the application switcher actually becomes a jump list for applications based on use. To look at it from Bannon et al.'s perspective: the application switcher gives a secondary view onto the workspace based on usage.

3.3.2 Window management and flow

Because of the single-window management of mobile platforms, moving around within an application with multiple windows becomes less trivial. To make matters more complex, mobile platforms are often tightly integrated and many applications rely on other applications to show content. An example of this is an application which wants to display a geographic location; in these cases an external "map view" is launched using the mapping application.

This system behaviour can be approached in three ways and these methods are found in the various platforms. The first is to view the entire device as a single-level experience: once you left the application, you left the application and you are responsible to reload the previous application. This is often not too labour-intensive, since the application was saved in the state you left it. This pattern is seen in Apple's iOS and is nicknamed by some as "diving in and out single applications".

Another approach is to view opening an application by the user as the principle action; all window switching done after that – be it in the same application or external – is seen as a single path. This allows the user to go all the way back in the application and is seen in Google's Android. All window switches are seen as intents which get stacked, allowing for a cross-application "back-button".

The last approach combines intent stacking with user-initiated window grouping. In this fashion, windows are grouped together by the system because of natural switches, combined with

the option for the user to include other windows. This negates the option to have a clean back path, but enables the user to make their own task groups. This behaviour is seen on HP's WebOS (previously Palm WebOS).

The interesting narrative here is the integration of the platforms and the consequences for window switching. These platforms use the natural progression of the interaction to group windows together in a fashion and because the platforms have semantic knowledge of the nature of the progression, they can make those. It is interesting to see how and if these interactions make it to the desktop, but they would require a large degree of structuring in the operating system.

3.3.3 Notification

Notification and the need to interrupt the user is one of the main purposes which differentiate mobile platforms from desktop systems. Since mobile platforms are communication platforms primarily and are also used as calendars, they have a built-in function to have to interrupt the user from time to time if he or she so wishes. One of the primary categorisations made by WebOS and Android is the type of event: does it need to interrupt the user? A lot of messages can be just caught up with without interruption. The three most common methods for handling events are: modal interruption, notification-bar notifications and a notification area.

The modal interruption is for time-limited matters which require interaction like incoming calls; these are handled with a modal pop-up. Less pressing matters are displayed in the top (Android) or bottom (WebOS) notification bars. These notifications can be accompanied by more intrusive notification methods like sound or vibration, indicating something has happened without the user needing to reply within a set time frame like with incoming calls. The final part is an overview of all notification in a separate area. This can, for example, be accessed by dragging the notification bar down in Android. This gives an overview of all recent notifications as well as other ongoing notifications like calendar appointments or which song is currently playing.

This layered approach again makes use of the fact that the system is given the semantics of the message to convey; what is notable is that it allows for the control over interruption as lobbied for by Czerwinski et al. (2004) and Nagata (2003).

3.4 Academic projects

As long as there have been implementations in the public domain, there also have been window managers based on the theoretical concepts Bannon et al. have suggested and other cognitive principles which could improve the act of switching tasks. The list given below is by no means complete, but each has a specific cornerstone: from hard splits between workspaces to automatic grouping based on use. An overview of the visual outlines of the projects can be found in figure 3.3.

Rooms (1986)

One of the more influential workspace managers was *Rooms* by Henderson Jr and Card (1986). They built on the principles Bannon et al. had set up only a couple of years earlier and tried to match the users' model of how to manage tasks. Because of that, *Rooms* still is a reference point for window management to this day as it is the first implementation of Bannon et al.'s guidelines in a windowing environment in contrast to the text-based environment they were created in.

It was based on one big virtual space with viewports into different workspaces, called rooms. They combined a familiar physical metaphor with a virtual space to make the transition from one workspace to another easier for the user. *Rooms* have windows and doors, but other elements were also used to mimic other guidelines from Bannon et al.

Most modern virtual workspace systems use these mechanics in some way, but *Rooms* had a strong sense of navigating and relating the different workspaces by specialised doors and also

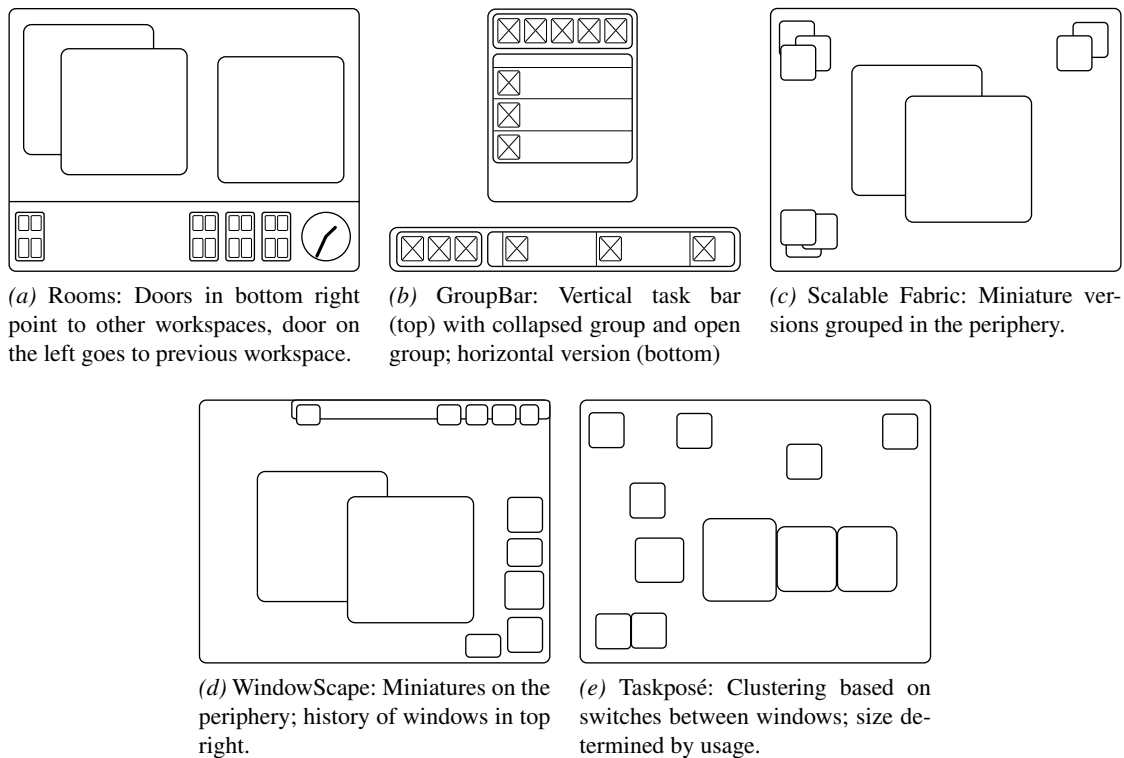


Figure 3.3: Visual outlines of academic projects

the possibility to use windows - often more generic tools - in multiple rooms or even always on-screen. Placement of windows in each room was unique and thus created a true workspace as defined by Bannon et al.

The grouping of windows into workspaces in this case is strict: going to a different workspace means leaving all other window behind. This interaction model is less cluttered and although it means less overview over the entire system, the possibility of placing windows in multiple workspaces independently or keeping windows in all workspaces means the user keeps some control.

GroupBar (2003)

Newer approaches stay closer to the current mechanics of major operating systems. GroupBar by Smith et al. (2003) makes only a simple addition to the task bar of Windows; they add the possibility of grouping windows together. The user is in control of manually grouping windows together whose representations on the task bar are spatially grouped together.

This creates a softer version of a workspace since all the other windows are still visible, but all the windows of the current task get overlaid on them when switching. They comment that this behaviour is probably better than the strict version of workspaces like Rooms used, since now the user can still glimpse at the windows below; a view that is shared by Hutchings and Stasko (2004b). They see people explicitly place windows so they can still monitor the windows below.

This method is thus an alternative method of handling groups of windows; it remains a question *why* people want to do this and can be seen as a failure of the entire system to keep the user updated as easy as just a preference of the user.

Scalable Fabric (2004)

A window manager combining the behaviour with window grouping with putting scaled-down windows on the edges of the screen is Scalable Fabric by Robertson et al. (2004). They let the user group windows in the periphery of the main screen in a scaled fashion so that the actual content of the window can still be extracted. This creates a simple monitor-mode for the application and makes the windows more recognisable. This behaviour is becoming more popular and is the same as Apple's Exposé and Microsoft's windows+tab switchers employ. The main difference is that Scalable Fabric keeps this mode throughout the session and not only shows it on demand.

This approach fits the location of our suggested switcher since it lies in the peripheral vision; it suggests using the *content* of the window as a representation. The interaction used – the grouping on the edges of the screen – is less suitable since our design is based around a single, almost one-dimensional, band above the keyboard which has less space for the groupings than the two-dimensional screen.

WindowScape (2006)

WindowScape by Tashman (2006) uses the same "zooming" mechanic, but leaves the grouping to a timeline, where the user can visually recognise and select a previous state of the workspace. This removes the need for the user to manually group the windows and places the burden of maintaining groups on the system by making timely snapshots. The manager basically creates a history of selected windows with the intent of creating recognisable snapshots and using time and previous use context as anchors for further recognition.

Aside of the similar usage of the window's content as a representation, WindowScape adds another method of viewing the windows: a chronological one. This has similarities with Apple's iOS discussed in section 3.3.1 which uses a jump list for applications used. This method looks promising for an alternative view when looking at application launchers and switchers.

Taskposé (2008)

Scaled down window previews give the user the option to scan and recognise the window based on content similar to WindowScape and Scalable Fabric. These tools do not offer information about relation between the windows however, something that helper application Taskposé (Bernstein et al.; 2008) tries to fix by placing related windows closer to each other, while the size of the representation reflects how often the window is used. Relation is based on how often the user switches directly between two windows. The grouping of windows is thus based on association (system-determined), not classification (user-determined). The real question left by this approach is whether or not system error will make the system annoying to use.

Part II:

Design of a touchable task bar

Now we have a background for what task management is and how direct manipulation or touch input can help the window management experience, we need to take a look at what can actually be improved: what is wrong with your current computer? And, more importantly: what can we do about that using the proposed setup?

After it is clear which areas can be worked on, there is an opportunity to start designing a replacement switcher and launcher. First up is learning lessons from the literature background and their results: what can be altered or redesigned and what should be looked for when creating a "better" switcher?

That is a starting point for creating the actual design of the switcher, but since the design process is based on quick iterations and quick-and-dirty testing, a lot of aspects and fine tuning will actually come from testing, which will be discussed in the final design aspects.

4. What is wrong with my current computer?

This chapter takes a step back and goes back to the literature and observations from current implementations: which aspects of task management are flawed at the moment and relate them back to background. This creates a foundation for creating solutions, which will take up the second section. But to start off: what aspects of our current computers is flawed?

It actually is very hard to answer the question what is wrong with your current computer. This is not a black and white context and it likely never is; a large part of the answer comes from appropriation: most computer users are intimately familiar with how their computers work and have devised their own way of using them. Learning how to use a computer with a screen, keyboard and mouse is by no means an easy task, it is just that almost everybody already walked up that learning curve. The system is usable, but that does not mean it is also optimal and improving task and window management on computers still is a subject open for research.

Before looking into the problem areas, some context is needed about what problems actually are semantically. What is a problem for a user? Or: how does a user look up to his or her system and translate it into meaning for their task at hand? This hooks back into the discussion of appropriation in section 2.1.2 on page 10, where the usage of the system feeds a model of adapting the both the system and the user's habits to optimise the system's use. This process also blurs the lines about what a user perceives as a problem and what a user sees as beneficial. Their previous experiences with systems used for a similar goal will dictate their attitudes towards it.

When wanting to create a "better" experience, one needs to find the underlying mechanics about what the user actually wants. This requires us to take a step back and look at what users actually intend to accomplish with their actions; in other words: what goal and meaning does the behaviour have?

To create a basis for the design, we first touch upon what aspects of task and window management can be improved and – most important – convince ourselves that there is a possibility for improving it. This "categorisation" of problem areas is derived from literature into task and window management behaviours and combines both observations and conclusions from the literature as well as observations while reviewing multiple literature sources. The backbone of the categorisation is what we perceive as the underlying problem for behaviour shown.

4.1 Problem areas

One of those sources is a study done by Hutchings and Stasko where they looked into how people actually manage the space on their screens, emphasising on larger screens (Hutchings and Stasko; 2004a,b). They found out that display properties play a major role in how people manage their windows; physical distances to control widgets like the taskbar and the many repetitive switches mean people are inclined to optimise their methods and probably also do so unconsciously while working with the system. For example, if the display is fairly small, almost all users maximise all their windows and use a taskbar or a keyboard short-cut like alt+tab for switching. The physical space does not allow multiple windows at the same time, so people use either their memory or a representation of windows to switch and since the task bar is never far away, it is easy to use.

Other sources include studies into the use of multiple monitors (Grudin; 2001) and organisation strategies of virtual workspaces (Ringel; 2003). Both Grudin and Ringel start, similarly as Hutchings and Stasko do in 2004, from the assumption people need the extra space to cope with the complex tasks they work with and extract general behaviours from their observations and interviews. Many of these behaviours are based on working with the tools at hand and making the system work within the confined space.

Another aspect of task management is interruption since it forces task switches with the accompanying operations needed. In contrast with the technical context provided by screen and virtual workspaces, interruption is a time-based phenomena which makes the user deal with unexpected situations. Czerwinski et al. (2004) studied the influence of interruption on task management on desktop systems and how people interleave their tasks to cope. Nagata (2003) studies the same subject, albeit in a different environment: mobile platforms; their conclusions and observations are similar though: people struggle with leaving their workspace in a state which they can easily pick up.

When looking at these sources, one can independently see patterns in the behaviours and extract three main problem areas, which will be discussed and explained below: *switching costs*, *resuming costs* and *notification*. The effects of these three areas are quite similar: people *leave more windows open* at any given time and they invest time in *organising the windows themselves*.

4.1.1 Getting around: switching costs

Switching costs are based on how much mental and physical effort it takes to actually make the switch between windows. This is not only influenced by the amount of windows, increasing the mental effort, but it is also influenced by the more physical attributes of our computer setups; one prime example here is the size of the screen; the larger it gets, the larger the distances to travel when using mouse-based switchers. Task bars or other representations suddenly are too far away, so people make sure to minimise distances by taking over the lay-outing of the windows (Hutchings and Stasko; 2004a). They invest time to make the switching costs lower during work by organising the windows beforehand themselves.

The interesting conclusions here are that screen size seems to move the effort from the actual switching on small screens to the preparation and display management on larger screens. It has been concluded that, in our current window-based systems, one should concentrate on how we actually switch between windows when screens are small and how we lay out windows to support task switching on larger screens (Hutchings and Stasko; 2004a). Since larger screens are becoming more and more commonplace, there is still research into how we place and manipulate windows to solve problems arising from large surfaces (Ahlström et al.; 2009; Tak and Cockburn; 2010).

Not all window switchers are based on representations as used by the task bar and switchers like alt+tab which are opened on-demand either do not use the mouse or are centred on the screen, reducing the distances needed by their mouse-based counterparts. There is another factor increasing the actual switching costs however: the amount of windows open. Larger screens, multiple screens and virtual workspaces as studied by Hutchings and Stasko, Grudin and Ringel all share one factor: there is just more space to place your windows. This however adds to the costs of most switchers having larger lists of windows to choose from. So where does this "need" come from and – more importantly – are all those open windows really necessary? This is where suspending and resuming tasks and notification come into play.

4.1.2 Picking up where you left off: resuming costs

One does not always have to switch between windows: you can also finish up and "suspend" the task while you pick up other work that needs to be done. Then, when you want to switch back, you can re-open the windows and resume where you left off. When not using computers this means, for example, one needs to stack those papers together so we know where we were, mark them with something to recognise which task it was and stash it somewhere we can find it, on a corner of the desk or in a drawer. This behaviour is seen more often when the user is interrupted by an external source, as Czerwinski et al. studied. In these situations one has to finish up and move to another task.

Suspending thus costs time and effort; you need to tidy up loose ends, group your work and stash it somewhere safe. On a computer this means you have to finish that sentence, function declaration or other activity, save it and close the related windows. At this moment, your computer has no understanding of what you actually left behind except for the content of the work. Some applications might save your view-port so you have something to come back to; other than that you are on your own.

Resuming your task again therefore is not that easy if it consisted of multiple documents over multiple applications. You need to re-establish all the view-ports and re-layout all the windows. Tools are used for multiple projects and often open in a general "what do you want to open" state, often with quick-links for recent documents. A solution to this is to find the related documents on the hard drive and re-open the task from the documents.

If users know they will be returning to a task, it is however much easier to leave everything open: almost no suspension costs and almost no resuming costs. The down-sides of this behaviour are increased use of computer resources like memory and free space on screen: resources which have increased immensely in recent times on desktop platforms. The penalty of leaving those windows open is thus transferred to making it slightly harder to switch windows, which is something users can work around by, as mentioned previously, taking over lay-outing (Czerwinski et al.; 2004; Grudin; 2001; Hutchings and Stasko; 2004b).

A major source for interruption are notifications received on the computer itself; these are the final problem area.

4.1.3 Keeping updated: notification

Notifications come in all shapes and sizes: from a simple time indicator on the task bar to pop-ups with subject and sender of an email. Not all information we want to keep an eye on is transferred this way: a lot of information is stuck in applications like email clients and terminals. Having this information is important to users: they want to make a decision whether a notification is worthy of interrupting them; they want to have control over when to switch tasks (Nagata; 2003; Czerwinski et al.; 2004).

Keeping those windows open is useful when tasks or activities are related to each other: keeping an eye on the content of the window. The primary example of this is email or other communication tools; these tools are often not just related to one task, but are a separate channel feeding in information about multiple activities. Keeping these windows open ensures one will be kept updated on incoming messages and other state changes.

Applications often use simple mechanisms for displaying these state changes by giving notifications on the window, creating a toast pop-up or adding an icon in a notification area like the system tray (Microsoft Windows) or menu bar (Apple OSX). But, this is often not enough as also seen in studies into multiple monitors: people want to keep an overview of what new messages they got, who they are from and what the subject it. For this, they keep the entire window on the screen somewhere and a secondary screen is often used to house the window (Grudin; 2001).

This feeds further into the number of windows occupying your task switchers and you are not even interested in the entire window, you just want to glance at a part of it.

4.2 Working towards possible solutions

To see what can be improved to alleviate these problem areas, one needs to look at the underlying causes for those areas and come up with solutions. There is a need to argue why people show that behaviour and what can be done about it.

From the behaviours seen by Hutchings and Stasko, Grudin and Ringel we can extract and reason towards a couple of ways to approach the problems. These solutions focus around the grouping of windows, their representation, the location of task manipulation and the method of manipulation.

4.2.1 Window grouping

It is quite obvious from the behaviour people are showing, coping with large numbers of windows can be done by laying out the windows yourself. Often, the windows in these groups are related to each other and are switched between quite often. At the moment window switching and (human) task switching use the same tools since the operating systems do not discriminate between the two, making window switching much harder. One can make good use of that willingness to manage windows by creating a suitable mechanism for grouping.

If windows were grouped, it opens a couple of avenues to improve switching interactions. First off, groups can be used to switch between multiple related windows in one go, removing the need to lay out all the windows one by one or having to re-open all of them. Second, it allows for more operations on the group, such as closing and saving the group in a similar fashion to how mobile platforms handle saving and closing applications. This would hook back in to the suspend and resume problem area, while again reducing the amount of lay-outing needed. Finally, if switching between groups of windows representing *tasks*, this would reduce the amount of open windows.

4.2.2 Location

Another issue with larger screens is the distance to window switchers like the task bar. Creating a switcher on a location with a near-fixed distance from any angle would improve the cost to switch.

While most of the issues like window grouping and monitor-mode adaptations of windows are solvable purely in software, this area hints towards a physical solution; especially elements like the distance to the task bar invite for direct manipulation. On our coordinate-mapped indirectly manipulated screen the distance to the task bar is given by the location of the cursor. The relative distance to the location of our switchers is determined by the size of our screen and the resolution we set it to. For a solution which makes this distance constant is becomes interesting to move away from the screen and to take a look at the keyboard.

Keyboards already have action-specific keys on top – the function keys, which are used for context-dependent and global actions. This location is in the peripheral vision of users and easy to reach. These two aspects suggest two possible uses: being able to monitor changes without it grabbing too much attention due to the location in the periphery and because it is easy to reach, one can use more direct manipulation styles.

4.2.3 Richer window representation

One of the major reasons windows are left open is to keep an eye on them, to monitor the content to some extent. The information given by toast pop-ups and operations on the title bar like flashing or updated information is not enough; people want control over when to react to a notification and to have information available to make those kinds of decisions (Czerwinski et al.; 2004; Nagata; 2003).

It has been suggested in literature to create a "monitor mode" for windows (Hutchings and Stasko; 2004a; Bi and Balakrishnan; 2009) in a way widgets and gadgets are doing this in modern operating systems. A monitor mode would be a concentrated version of a window, similar to WinCuts (Hutchings and Stasko; 2004b), which only displays the relevant portion of a window. For your email program this would be, for example, your unread mail, including sender and subject. Another example is your music player, where the relevant information might only be the current song.

This monitor mode has more space than current representations used on the task bar for the information users want to be able to access without having to have the entire window open. A solution is to combine this with the representation on the switcher and since the location of the switcher is in the peripheral vision of users, it actually sounds like a solid solution. The peripheral

location would provide a suitable mechanism for notification and the location itself – near your fingertips – would make it easy to respond.

4.2.4 Direct and more complex manipulation

Since the suggested location affords use of direct manipulation, it would be insane not to make use of it. A touch screen would allow for more and more complex interactions with the representations of the window. A prime example is to manipulate the window grouping directly; the user can just drag and drop windows together and order them according to their wishes.

Another mechanism for interaction is using more than a single finger for interactions. These interactions are used, amongst others, by Apple on their laptop track-pads; these can recognise gestures with multiple fingers for easy scrolling and zooming.

5. Principles for switcher design

After having seen what aspects of task management can be improved, it is time to further develop those solution areas to design principles. This chapter revolves around bridging the theoretical solutions to more practical principles for design.

The design principles are modelled around the three problem areas identified in the introduction of this part: the cost to switch windows, the costs to suspend and resume and notification.

The four principles used to combat these areas are a combination of the concepts of moving the switcher to the keyboard and the incorporation of concepts from workspaces and notification principles; they are: the location on the keyboard, the use of direct manipulation, the grouping of windows and using rich windows representations.

5.1 Place the switcher on top of the keyboard

Helps with: Reducing the physical cost to switch between windows, providing a non-obtrusive area for notifications.

The location on the top of the keyboard (see figure 5.1) was chosen for multiple reasons; first, being on the keyboard, it is easy to reach. The distance to the touch area is almost constant and is not dependent on the location of your mouse pointer. So, no matter what size your display is, the physical distance your hand needs to travel to switch between windows is relatively constant.

Second, when considering where near the keyboard the switcher fits best, one needs to factor in occlusion and the nature of the switcher. Since it is purposed as an interaction area with dynamic content with the window representations also covering aspects like notification, you need to be able to see the switcher unobstructed. This in contrast to the keys on the keyboard, which do not change. With the location at the top, there is hardly any occlusion from your hands on the switcher while using the system. Your hands are mainly on the keys of the keyboard while using it and thus not obstructing the view on the switcher.

Third, the location at the top of the keyboard means the switcher is in the peripheral vision of the user. The peripheral ranges in your vision allow for subtle notification; the perception of colour is very low, but motion gets detected (Palmer and Rosa; 2006). This creates a nice pallet of options for creating notifications from attention-grabbing to barely-noticeable, which give the user some control over how to be notified.

Finally, when looking at figure 5.1, the configuration removes the function-key row. Clearly this cannot be achieved for the prototype in mind, but from a design principle stance, it does make sense; Function keys are keys which can be programmed differently for different contexts. There are a lot of similarities, for example F1 almost universally displays help, but in principle they are intended for dynamic assignment of functions. On Apple's OSX they are often used for system-wide functions like media play and pause or volume control. Since a lot of these functions can be covered by the touch area – which can signal dynamic assignments much better – it both improves the options for function keys *and* places the switcher closer to the user's hands.

5.2 Use direct and complex manipulation

Helps with: Reducing the operations needed to layout windows, reducing the mental load needed for switching.

Since the switcher is near the hands of the user, direct manipulation and using gestures for interaction become options. Direct manipulation lets the user associate the representation of the

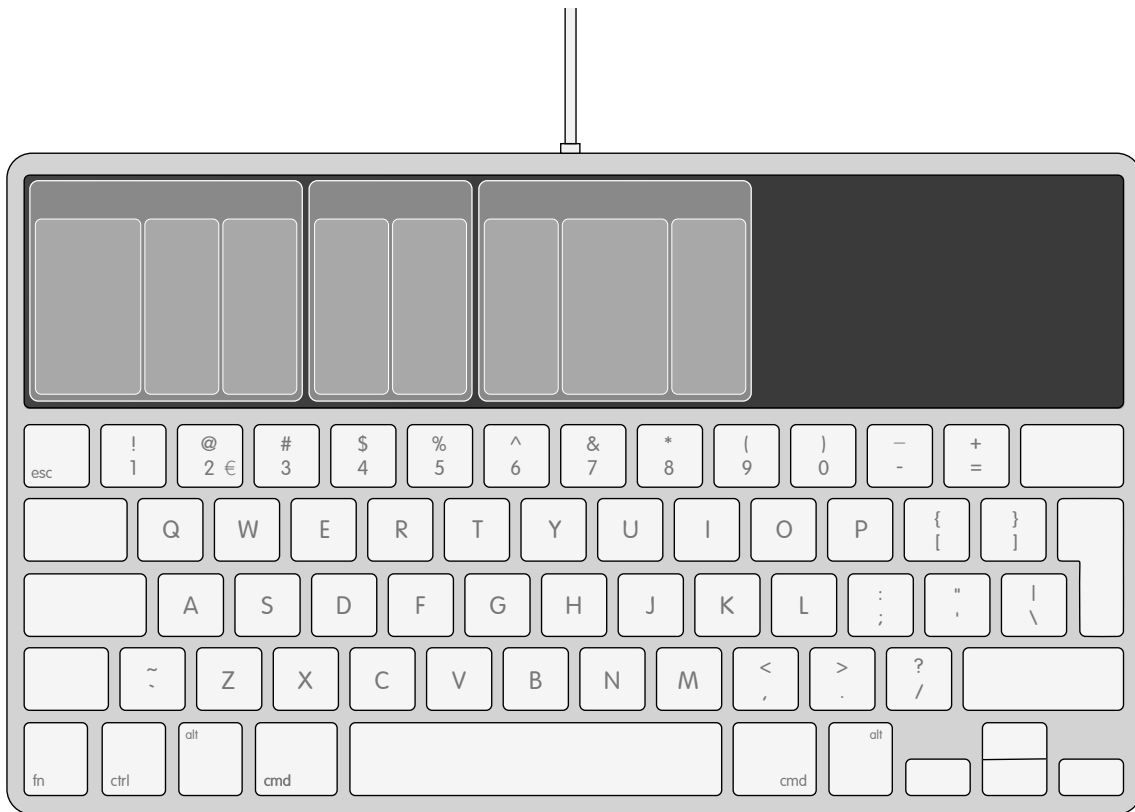


Figure 5.1: Suggested location of the touch screen: within easy reach; based on international Apple keyboard.

window directly with the window on the screen, while the more complex interactions delivered by gestures can be used for simple operations on the window representations themselves. These simple operations are comparable to the media-keys already found on many keyboards these days, such as skipping songs in the media player.

The direct relation between window representation and window also creates a memory map for the user; the user can place windows or tasks in a certain location on the switcher and use motor-memory to refer to it: either by switching to it or interacting with its content. This way, the mental load is reduced when operating them.

Easy access and the possibility to create more complex interactions also open up methods for allowing quick reshuffling or re-ordering of the entire workspace, which would enable the user to quickly get another perspective of the workspace, such as an alphabetical list of all windows regardless of task. Other options include to create interactions for creating a special-purpose modal perspective overlay, for example by selecting multiple windows to tile next to each other so the user can compare their content.

5.3 Make windows groupable into tasks

Helps with: Reducing the number of operations needed to layout windows, reducing the costs to suspend and resume, reducing the amount of open windows.

In the context of window switchers, groups are no new idea; they fit in nicely with the concept of workspaces by Bannon et al. Because of that, a lot of implementations already use the idea, albeit they use it in different forms.

Rooms (Henderson Jr and Card; 1986) uses grouping in the true workspace fashion, while GroupBar (Smith et al.; 2003) uses grouping in a more direct way by allowing the user to create

groups of windows on the task bar. This idea aligns with both the location of our switcher and the direct manipulation. It provides a low-threshold interaction of organising and configuring the groups. The resulting idea is shown on the switcher in figure 5.1, where groups of windows are shown.

Grouping windows into tasks is a direct implementation of Bannon et al.'s guidelines to create "functional groupings of activities". The combination of having easy manipulation allowing for complex interactions due to the multi-touch panel allows for interactions which can also be used for suspending and resuming entire tasks in one go.

A guideline which is harder to follow in this context is *interdependency*, the ability to link windows to multiple workspaces or groups; although this can be accomplished by simply incorporating the window in multiple groups, but this might quickly become confusing for users. Because grouping is already an extra layer for the user and interdependency is a concept built on top of it, we decided to keep this out of the design for now.

5.4 Use rich window representations for notification and simple interactions

Helps with: Reducing the operations needed to layout windows, reducing the amount of open windows, keeping the user notified of events.

The contents of the windows should be rich in information, but it should limit the information to the state of the application. In other words: only information the user is interested in should be visible. This keeps the clutter low and keeps the window representation easy to glance at. Besides that, it offers the user a window for staying updated and – in combination with notification – can be used to give the user precisely the information needed to make decisions about that notification: the control over when to act or when to ignore notification.

The window representations are in the peripheral vision of the user and can make use of the benefits of being in that part of vision; it means notifications can be made subtler. Representations can make use of motion in their notification, since the peripheral vision is primed for it, but can also use colour and shape to highlight items which the user can pick up when glancing at the representation. This as well fits in with the idea of giving the user control over the importance of a notification.

It is, however, very easy to fall for ideas which make complex use of the multi-touch panel at your fingertips. The philosophy used for the switcher is that it is a window switcher which makes use of its location for easy configuration and subtle notification, not for complex interactions: it is the task of the window on the screen to handle complex matters and the representation should – first and foremost – focus on being a recognisable representation of the window, followed by being a platform for notification. Finally, of course, there is also room for simple, atomic, interactions like skipping a song or pausing.

This philosophy has partly ergonomic roots, because it should not be the goal to engage the user in interactions at his or her keyboard which require them to look at their keyboard for longer periods of time. The current setup of screen and keyboard, which allows the user to take a relatively neutral posture by looking straight at their screens and therefore causes less tension in for example the neck, is not one we want to change. The possible actions on the representations should stick to extremely simple interactions, which can ideally be executed without much attention focused on the representation on the switcher.

6. Prototype design

Based on the design principles a prototype was created; the principles by no means define a finished design implementation. In this chapter, the actual design gets discussed: the final step from theoretical problems and their solutions to a prototype. This prototype was built for an Apple iPad (first generation) and an Apple OSX desktop system. An example session can be seen in figure 6.1.

This prototype was created from the design principles set, but concentrated on the main "lower-level" aspects of the design: the ability to switch and interact with the implementation. After this, extra aspects of the design were added, such as group configuration and more complex interactions with window representations. The prototype was developed in short development intervals with a "quick and dirty" testing cycle after the implementation or improvement of each feature.

The prototype's design revolves around two major elements: the *window representations* and the groups – or the more appropriately named: *tasks* – they are put in. Finally, a description of the technical background is provided.

6.1 Window design

The windows are the most used part of the design; they are used for both switching between windows as well as conveying the state of the contents of the window on the main screen. Window representations only have one common element: the name of the underlying application is displayed in a relatively small font in the bottom left of the representation, as seen in figure 6.2. The rest of the space is reserved for conveying the state of the window in a matter which makes it distinguishable from other window representations.

6.1.1 Size and placement

Because of the limited usable surface of the iPad's screen, sizes for the windows are variable. The size is relative to the content and interactions used by the representation of the window. So, for example, the representation of media-player iTunes uses the artwork of the current song and uses that space for simple "go to next/previous song" interactions, making the window-size on the switcher relatively large.

For the prototype, the representation for the media player iTunes is the only representation with interactions and during many of the iterative design steps, the size used by the artwork on the representation turned out to be very beneficial for the accuracy of the interactions. Since the touch-panel has no haptic feedback for representation boundaries, the extra space makes it possible to perform the gestures on the representation without having to divert much attention towards getting the starting position of the fingers correct.

Since switching between windows is the primary use of the representations, they are placed as close to the bottom edge – and the user's hands – as possible. This placement should keep misses to a minimum, as there are no other interaction elements nearby. Any other interactions with the switcher in its entirety are handled with interaction elements above the window representations and are likely to require the user to shift their attention to their hands to see what they are doing. This layout was previously shown in figure 5.1 on page 32.



Figure 6.1: Switcher running on the iPad

6.1.2 Interaction

As part of the design philosophy, interactions on representations should be kept simple and minimal since it should be the task of the window to handle complex interactions and not the purpose of its representation. Interactions based on gesture need space to overcome the lack of haptic feedback on the touch panel and even the most basic and important operation on a representation has that "problem": activating a window, the core of the switcher.

To overcome the lack of space on the switcher, primary and often-used actions are operated by single-finger gestures since one finger takes less space than two, making it easier to do unattended. Primary operations thus are: switching between windows and configuring the order and compositions of window representations and tasks. Secondary operations – basically the custom interactions with the window representations – are done with two-finger gestures.

6.1.3 Implemented representations

Representations of windows are powered by widget-like visualisations; they convey the state of the window and optionally accept simple interactions or grab the user's attention for sake of notification. For the prototype, a couple of example representations were created to test the principles of their design, which are shown in figure 6.2. These are: Adium (instant messaging), Mail.app (email), Safari (browser) and iTunes (media player).

Instant messaging

Instant messaging client Adium is built up around a contact list windows and one or more chat windows. The window representation designed shows the current open chats – regardless of in which window they are – with their name, contact image and whether or not they have unread messages. Unread messages are displayed in a contrasting orange colour to make discovery while glancing easy.

In the original design, when unread messages arrive, the representation uses a small motion-based animation to convey the arrival of the message; this animation stops quite quickly and just leaves the chat highlighted. The rationale behind this is that the user gets a short primed notification of change in their peripheral vision, after which colour is used to grab attention when the user glances at the switcher, which is not really attention-grabbing in the peripheral vision. For this prototype, this motion notification was disabled to test its visibility when focused on the main screen.

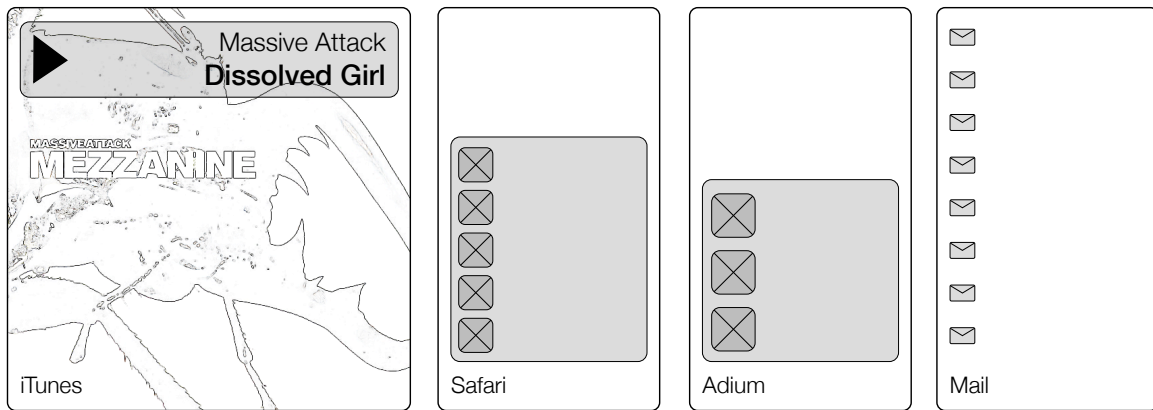


Figure 6.2: Window representations for (from left to right): Music (iTunes), Browser (Safari), Instant messaging (Adium) and Mail.

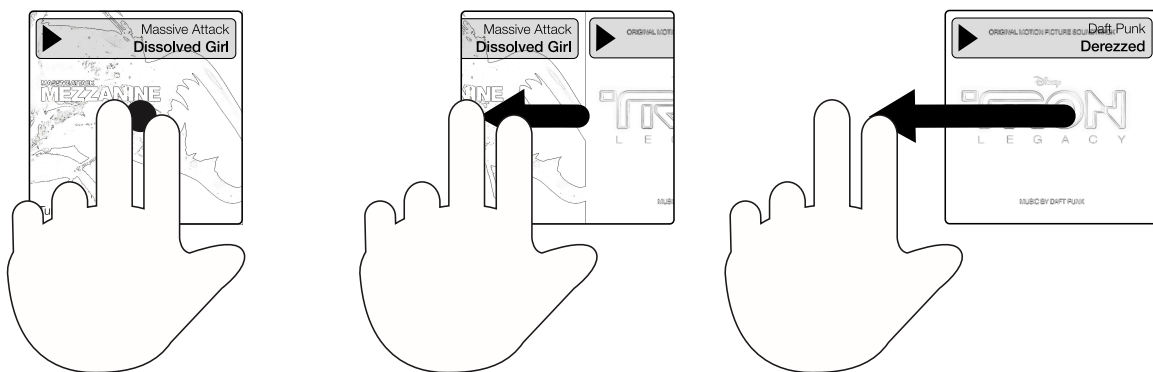


Figure 6.3: Interaction for music window: dragging to select the next song

Mail

The mail representation is based on the representation of instant messaging, which is discussed above. Mail is different from instant messaging as it is a continuous stream of messages with sender and subject, which are grouped by subject and not sender. For the time being, a simple list of messages is shown, using the same principles for notification as with instant messaging. Simple envelop icons are used to give a better overview when glancing about the number of messages shown.

Browser

The browser representation is simpler in nature because browsers have less intrinsic notification. The initial purpose of the representation was to also show current downloads, but because needed functions lack in the programming interface to browsers via AppleScript, this was not achievable.

The representation shows for every open window which tabs are opened and which one is currently active. Each tab is represented by the icon from the page opened as well as its title. The background of the entire representation has the same colour tint as the page of the active tab.

Music

The music-widget, based on iTunes, is in nature a very simple representation: it shows the current playing song with title, artist and album art. If nothing is playing, a place-holder is shown. The entire representation uses the album art as the background with the song information overlaid in the top third of the area.

The representation uses motion to notify the user a song has finished and the next one is playing. This is done by treating the representation as a view-port into the current play list, with all the songs laid out from left to right. When the song changes, this canvas shifts to the correct song. This way the user gets a subtle notification in the lower edge of their vision something has changed; a simple glance gives away the current song.

This mechanism is also used for the interaction with the representation; the user can skip forwards and backwards by flicking the canvas back and forth with two fingers in the same manner as the animation switches between songs, as can be seen in figure 6.3. This is also the reason the song information is in the top part of the canvas, so the user can still see the information without really occluding it. Pause and play are implemented by a simple two-finger tap.

6.2 Task design

Tasks represent the grouping aspect of the switcher; they provide visual contrast between the different groups and are completely configurable by the user.

Tasks can have names which the user can set themselves. The label on the task with the name is used as the handle for spatial configuration. This label is placed on the top of the task, a location which will cause the task be occluded by the user's hand while being dragged. The occlusion down-side will allow the window representations to be closer to the bottom, which will be used more often. Spatial configuration of tasks will not be used that often and the operation is less sensitive than windows: the area for a task is rather large, so the need to see what one is dragging is less.

Although the design calls for options for resuming and suspending tasks, these were not implemented for the prototype due to their technical difficulty to implement: no API exists in current stable versions of OSX and "faking" the interaction turned out more complex than possible in light of the scope of the prototype development time line.

6.2.1 Mechanics of window grouping

All operations for the grouping are primary, so they require only a single finger. When a user drags a window representation a proxy follows the finger directly, while the original window shows where the representation will end up if the user stops dragging. Due to the multi-touch screen, multiple windows can be moved simultaneously.

New groups can be started by dragging a window representation into an empty space on the switcher. The switcher will signal the operation by changing the border of the proxy, so the user is certain a new task will be started.

6.3 Technical background

Because of the limited resources available, the prototype was built on an Apple iPad 3G (first generation). The switcher is implemented in HTML5¹, running in the built-in Mobile Safari browser of iOS 4.3. A lot of features of the – still-developing – CSS3 styling specification were used to give the switcher at least some professional look without having to spend much time on those elements of the design, leaving the bulk of time for the interactions.

Communication with the target computer is done via the network connection of the iPad. Commands are sent back and forth as JSON² objects from the JavaScript embedded in the HTML page via the HTML5 WebSocket function to a small web-server on the target PC implemented in Python. This web-server hosts the source files for the switcher interface as well as the logic

¹See <http://dev.w3.org/html5/spec/>

²JavaScript Object Notation, see <http://www.json.org/>

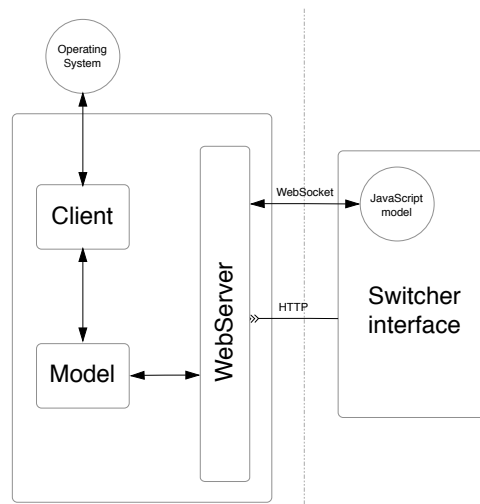


Figure 6.4: Prototype model: on the left the host application running on the target computer; to the right the web application on the iPad

needed to extract information and perform actions on the target computer. Loading the switcher thus is just like loading a conventional web-page.

Because of lack of a comprehensive push-based programming interface for switching windows, the Python application scrapes the information from the target computer by pulling the current open windows and their state. It was implemented for the Apple OSX platform, using the usability APIs for window discovery and the interfaces provided by applications through AppleScript for gathering in-depth data about windows and the underlying application. This approach is, by far, not suitable for stable development, but works just fine for a prototype.

Internal state is kept both in the web-application in the browser as well as in the Python application with a platform-agnostic model, making it possible to add new platforms. This model can be seen in figure 6.4.

6.3.1 Touch operation limitations

Since the iPad's screen is multi-touch enabled, the embedded touch-events framework in Mobile Safari makes it possible to capture gestures and work on multiple window representations on the same time. Due to the prototype glitching and losing touches when working on multiple aspects at the same time, the input was limited to a single window representation. This is by no means a limit of the switcher design or the capabilities of a multi-touch input device, but of the prototype not responding correctly yet to the events all the time. This limitation was put in to minimise confusion with the participants while operating it and touching the screen with other parts of their hands than their fingertips.

Part III: Putting the design to the test

Since most of the features of the design are grounded in literature and theory, combining those features on a single prototype requires some verification to see if they work as hypothesised. For this, a small study with users was performed on the prototype. Chapter 7 describes the study as performed, while chapter 8 walks through the results it yielded.

7. Study design

To test the prototype discussed in the previous part a study was constructed, which design gets detailed in this chapter.

The study was set up as an exploratory small-scale test consisting of three parts: a qualitative contextual inquiry to establish a base line of the participants' switching behaviours, an exploratory acclimatisation test to get familiar with the design and finally a test to see whether or not the hypotheses about the features were correct. Because of the complex and varied nature of the problems, a qualitative approach was chosen for all parts, since with such a rough prototype it is hard to know whether you are measuring the target metric or just the speed with which the user adapts to the new interface.

These three parts were conducted and recorded in a single session of about an hour and were held at the location where the participant works. The study was performed with *five* participants of which four were male; two software developers and three information (administration) workers. They are introduced in the last section of this chapter.

The language used during the studies was Dutch, which is the native language for all people involved, to minimise problems while expressing themselves.

7.1 Participants

The studies were conducted with *five participants*; these five will be individually discussed in the results, but will be referred to anonymously with fictional names. They are:

Alice is our only woman and is in her mid-fifties. She works from home in an administrative role for a substantial hobby; this hobby requires her to be on location most weekends. Her computer setup is based around a 17" laptop with Microsoft Windows 7 which is connected to a docking station when at home, making her desktop setup a dual-monitor setup with a 24" external wide-screen. She has worked with computers since the early nineties, from DOS and Windows 3.1 up to Windows 7.

Bob is also in his mid-fifties and works for an insurance company where he deals with the financial administration of the IT department. He occasionally works from home, where he has a docking station with a conventional (4:3) 19" LCD screen while his laptop is closed. This laptop issued by his work runs Microsoft's Windows XP. Previously he worked in various roles in the IT department since the late eighties, always in an administrative or management role.

Charlie is in his mid-to-late twenties and has a master of science in computer sciences. He currently works as a freelance developer from home, both on web-design as well as more technical projects. His setup revolves around a single 12" laptop with an external keyboard and tablet. He runs Linux on it with the Awesome ¹ window manager, which is a tiling window manager with virtual workspaces.

Dave is an IT student and in his early-to-mid twenties. Besides his lectures and other work as a student, he works as a system administrator for which he develops. In his free time he also does some graphical work. His setup consists of a 15" Apple MacBook Pro which is docked to a 24" wide-screen LCD monitor with an external keyboard and tablet. He runs the latest Apple OSX Lion Developer Preview on his system.

¹See <http://awesome.naquadah.org/>



(a) Using the prototype with the laptop



(b) Using an external screen

Figure 7.1: Study setups

Evan is our final participant and is in his mid-twenties. He works at a public broadcasting organisation where he works as an editor. At his work, he works with multiple computers and his principal computer has a dual-screen setup running Windows. At home, he uses a MacBook Pro with an external 24" wide-screen monitor running OSX Snow Leopard.

These five people were chosen for their apparent similarities and dissimilarities. For example, both major platforms OSX and Windows are represented, but also one Linux-user; furthermore, an age difference was included to test adaptability: most younger participants might be familiar with touch-based systems, while elder people might not. Finally, both a dual-screen setup and a single small-screen setup were included, which might give some insight into differences of size and multiple-monitor use.

All participants either work from home or have the capability to work from home at least partially; they were all interviewed at home, which makes comments on their work environment coloured by the interpretation of the participant. These comments were recorded and considered while creating the results, but were set in proper context.

7.2 Prototype study setup

Since only support for Apple OSX was built into the prototype, the test was done on a separate and pre-prepared system running on an Apple Macbook Pro with an external keyboard. The iPad used for the actual prototype was positioned on top of the internal keyboard, as close as possible to the top of the external keyboard.

A camera recorded all interactions with the keyboard, mouse and prototype, as can be seen in figures 7.1a and 7.1b. The camera was not used during the interviews and contextual inquiry, where only audio was recorded for later reference.

7.2.1 Part 1: Getting a baseline

Because window switching on a computer is a varied and heavily appropriated task, a base line was needed to place observations made during the tests in perspective. There is no guarantee this covers all possible behaviours seen in the test, but it improves the chance a correct assessment is made when concluding from the tests. For this, the participant was given a couple of example situations or instructions and was asked to work accordingly on their own computer while being observed.

The discussion following this initial request grew organically as the participant got to work. The main subjects steered upon were *what kind of applications were used, what conscious appropriations for window management were made to the workspace by the participant and the nature and quantity of the tasks they performed*. While the participants used their computer and answered questions by performing them in their own working environment, behaviours about the methods of switching windows, the layout of windows and interactions while switching between higher-level tasks were recorded. Participants were asked after the fact to explain seen behaviour and whether they thought there was a conscious choice of method.

Participants were asked to be verbose about their actions as if they were explaining them to a new colleague and the conversation was kept active to minimise the feeling of being observed and tested.

7.2.2 Part 2: Acclimatisation

The majority of the participants (three of five) were not accustomed to the OSX operating system and did not own touch-capable systems such as smart-phones. To increase the odds of actually testing what we intended to test, the first few tasks were purposely designed to get familiar with both the operating system as well as the prototype.

Participants were at all times allowed to ask questions on how to operate the computer, but were not encouraged to ask questions about the prototype. They were asked questions based on information both available and not available on the prototype, forcing them to switch windows. Participants were encouraged to make use of the prototype if they felt like it could perform the actions asked.

7.2.3 Part 3: Main tests

The main part of the study is built around testing the core features of the prototype and their effects. These effects are the ease of access due to the *location*, the effectiveness of the notifications on *richer window representations*, the ease of *grouping windows* and the ease of use of the gestures designed for *complex manipulation*.

Ease of access

During the course of the tasks, participants often have to switch between windows. They were encouraged to use the prototype when they felt like they could use it, but were not punished for using their own methods.

Although the prototype setup has a larger space between the tips of your fingers and the main interaction area because of the edge of the iPad and the function key row on the keyboard, the distance was still relatively small.

Grouping

The ease of grouping windows and their use was tested mainly with a configuration task, but re-grouping could be done at any time. No interactions with groups of windows were possible except for re-grouping the groups, while the original design calls for interactions to suspend and resume groups (tasks). This was left out of the test due to implementation difficulties.

Notification

Since peripheral vision is good at detecting movement, but less suited for detecting colours and shapes, motion was used for notifications that the user preferred to be aware of when it occurs. Colour and shape are however suitable for highlighting changes when the user glances at the prototype.

In the prototype motion was used when the currently played song changed and colour was used for incoming (unread) mails and chats. This was done to check both types and check when stimuli were detected. Both events occur several times during the course of the tasks. Participants were asked to mention anything they notice, without giving specifics about what to look for.

Gestures

Besides the "gestures" for switching and grouping the windows with one finger, two-finger interactions were added for the iTunes representation: playing/pausing the music and changing to the next or previous song. Participants who did not find this interaction on their own were given incremental information to see how fast they figured it out. This increments consisted of:

- The existence of the functions
- Reminding the participant of the song change notification
- Mentioning interactions they tried were assigned to other operations (switching, grouping)

8. Results

The results encompass a view back from the study to the design and tries to answer the knowledge question *does the design work?* in form of the prototype.

But since the study design contained two different parts: getting the baseline to be used as a context for reasoning and actually testing the prototype, there are more than just results regarding the prototype; we can start with an insight into the question asked earlier: *what is wrong with task management on computers?*

8.1 Validity of problem areas

Interviewing and observing the participants for the base line provided an opportunity to take a qualitative measure of the validity of the problem areas; whenever there was a specific behaviour expressed or observed during the interview, a short follow-up was done to try to get a sense of the underlying motivation to do that. For example, when Evan booted his computer and automatically started three applications, a discussion was started about the how and why, which turned out to be related to the cost of suspending and resuming: the applications were used so often that saving the costs of closing and restarting outweighed the costs of switching with more windows open.

Some other behaviours were quite obvious and spoke for themselves, such as Dave using tabs in every application possible to limit the amount of windows, while others required more observation, such as Charlie never splitting a screen using the tiled window manager because the interaction with the windows lacked keyboard-control.

The five participants were all quite different in their approach of switching and managing windows, but all had behaviour-overlap with at least one other participant; they ranged from purely keyboard-orientated (Charlie and Dave) to purely mouse-orientated (Alice, Bob, Evan) and from all windows maximised (Bob, Charlie) to just a patch of windows which get sized on-demand (Alice, Dave, Evan).

All participants expressed and showed they appropriated their behaviours to the capabilities of their operating system and, while this is debatable for a Linux window manager which almost forces you to customise, none of them used external tools for switching and launching applications: all was done with the tools at hand provided by the window manager. Dave had had positive previous experience with third-party tools, but had not really had any major motivation to install them on his new developer preview version of Apple OSX.

When looking at the appropriations made, all three problem areas were evidently cause for behaviour; because of their unique way of using their computer, all participant dealt differently with the symptoms.

8.1.1 Switching costs

Keeping switching and switchers manageable was a recurring and contested subject; like mentioned above there are situations where keeping windows open can be desired. Alice, Bob and Evan – the mouse users – kept the overall amount of windows low when possible, whereas the keyboard users Charlie and Dave used the varied commands which hot-keys can provide to circumvent the problem.

Charlie and Dave both used the multi-document nature of the applications they use and the key combinations to switch between them. What is interesting is that they both employed a form of tree structure for structuring the windows; Dave used the tab-interface provided by his terminal application to subdivide the windows in his code editor, while Charlie used a similar strategy with virtual terminals (screens). The difference here is that Dave has the benefit of having a visual cue

of the amount of tabs open, while Charlie has to rely on his memory. They are practically creating the layered switching structure: the application uses certain hot-keys to switch documents, while window switching uses other hot-keys and switching entire workspaces again another; this approach brings the complexity of switching down from linear $O(n)$ to logarithmic $O(\log(n))$.

The mouse-oriented participants have to limit the amount of options (windows) open or use the direct-click window grouping as previously discussed in section 4.1.1. Alice, Bob and Evan had different "cut-off" points – the point at which they closed the window – for their windows. Bob is by far the most conservative, which he relates to his long history with computers: he structures his work so he can quickly close the window again, which frees up the resources taken by it. This forced structure is seen with Alice as well, albeit in a less strict form; Evan finally just keeps the windows open until he feels he is done with them. The older Alice and Bob both strongly feel they need to free up resources although they are aware their computer can probably handle it; the question remains whether they structured their work because of how they perceive the effect on computer resources or are just structured themselves.

This difference in these two approaches is also mirrored by their settings for other window switchers: the keyboard-centric participants Charlie and Dave hid all task bars since they knew the key combinations to switch, while mouse-centric participants made sure they were visible so they could click them.

8.1.2 Suspend and resume

All participants were, like mentioned previously, at best hesitant to close windows they were not sure they were done with: if there was any argument for keeping it open, they kept them open. They all felt picking up windows was complicated and required more work than they would like. What did differ was their awareness and structuring of their tasks: some were structured and worked sequential and reasoned they were done with the task, while others jumped tasks during the day so they kept the windows open. One variation of keeping the windows open used by Bob was to minimise all unused windows to remove the clutter, but keep the windows easily accessible.

When asked why the specific participants, Alice and Bob, worked sequentially, they actually answered this was based on a hesitance to keep many windows open mainly due to cluttering their task bar, but also based on a long-standing habit to not open too much due to the lack of resources in the past.

8.1.3 Notification

Notification and interruption bled through appropriations: Bob left emails open with the current task as reminders when interrupted, Evan placed applications like Twitter and IM partly visible to monitor them. All participants expressed the need to get more than just "a new email has arrived" to make decisions on how to respond without being specifically asked about it. Toast pop-ups from Outlook and the OSX notification helper Growl¹ do provide this information and are seen as useful, but had the problem of being volatile by disappearing after a short amount of time. To get updated, all participants switched to their email or other communication clients to check for events that happened.

On the other end of the spectrum, Charlie used no notification at all except for a simple "there are changes". He wove in checking those applications – which were in a completely different workspace – often and decided quickly to stay or change back. This was made easier by his keyboard-centric setup: the switching was cheap, so he could do so.

An interesting difference is using sound for notification; this was only seen with Alice. The other users, with more mobile laptop setups, did not have sound set up, probably due to the fact

¹See <http://www.growl.info>

they did not want to annoy others in more public spaces. Sound was used because, as it turned out, she did not notice changes in her peripheral vision that much on her dual-monitor setup. She expressed no desire to see who sent the email or have any other control about the notification, but this could just as well be that she was used to her notification setup.

8.2 Effect of solutions

The observations strongly follow the problem areas constructed from literature, which strengthened – in our eyes – their validity. To see whether the prototype actually succeeds in solving these, is a different story.

The main purpose for the studies indeed was to see whether the hypothesised effects of the solutions suggested actually work. During all operations on the prototype, the direct and indirect interactions of the participants were observed and an attempt at following their train of thought was made. The results reflect the hypothesised effects, which is really encouraging, but also other related effects popped up, which get detailed in their respective sections.

First of all, one of the most widespread reactions to the prototype was one of exploration; participants enjoyed the touch-based device because of the direct manipulation: almost every action has an effect on the prototype. This caused the studies to evolve and change organically since, almost resembling small children taking over any task you put in front of them, they took command of what was happening and the experiment leader was left more with course corrections than steering.

8.2.1 Notification

Related design principles: *Location* (5.1), *Window representations* (5.4)

Notification consisted of two aspects mainly: motion and colour. Motion was tested with the iTunes representation which used a scrolling effect to signal which song was now playing; colour was used to signal new messages in both the email client and the instant message client. The overlapping question is whether this can provide notification both to attract and to inform.

The results strongly reflect the hypothesis that motion is noticeable in peripheral vision, while changes in shape and colour are not or barely noticed. Participants often noticed the song scroll or could recall what happened when asked afterwards. It must be mentioned that the actual auditory cue of the song changing might have triggered a subconscious preparedness for the animation, but since the participants were often in discussion or telling what they were doing and did not know to expect it, we are confident they actually noticed the notification.

Colour was not perceived while focused on the main screen, which was also expected. Only Charlie noticed the orange colour of a new message in the Instant Message client, but was drawn there by the appearance of that block of information since it was not previously available. After that appearance – motion – caught his attention, he quickly discovered what was happening. The other participants noticed the colour when glancing on the prototype for unrelated action, such as glancing where a certain window was located.

This together makes the design allows for the pallet of notification options both attention-drawing with the motion and low discoverable with colour.

A significant point must be made for accessibility however; the older Alice and Bob had more problems detecting motion in their peripheral vision and Alice actually only picked up the motion after knowing about the notification and probably diverting some unconscious attention to it. Both used reading glasses while operating computers, blurring their peripheral vision around the keyboard.

This conclusion could also explain the behaviour of Alice to rely on sound or relying less on toast pop-ups and other notifications which make use of the peripheral vision. It is however not

possible to find out if that notification setup was chosen because the notifications in the peripheral vision did not work, or that the setup grew organically. She did not even know why she did it.

8.2.2 Grouping

Related design principles: *Grouping* (5.3)

Grouping was very natural to all participants, without exceptions. Due to the limited interactions involving grouping created with the prototype, participants had some questions about the reasoning behind it, a fact which should be emphasised and picked up when doing any further work with the design. In case of Evan, he stopped to ask what the grouping did, since he wanted to factor in that when he created the groups. Other participants chose a semantic for the grouping, but were confused because they lacked a primary reason or effect: what does it mean when I group these windows together and – more importantly – what can I do with them? They want to base their choice of which window should go with another on the effects of grouping them together.

When participants started grouping windows together, a couple of possible improvements to the implementations came to light: the option to place tasks at specific locations without them snapping back and the option to create new groups to the left. Both features were included in the design to create, but abandoned because of implementation difficulties, and since these interactions were assumed natural by all participants, they should be explored in further studies and implementations.

Grouping created a lively discussion after the tests about their expectations for window grouping and the discussion was used as a sounding board for ideas given in the original design such as the possibility to suspend and resume entire groups. The suggestion of using groups as strict virtual workspaces – with all other windows disappearing when switching – was met with mixed responses, since some participants were used to having windows in the background for monitoring. This behaviour should be negated however by the richer window representations – which the participant might have forgotten – but further development and testing should be done to confirm this.

This subject is the most expressive example of meaning in the interface and what is perceived as a problem, which is further aided by the fact the design allows for new ways of gathering information via the representations, ways which are not yet incorporated into the mental model of the participants. Their preference is still based on their *current* and *compound* view of the entire problem of task management.

8.2.3 Ease of access and gestures

Related design principles: *Location* (5.1), *Manipulation* (5.2), *Window representations* (5.4)

The main two effects of the manipulation options offered by the location and the multi-touch panel were the capability to group windows together easily and to perform more complex manipulations using (optionally) multiple fingers or manipulating multiple objects at the same time.

As discussed above, the gestures for dragging windows together and managing groups were simple enough to use without any instruction and were often attempted without any instruction or task. This was expected, since they were operated by basic single-finger operations and literally are the first things one tries. Directly after being given a configuration task – "*group this set of windows anyway you like*" – and understanding what to do, participants started dragging and were done in mere seconds.

This natural instinct of what to do did not transfer to the two-finger interactions designed for the interactions with the content of the representations. The participants all struggled – independent of their knowledge of touch systems – with what to do to change the currently playing song. When performing this task, three groups of problem solving were identifiable: familiar with similar

touch-based systems, knowledgeable about touch-system's capabilities and having no previous experience or knowledge about them.

Dave and Evan are both smart-phone users and both own an Apple laptop with a multi-touch track-pad. This made them familiar with using multiple fingers for advanced gestures, since OSX uses 2-finger interactions for scrolling. After discovering two fingers should be used and taking a second to reason the sought-after gesture, they concluded and tested the correct gesture.

Evan however, suffered the same handicap as the knowledgeable Charlie: before trying the "logical" correct gesture, they went through the gestures they knew to exist for 2-finger gestures, namely pinching and rotating. This is where Evan's familiarity with the track-pad kicked in and he transferred to the correct gesture, while Charlie was left to ponder and test gestures.

Finally, Alice and Bob had no experience or knowledge of touch-based systems and they struggled the most; Bob only picked up when all the hints were given, while Alice gave up before finishing. Both expressed the gesture was foreign to them, but felt comfortable using the gesture afterwards.

These results are encouraging in the fact all participants agreed on the choice of gesture, when reasoning back to the nature of the 2-finger operation; due to the way the hints were layered, they discovered on their own what the differences between them meant. This indicates that, although the participants had their struggles finding out what to do, the progression from beginner to expert is probably short.

8.3 The perceived usefulness

One of the major questions during the interview after the tests was about fitting this prototype into their own way of working. This question was meant to force participants to review their own methods in contrast with the prototype. As expected and predicted in the discussions about appropriation in section 2.1.2 and in the introduction of chapter 4, most had problems seeing it fitting in. Arguments given were almost always given from the perspective of their habits – the habits they had learned themselves to cope with the system – and not from their probable needs. They were able to put it in perspective at the end of the interview after a discussion about their intents what meaning the activities had, but the effort required to change their habits weighed heavily.

Participants all saw the benefits of elements of the prototype however; they were most interested in the notification options the design provided which often relieved problems they saw with their current setup. The best example is Evan who kept parts of windows in the corners of his screen to monitor them. Besides that, they enjoyed the ease of access and realised it was a quick way to switch, but the unfamiliarity with the semantics of the grouping, they were reluctant to show clear opinions about it.

Due to the fuzziness of use and probably also due to the unknown structure, that window grouping was least favoured. Mostly, this dislike was based on arguments about how their applications are currently set up. For example, Dave expressed his programming environment already solved that problem with tabs and was more flexible than window grouping, an argument shared with others. This further strengthens the need to improve how window grouping can be used and provide better integration with applications to make them aware of the external grouping.

A final – and not previously designed for – method for using the entire concept is for window lay-outing itself. Both Charlie, using a tiled window manager, as others using a windowed manager saw possibilities to use the easy access and manipulation of window representations as quick re-lay-outing and tiling tools to setup their window locations. This feature is similar to what Smith et al. offer to tile groups of windows in GroupBar.

Part IV: Wrapping up

At the end, after the design, implementation and test of the prototype, there is room to reflect back on issues which surfaces during the project and for a final summary.

9. Discussion

Matters are hardly ever perfect and fair, no matter how promising the results actually were; during all phases, from design to user studies, issues were discovered and some were not addressed in the prototype or even the design. Other discussion points came up during the debriefing of the user studies and give a direction for future work, but first will be elaborated in the next sections.

9.1 The semantics of window grouping

One of the primary discussions, as mentioned in the results in chapter 8, is about the semantic implications of window grouping. The prototype did not implement any predefined semantic, but the results showed it influences the use of the switcher. The suggestions and designs on the table are based on the implementations used in Rooms (Henderson Jr and Card; 1986) and GroupBar (Smith et al.; 2003): either use them as full-fledged virtual workspaces removing all windows other than the selected group (Rooms) or to just foreground all windows in the group (GroupBar).

Both methods have their merits; Smith et al. argue their solution in GroupBar allows for the user to keep an overview of the other open windows and their state. What must be said is that the method allows for more control from the user, who can decide for themselves whether or not to remove the windows from view. It can also be argued users will like the hard split in workspaces to really define the difference in purpose of the different groups.

Based on the discussions during the studies and in the literature, it would be not a bad idea to make *both modes* available so users can choose for themselves. These two options are only available for windowed managers, since when using a tiled window manager the only option available is to use the groups as virtual workspaces.

This discussion is also related to the discussion about appropriating and habits: what meaning does the user want to give to groups? how does he or she want to use the groups? It is very hard to prove which way would be objectively better and even it is possible, does it even matter since personal preference will still determine the perceived best option. The previous experience will tip the scales; what *is* interesting, is the question what new users prefer and if they tend to choose the same option.

9.2 Limitations of the prototype

Two of the first questions asked when describing the design are: *does this replace alt+tab* and *does this solve everything?*. The answer to the first question is a firm *no, but...*: there will still be a need for a command which quickly switches back-and-forth between two windows to shift the focus, no matter what type of window manager one has. This design might replace the switching between arbitrary windows, but not the quick behaviour of switching back-and-forth which can be done with alt+tab and its clones.

To answer the second question, there are two aspects which are not solved: required physical space and the ability to launch or resume sessions.

9.2.1 Required physical space

When it comes to the weaknesses of this design and what it does not solve, there is one obvious problem: it does not solve the problem of switcher space when the amount of windows goes up; especially when users first start working with the design, they will carry over their appropriated switching management from their previous setup and keep windows open as long as they think they need them. This window switcher tries to keep the amount of windows down via its features, but the fact remains that it uses a lot more space per window which is on the switcher. Future

work has to find out whether or not there is a balance between the physical size of the switcher, the size of the representations and the amount of space needed for gestures.

9.2.2 Launching and resuming

This design is only one side of the cycle: actually launching applications and resuming saved sessions has not been addressed. There are two options for the design of such a system: incorporated in the switching design and on the main screen. In case of resuming saved sessions, it stands to reason to expect the interaction for resuming to be on the switcher as well, since the sessions were saved there. This approach was played with with one of the earlier prototypes built for the student innovation contest at UIST 2010 in New York: application and session launching was done in a different layer which could be accessed. The animation used for the transition between the layers used a familiar metaphor – a chest lid opening and closing – to facilitate learning. Explorations and studies are needed however to validate the ease and effectiveness of use.

9.3 Location of the prototype

Also, it is debatable whether or not the location at the tip of your fingers actually makes the distance to switch shorter. Mouse-movement is quite refined and subtle, so the actual physical distance moved might be even less than you need to move on the suggested design, depending on its final size. Because of the physical attributes of this prototype, with the iPad being a separate component, this was not tested in the study since it would not yield suitable results. What is also important in this case, is the possibility of having fixed locations for windows, which might make the perceived distance shorter because it requires less focus at the end of the gesture: reaching for the small button on the current task bar requires paying attention if you click the right small button as it might have moved, whereas static locations could make that a less complex interaction.

9.4 Learning curves and "natural" interaction

The design and created prototype do require the user to learn the gestures and underlying structure of the design. This means there will be a learning curve for the user when starting to use it. Part of the study design had pre-determined plans to allow the participants to learn the interactions without giving it away; this bit-by-bit release of information and suggestions showed – in our minds – the low threshold of learning to interact with the design. More importantly, participants extrapolated correctly from the given information in multiple cases, even suggesting planned-but-not-implemented gestures on window representations.

Natural interaction is very subjective; one can argue the interaction with our keyboards is natural, but learning to effectively use all its features requires practice. You need to explore and discover the purpose of all the modifier keys or some external source has to inform you. In our eyes, we saw a similar learning curve for participants trying out what they perceived should be possible to do. Natural interaction is not that it works straight away, but that the learning process from beginner to expert is relatively short and painless.

Especially the two-finger gestures used for interactions with the representation's contents suffered from low discoverability, a trait also observed by Smith et al. when testing GroupBar. Smith et al. suggested to include bubble help messages or use some automatic discovery algorithm. Something different, but similar in goal, should be possible for this design, incorporating subtle hints on what to do for beginners.

9.5 Accessibility

A final point worthy of discussion is accessibility; during the studies it became clear that the accuracy of peripheral vision varies from person to person and gets worse with age. This argument

can be extended to the question how well this design fares with other known accessibility options. Is this usable for blind people for example without tactile feedback? While colour-blindness was considered during implementation of the design when choosing colours, accessibility deserves a larger role in future work.

First of all, problems with peripheral vision seem to make it harder, but not impossible to notice changes in peripheral vision. It might be possible to just amp up the visibility by using motion more, otherwise applications will have to rely on other notification methods such as sound or pop-ups in the focal field of vision. When it comes to blindness, touch-screens have actually proven useful by using multiple modalities for interacting. Apple's iPhone comes with accessibility options which change the base interaction model with the phone so they can still be useful.

Accessibility should be a major pillar of the final design and not an afterthought as it is possible to make it accessible.

9.6 Validity of study methods

There is a pitfall and that is the number of aspects which are added in the design. There is a strong need to keep observations and opinions separated according to cause. This prototype adds not just one testable feature to the task bar, but multiple. The choice to use multiple features is caused by the interwoven nature of the features: the location invites for direct manipulation, which is usable in all other aspects like the representations, while the peripheral location also invites for notification and the improvement of the representations. Further specialisations can be seen as future work of this exploratory research.

Exploratory also means no earlier work exists for this kind of task switcher and the limited amount of time only allows for the basic parts of the switcher to be designed. This means the focus lied on verifying whether or not the basics driving the design actually function as predicted and hypothesised, something we find we successfully did.

Designing an artefact as part of design science also means a lot of questions are answered just by looking into other work without further questioning. These questions are seen as motivation for design decisions, not as goal to answer only within this work.

Nonetheless, research rigour is still important and much thought has been put into choosing the methods for the studies. When trying to get hard numbers one quickly hits a wall: there are no metrics available for a prototype that cannot be used without supervision because of the chances of it failing. Furthermore, the type of information we were interested in is hard to capture in 100-percent objective data; to test whether notification in the peripheral area actually works, you need to set up eye-tracking or similar systems. Quantitative measurements can be done as soon as there is a reliable base.

To optimise the accuracy of a qualitative study, a contextual inquiry was chosen because it is in the natural environment of the participant. This setting allows the participant to just go to work and ignore the observer as much as possible. In our eyes, this method worked like a charm. It provided a real suitable context for analysing the data collected during the test of the prototype, which gave the opportunity to also check the behaviour and problems of users we found in literature. We also feel it encouraged the participant to think aloud, being able to set it in contrast with the discussion during the earlier inquiry.

As soon as a more reliable prototype is available – which requires it being embedded in the operating system in a non-intrusive fashion, we propose a study similar to Smith et al. as used with GroupBar. Their study compared the use of GroupBar in contrast with the normal task bar by letting participants use the system for a longer period of time and in their own work structure. This would give a much better insight into how the switcher fits into their work-flow, something that was just a discussion after-the-fact in our studies.

9.7 Who is the user?

The knowledge question oozing through all the discussion about appropriation and habits is *who is the user?* What can we say about him or her, since this is in fact one of the most important questions one asks themselves before doing any form of design work (*what does the user want from the design?, who will use this?*) as well as – and many forget this – in any scientific work (*what does the reader want to know at the end of this?, who will read this?*). In there lies also the main difference which gets unearthed in design science: the practice versus knowledge. In the end, the user will be your *starting point* and you want that user to be as *real* as possible, since that implies your result will be as *true* as possible.

However, this discussion gets convoluted mainly by the general domain of task management where the design tries to find a solution for: there is no single user, there *might* be archetypes of users to be found – for example the *mouse-users* versus the *keyboard-users* – portraying groups of users, but the more interesting question is one step up: *what makes a single user part of an archetype?* What motivation led to the original "decision" to attack problems following such a pattern? Is there a mechanism to "reverse engineer" our true user?

9.7.1 The philosophical foundations of motivation and appropriation

The theory behind appropriation we have after doing the studies is focused on the deconstruction of the motivations – beliefs even – of the user. This is based on the response we got when going after the reason *why* somebody does anything; beliefs are based on other beliefs and to get to the relative bottom, you have to ask somebody to keep converting their beliefs into constructs, which you hope are arguments, but often are other beliefs. This is similar to the concepts of *signifiers* and *signified* as used by Derrida in 1983 and earlier by de Saussure in 1916: where they were after written language and implied meaning, we are after motivation and their implied beliefs.

This layered cake of beliefs similar to the theory of meaning behind deconstruction boils down to the regress problem: to justify one belief, one has to refer to a related, further justified, belief; does one ever get to the bottom of it, in this case: can we ever find a base of argumentation? In our opinion, you need not have to hit rock bottom to find which pillars users build their beliefs on. The useful information is probably available on the level before we dive into the abstract or *choice*. One could say certain beliefs are foundational in the context of what we want to achieve: untangle beliefs to a point we can create behaviouristic models and leave the rest to the philosophers. Who determines what is foundational is for the lucky person to devise models and defend them.

9.7.2 Appropriation as iterative experiences

What can be said is that users build up their convictions based on experience. In the process of jumping from belief to belief there is always a pseudo-argument or motivation: *because when I do this, then . . .* The artefact – specifically the artefacts used before – is as much part of this structure as is the user is, but when an aspect of one artefact pleases either objectively (the efficiency) or subjectively (the experience), it sticks, it becomes a belief or maybe even a foundation. But this is not all an individual journey: social aspects weigh in as well. As we speak Alice is discovering how to use keyboard-based mechanisms for switching after discovering the mechanic from both the prototype and the post-interview where the usefulness of the prototype was discussed.

The discovery of a new way to work with her computer will require her to unlearn behaviour and will probably mean she will use both methods for a while and settle with what she perceives as easier or more efficient. The decision to unlearn is based on an insight it might improve on some problem she has *or*, more interestingly, on a problem she has just now identified. Part of what lies ahead in coming up with new human-computer interfaces will concern unlearning: users paint new interfaces against their current ones and even the ones before that and need to be convinced their methods are "flawed" and the road to learning a new method is not too steep.

And there also lies the practical problem, we can ask the knowledge question "what is *the* better solution", while the practical problem actually is "what is *a* better solution". In the end system developers choose a set of solutions – options – and present them as configuration options, hoping their defaults are sane. People choose between and stick with those options based on how they fit *them* and what it will cost them to use them. One of the major problems for any model or implementation is the fact this decision is made subjectively and based on perception instead of pure argumentation.

The argumentation for a design is often not even included in the design – in many cases on purpose to keep an interface "natural and easy" – and users will need somebody that has discovered that argumentation to be convinced. This phenomena can be classified as low discoverability and solutions require finesse to win over new users, but not to annoy expert users.

9.7.3 The user as a practical problem

Where does this leave us? In a sense the design discussed in this work tries to target the average-case human based on user observations, but has to make space for the large exceptions. This can be seen with the window grouping primarily. It does not try to cover everybody, because it cannot do so realistically. It makes developing for such general systems viable, but also flawed from a strict view.

However, we think design science is a good approach for such open-ended subjects. Lacking an easily identifiable target, you need to make the subject a problem and not a question. An artefact is a good way to refine and create models from and the process can be easily repeated. As long as the methods used are sound for their purpose, it will yield significant knowledge about the user, which is it at the heart of it. Ask "who is *a* user?" often enough to get a good idea of who *the* user is.

In the end this all comes back to a simple best *practice*: to know who your user is, you need to go out and observe. From that point you iterate until satisfied; the main difference is you cannot afford to stick too much to your previous convictions of your users as they have incorporated your artefact and might have changed significantly due to exposure to new ideas.

9.7.4 The user as a knowledge question

When we return to the knowledge question about the user, the changes in between the iterations should be the most interesting: which remain core beliefs and what convictions get watered down by new insights. The person making the belief foundations model should be focusing on these and might wind up with a nice set. This set will not go down easily with everybody in practice, since which person on earth will recognise themselves as a pure implementation of these, untouched by their own experience?

10. Summary and conclusions

This thesis revolved about creating a suggestion for solving a couple of problems with window switching on our current desktop systems. This yielded a design with a prototype which implemented a reasonable subset of that design. This prototype was tested in a small-scale qualitative user study and was found successful in providing ease of access and a useful source for notification. As this was an exploratory dive into this subject, a decent amount of further work was discovered and discussed, both on gaps the design left, as well as what subjects discovered during studies.

10.1 Design and prototype

The design of the window task switcher revolves around solving three problem areas we identified from literature and experience: the costs of switching, the costs of suspending and resuming and notification. All these areas are intertwined and rise to the surface in different ways for different users, but one of the most prominent symptoms is the amount of windows open.

These areas were addressed in the design and the prototype by using four solutions as its backbone: moving the task bar to the keyboard area, allowing grouping of windows, creating richer representations of windows in the switcher and using direct and more complex interactions. The principle factor carrying all these solutions is the location combined with a multi-touch screen.

10.2 Limitations and scope of study

The design limited itself to switching between windows and tasks, not launching applications. Interactions for suspending tasks were touched upon, but resuming these sessions was placed with the domain of launching applications. The implementation of the prototype did not have specific operations for window grouping except for organisation.

The exploratory user studies with the prototype were focused on the hypotheses based on the solutions used for the design. These were: does the location of the switcher make for easy access, do the notifications on the window representations really work, how is switching and organising windows perceived and how easy are the more complex interactions used? They were thus focused on seeing whether the solutions would work at this base level.

10.3 Conclusions from the studies

The results of the study with five participants confirmed all expectations and was incredibly encouraging: ease of access and notification worked and grouping was easy. The first part, a contextual inquiry into the participants' current behaviour not only gave a baseline for reasoning, but also confirmed the problem areas identified while creating the design. The study was, however, not without remarks for further work.

The test with the prototype showed notification worked in the peripheral vision, but brought to light accessibility options should be considered for people with deteriorated peripheral vision. Other remarks were found while testing more complex gestures; as with every interface some learning was required, but participants quickly went from beginner to novice with minimal information.

A prominent conclusion from the study is that users need a purpose for grouping windows together and, since the prototype did not implement any, that led to confusion. Participants could use the mechanics of grouping without hesitation, but to really test the usefulness of the concept, a semantic for grouping (or multiple) should be designed and added in later prototypes.

10.4 Overall conclusions: answers and solutions

This work revolved around the practical problem *How would a task switcher using the location on top of a keyboard in peripheral vision with direct manipulation interactions look like?* This iteration of such a design is the solution to that problem and brought along a discussion of the related areas.

First off, the problems uncovered in chapter 4 can be used generally and are supported by the prototype study. Switching costs, resuming costs and notification should be considerations in *any* task switching design; they answer the knowledge question about what the problems with current systems are.

Second, the design discussed in chapter 6 makes use of both theories used in task management perception and manipulation in the horizontal plane and is *a* solution to the problems uncovered. The conclusions in chapter 8 from testing the design in the form of a prototype firmly indicate these solutions work, but also indicate further work based on problems both in the practical as in the theoretical.

10.5 Future work

The main practical problem lies in the further development of areas left open in the prototype: implementing further interactions and choosing semantic implementations for grouping, for example. Theoretically, the main question is a more philosophical one: for whom are we developing this? The answer to this question is useful in many design decisions, as well as an interesting investigation into the bases of appropriation and habits.

Regarding the design, the base with the improved representations and location in peripheral vision holds and users show little problems while using it, but limitations should be addressed to get clearer results.

First and foremost, semantics for window grouping should be incorporated into the design, where the suggestion is to include both strict workspaces and more loose grouping as used in GroupBar (Smith et al.; 2003), letting users decide what to use. This choice is the second aspect to grouping as it revolves around what users will perceive as useful, which will be heavily influenced by their current methods to switch.

Furthermore, interactions for suspending, resuming and launching tasks or applications – whether they are on the keyboard or not – should be explored and designed. This will not only add purpose to the grouping, but – since just adding for the purpose of purpose is not what we strive for – this will also improve the experience of switching in accordance with Bannon et al.'s guidelines.

Beside all direct features of the design, accessibility should be considered from this point on. When creating the design – although current interactions were designed with accessibility in mind – no time was put into designing the specific interactions for those cases. This can be picked up and designed simultaneously with the "normal" design.

Finally, a longer test with a more stable prototype should be done similarly to the test design of GroupBar (Smith et al.; 2003): let participants use the prototype for a longer period of time instead of their current switchers. This requires the prototype to be extended with the design features discussed above to be really viable for replacement.

Bibliography

- Ahlström, D., Großmann, J., Tak, S. and Hitz, M. (2009). Exploring new window manipulation techniques, *Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group: Design: Open 24/7*, ACM, pp. 177–183.
- Badros, G., Nichols, J. and Borning, A. (2001). Scwm: An Extensible Constraint-Enabled Window Manager, *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference* pp. 225–234.
- Bannon, L., Cypher, A., Greenspan, S. and Monty, M. (1983). Evaluation and analysis of users' activity organization, *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, number December, ACM, pp. 54–57.
- Bernstein, M., Shrager, J. and Winograd, T. (2008). Taskposé: exploring fluid boundaries in an associative window visualization, *Proceedings of the 21st annual ACM symposium on User interface software and technology*, ACM, pp. 231–234.
- Bi, X. and Balakrishnan, R. (2009). Comparing usage of a large high-resolution display to single or dual desktop displays for daily work, *Proceedings of the 27th international conference on Human factors in computing systems - CHI '09* p. 1005.
- Brandl, P., Forlines, C., Wigdor, D., Haller, M. and Shen, C. (2008). Combining and Measuring the Benefits of Bimanual Pen and Direct-Touch Interaction on Horizontal Interfaces, *Proceedings of the working conference on Advanced visual interfaces* pp. 154–161.
- Czerwinski, M., Horvitz, E. and Wilhite, S. (2004). A diary study of task switching and interruptions, *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04* 6(1): 175–182.
- de Saussure, F. (1998). *Nature of the Linguistics Sign*, 2nd edn, Bedford, pp. 832–35.
- Derrida, J. (1985). *Derrida and Différance*, Warwick: Parousia Press, chapter 1, pp. 1–5.
- Grudin, J. (2001). Partitioning digital worlds: focal and peripheral awareness in multiple monitor use, *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, pp. 458–465.
- Henderson Jr, D. and Card, S. (1986). Rooms: the use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface, *ACM Transactions on Graphics (TOG)* 5(3): 211–243.
- Hevner, A., March, S., Park, J. and Ram, S. (2004). Design science in information systems research, *Mis Quarterly* 28(1): 75–105.
- Hutchings, D. R. and Stasko, J. (2004a). Shrinking window operations for expanding display space, *Proceedings of the working conference on Advanced visual interfaces - AVI '04* p. 350.
- Hutchings, D. and Stasko, J. (2004b). Revisiting display space management: understanding current practice to inform next-generation design, *Proceedings of Graphics Interface 2004*, Canadian Human-Computer Communications Society, pp. 127–134.
- Iqbal, S. and Horvitz, E. (2007). Disruption and recovery of computing tasks, *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, pp. 677–686.

- Myers, B. (1988). A taxonomy of window manager user interfaces, *Computer Graphics and Applications, IEEE* **8**(5): 65–84.
- Nagata, S. (2003). Multitasking and interruptions during mobile web tasks, *Human Factors and Ergonomics Society Annual Meeting Proceedings*, Vol. 47, Human Factors and Ergonomics Society, pp. 1341–1345.
- Palmer, S. M. and Rosa, M. G. P. (2006). A distinct anatomical network of cortical areas for analysis of motion in far peripheral vision., *The European journal of neuroscience* **24**(8): 2389–405.
- Peschier, B. and Majid, A. (2010). Creating an interface for an everyday multi-monitor multi-touch system, *Scientific Writing HT09* .
URL: <http://fizzgig.nl/static/sciwri-multitouchmonitorsetup.pdf>
- Po, B. a., Fisher, B. D. and Booth, K. S. (2004). Mouse and touchscreen selection in the upper and lower visual fields, *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04* **6**: 359–366.
- Ringel, M. (2003). When one isn't enough, *CHI '03 extended abstracts on Human factors in computing systems - CHI '03* p. 762.
- Robertson, G., Horvitz, E., Czerwinski, M., Baudisch, P., Hutchings, D., Meyers, B., Robbins, D. and Smith, G. (2004). Scalable fabric: A flexible representation for task management, *ACM CHI*, pp. 1–11.
- Ryall, K., Forlines, C., Shen, C., Morris, M. and Everitt, K. (2006). Experiences with and observations of direct-touch tabletops, *Proceedings of IEEE TableTop the International Workshop on Horizontal Interactive Human Computer Systems*, number Figure 1, IEEE Computer Society, pp. 89–96.
- Smith, G., Baudisch, P., Robertson, G., Czerwinski, M. and B (2003). Groupbar: The taskbar evolved, *Proc. OzCHI* pp. 1–10.
- Spink, A., Cole, C. and Waller, M. (2009). Multitasking behavior, *Annual Review of Information Science and Technology* **42**(1): 93–118.
- Tak, S. and Cockburn, A. (2010). Improved window switching interfaces, *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, ACM, New York, New York, USA, pp. 2915–2918.
- Tashman, C. (2006). WindowScape: a task oriented window manager, *Proceedings of the 19th annual ACM symposium on User interface software and technology*, ACM, pp. 77–80.
- Wieringa, R. (2009). Design science as nested problem solving, *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology - DESRIST '09* p. 1.
- Wigdor, D., Shen, C., Forlines, C. and Balakrishnan, R. (2007). Perception of elementary graphical elements in tabletop and multi-surface environments, *Proceedings of the SIGCHI conference on Human factors in computing systems* p. 482.

Appendix: Study structure

This is the *original* structure as used for the entire study. It is by no means a strict guideline and, as mentioned earlier, was often diverged from due to the organic nature of discovery. It however does include all subjects touched upon and information given to the participants.

Situation

At the location of the participant, so everything should be mobile. We're looking for their normal behaviours and since the setup is quite mobile, let's make use of that.

Part 0: Introduction

Tell participant the course outlines of what we want to do:

- Test a prototype for task switching
- Not testing the user's adaptability for new interfaces ;-)
- Some background on how participant is currently managing.
- A discussion/interview after playing with the prototype.

Part 1: Observation/Interview

- Mainly in the interest of creating a context/baseline of the habits of the interviewee.
- Questions about task switching ideally after observing behaviours (to prevent biasing/priming)

Setup

Camera on keyboard, mouse and screen. 'Assistant' taking notes about responses (minutes), I take notes of behaviour.

Procedure

Interview while working on computer, so participant can show what he/she means. Observe a short (<5min) period of work.

Observation interests

Setup:

- Screen amount & size
- Operating System
- How are notifications setup
- Any additional tools for switching/notification
- ...

Behaviour:

- Use of current switchers (keystroke/task bar/visual/direct click)
- Keyboard-centric or mouse-centric
- Layout of windows (peripheral placement, grouping)
- Amount of windows open
- Operations used during complete task switch/interruptions
- ...

Interview topics

Baseline:

- What kind of computer user are you? (recreational, professional: nature of profession?)
- How many hours per day do you use your computer?
- What kind of applications do you work in primarily? (office, graphics, IDE, games, ...)
- ...

Switching:

- How many different tasks would you think you work on at given times?
- Do you have any type of (conscious) strategy for structuring your windows?
- ...

Based on observed behaviour:

- Why do you keep your <communication client> open/closed/hidden/tucked in peripheral vision?
- ...

Opinions-FreeForAll:

- What are your major annoyances while managing your windows?
- ...

Part 2: Prototype acclimatisation

- Easy and simple operations; prototype is course anyway. Introduction into the purpose and limitations of the prototype (without that much spoilers O:))
- Participants are asked to use the switcher as much as possible (reflexes might still trigger command+tab and others).
- Participants may ask questions about operations in applications (maybe they are not used to them).
- Participants are asked to explain as much as possible what they are thinking/reasoning.
- No pressure O:-)

Setup

Camera on prototype, mouse, keyboard. Assistant giving instructions. I take notes: behaviour, comments.

Task 1: Exploration

Setup: System is pre-setup with applications in groups.

Task:

- Get topic of latest email and the amount of emails in inbox
- Open a new tab in Safari and go to your favourite news webpage
- Switch to the next song in iTunes

Task 2: Configuration

Setup: System is pre-setup with applications all in one group.

Task:

- Group the windows to your own liking
- Open a new browser window and setup a couple of tabs
- Open and setup a new browser window intended for a specific group

Part 3: Prototype testing

Task 3: Notifications

Setup: Result of task 2

Task:

- Type in a short summary of three texts (from different open windows) about the same subject.
» *During work, music plays (and goes to next songs) and email arrives.*

Direct-post-task question: did you notice the notifications?

Task 4: Extended exploration

Most of it does not really do anything, but is needed to reflect on by the user.

Pre-test: tell about the interactions on iTunes representation?

Setup: Result of task 3

Task:

- Pause music
- Close all windows related to task 3
- Play music
- Skip to next song
- Flip over iTunes representation
- Flip back
- Flip over task containing iTunes
- Flip back

Part 4: Post-interview

Setup

Camera still on prototype, but mainly used for audio, but can be used if participants wants to explain a statement. Me: questioning+notes, Assistant: minutes.

Features

- Were the notification suitable?
- Would you hide/close your email, im in these circumstances?
- How would you fit this into your task managing schemes?

Interaction

- How logical were the operations?
- (some question about 1-finger versus 2-finger)

Free-for-all: fun, but not critical

- What would you say are the main strengths?
- What would you say are the main weaknesses?
- Any ideas/suggestions?
- Anything else?

Part 5: Post-stuff

thanks, other discussions, cup of tea.