Vrije Universiteit Brussel

Faculteit Wetenschappen

Departement Informatica

Academiejaar 2001-2002

Task-based Web Site Design

Gregory Vandenbroucke

Promotor: Prof. Dr. Olga De Troyer

Verhandeling met het oog op het behalen van de graad van Licentiaat in de Toegepaste Informatica

Preface

This thesis couldn't have been written without the help of many people around me. First of all I would like to thank Prof. Dr. Olga De Troyer for guiding me and for correcting this dissertation. I would also like to thank Prof. Dr. Patrick Steyaert for giving me the subject of this dissertation.

I would also like to thank my family for supporting me during the elaboration of my dissertation. I would like to thank my aunt, Monique, for supporting me during these stressful times. I thank my brother Cedric for pushing my gently in the right direction when I needed it. I also thank my brother Pierre for distracting me when I needed it the most.

Finally I would also like to thank my friends. I won't enumerate all of them, but I thank Jonathan with whom I could share my thoughts when I needed it the most. I also thank David, Jan, en Arne for all the insightful discussions we had over the past year.

Introduction

The World Wide Web (W3) was initially, in 1989, developed by Tim Berners Lee to allow scientists to share knowledge through the internet using hypertext. Since its democratization, the W3 has evolved into a more complex system where information sharing is not the only goal. Nowadays, it allows people accomplishing complex tasks. Among the most popular ones are managing e-mails and shopping on e-commerce web sites. They offer users services that were originally not available (e-commerce) or only available as desktop applications (e-mail).

Offering users greater experiences when using user interfaces has become a major concern for the past decade. However, the main focus has been directed towards desktop environments. This is partly due to the fact that existing methodologies can't be applied on the hypertext metaphor of the W3. Offering users greater experiences on the W3 has become critical since it democratization. Few methodologies for improving a users' experience on the W3 exist, but most of them fail to address the design of a web site from the early stages; namely, the process of knowing your users; to the ultimate design of a web site.

In this dissertation I present a Task-based design methodology based on three existing methodologies: Task Analysis and Modeling, Bridging the Gap methodology, and Interaction Design Patterns. Task analysis and modeling is a user-centered design methodology which focuses on collecting data about users and the tasks they perform and on structuring the data to gain insight into the users. The Bridging the Gap methodology is an Object Oriented design methodology helping designers making an initial web site design based on the data collected during Task Analysis and Modeling. Finally, Interaction Design Patterns help designers improving a web site design with respect to the user using proven design knowledge. The task-based methodology I present unifies the three methodologies of choice into one unifying iterative design process for enhancing the users' experience on the W3.

The iterative design process I present in this dissertation helps designers from the early stages of knowing your users to the final web site design. This iterative process ensures that the users' tasks will be integrated in the final design and that the web site will be usable. I also present several guidelines which will help the designers in

taking the various steps and will help them at the various stages of the iterative design cycles.

Chapter 1 focuses on the W3 and the users. There I will present the difficulties users may encounter when moving from a desktop environment to the W3. Then I will discuss how users perform tasks on the web from a hypertext perspective. In Chapter 2 I will present the Task Analysis and modeling methodologies and several techniques for performing this methodology. I will also discuss why it is important to consider those methodologies. In Chapter 3 I will discuss the Bridging the Gap methodology and how it should be applied. In Chapter 4 I will introduce Interaction Design Patterns and how they can help in improving a web site design. I will also present a Pattern Language and explain why it is my language of choice. Finally in Chapter 5, I will introduce my iterative design process and explain how it should be applied and why it improves the design of a web site.

Contents

1 The Web

The World Wide Web (W3) is based on the hypertext metaphor. This metaphor has several consequences with regard to the users performing tasks on the web. I will first introduce this chapter with a short history of the W3. Then I will compare the web environment with the desktop environment and discuss the implications of moving from a desktop environment to a web environment with respect to the user. Finally, I will discuss how tasks are performed on the W3.

1.1 A Brief History

The ideas of the web can be traced back to 1945. In (V. Bush, 1945) a machine called the Memex is described. The purpose of this machine was to enable scientists to share knowledge and to organize that knowledge in order to retrieve it more efficiently. To organize that maze of knowledge Vannevar Bush described a linking mechanism that would link related pieces of knowledge to each other. This mechanism is today known as hypertext.

In the 1960s Doug Engelbart prototypes an "oNLine System" (NLS) which does hypertext browsing editing, email, and so on. He invents the mouse for this purpose. Further on, Ted Nelson coins the word Hypertext in (L. Wedeles, 1965). Andy van Dam and others build the Hypertext Editing System and FRESS in 1967.
In 1989 a proposal (T. Berners-Lee, 1989) concerning the management of general information about accelerators and experiments at CERN is circulated. It discusses the problems of loss of information about complex evolving systems and derives a solution based on a distributed hypertext system.

From this point on, research about what we now call the World Wide Web (W3) is initiated. In (T. Berners-Lee, 1993) the W3 is defined as follows:

- The idea of a world in which all information items have a reference by which they can be retrieved;
- The addressing system (URL) which the project (the W3 project) put in to make this possible, with many different servers;
- A network protocol (HTTP) used by native W3 servers giving performance and features not otherwise available;
- A markup language (HTML) which every W3 client is required to understand, and is used for the transmission of basic things such as text, menus and simple online help information across the net;
- The body of data available on the Internet using all or some of the above.

Tim Berners-Lee is regarded as the inventor of the W3. His vision of the Web is something much more than a tool for research or communication; it is a new way of thinking and a means to greater freedom and social growth than ever before possible. He is currently the director of the World Wide Web Consortium, the coordinating body for Web development, and he occupies the 3Com Founders chair at the MIT Laboratory for Computer Science.

1.2 Web vs Desktop

The World Wide Web, nowadays, helps users in accomplishing certain tasks. The tasks may range from sending e-mails (e.g. Hotmail), to consulting encyclopedia, or managing a bank account, etc.

Tools (e.g. DreamWeaver) and technologies (e.g. Flash) for designing web pages help designers for creating highly interactive web pages. But unlike most desktop application (e.g. Windows applications), websites aren't based on a set of standards which can help users in completing their tasks.

When moving from a desktop environment to a web environment, users get lost because the model of navigation on the Web differs from that of desktop applications. This difference causes confusion among users (H. Shubin, 1997).

Five main differences between a desktop application and web sites can be distinguished, which can have a significant impact on a users' experience when fulfilling certain tasks. These differences are also discussed in (H. Shubin, 1997) and (D. Van Damme, 2002).

1.2.1 Structure

The web is an enormous environment with interconnected pages using hyperlinks. Google[1] for instance served a direct access to 3 billion documents in December 2001.

[1] Google is one of the most popular search engines nowadays, http://www.google.com

No wonder people can get lost in such a structure compared to a desktop application. Furthermore, web designers can also create pages of any length, with any number hyperlinks in it. People often get buried in the amount of information available, and get lost in their tasks if not enough attention is paid to the structure of a given site.

1.2.2 Quitting a Task

Unlike desktop applications, web sites don't provide control in how tasks can be terminated. Desktop applications offer clear marked controls (e.g. exit commands in the File menu) which let's user exit an application. Usually, the only way to exit a website application is by exiting the web browser itself. Closing the browser doesn't offer the user a command to save work in progress, even though embedded applications (e.g. a Curl applet[2]) can offer a feature to save work in progress. But as this is not tightened to the browser itself nothing prevents the user of losing the unsaved data. Another way of quitting a task is by following hyperlinks. Hitting the back-button doesn't guarantee that a task will be restituted in the same state as was previously obtained.

1.2.3 Heterogeneity

The main purpose of a web browser is supporting the activity of navigation between web pages. So in order to be able to fulfill some task on the web, a desktop application is needed. This tends to blur the separations between both environments. The implication of this blurry separation is best understood with an example:

[2] Curl is mainly an object oriented programming language developed by Tim Berners Lee and some people at the Massachusetts Institute of Technology http://www.curl.com

> Imagine a user checking his mail on a Hotmail account. Hotmail let users manage their mails in a web environment. Users may send new mails, may delete older mails, may read received mails, etc. When people use the web interface to manage their mails, the tasks they are doing are consistent with respect to the result they see on screen (e.g. deleting a mail). Mixing the controls of the web browser with those provided by the web interface may lead to confusion. For instance a user who deleted a mail may try to use the back-button to undo the delete. Obviously, the mail won't be restored. The only way of restoring your mail is by using the web interface (restoring it from the trash can).

This may seem to be a simple example. But how many times didn't people lost information (e.g. when filling forms, etc.) just by using the features of the browser. Warnings like "Page Expired" are unfortunately very common.

1.2.4 Context

Dialog boxes help user complete subtasks in a desktop application or help providing useful information to the user (e.g. error messages). Those boxes retain the focus of the application and appear in front of the main window of the application. As a result users are forced to deal with the dialog box if they want to complete the rest of their task, and the main window behind the box provides useful context in completing a subtask. On the web, information is mostly shown without context, requiring the user to go back and forth between pages to complete a given task. Pop-up windows can be used, but they don't retain any focus and some browsers (like Mozilla) allow killing pop-ups.

1.2.5 Delays

One thing inherent to the web, and also the internet, are network delays. Navigating between pages takes longer than moving from a main window into a submenu of a desktop application for instance. These delays are due to the network traffic and to computation. It adds to the users' cognitive loads and therefore can have a negative impact on the users' context. Rich clients can reduce the delays by maintaining as much computation of the presentation layer, (D. Van Damme, 2002), on the client. But still communication will be required. Besides adding to the cognitive load of a user, users can get frustrated by the delays and ultimately abandon the task.

1.2.6 Guidelines

On the standard platforms (e.g. Windows, Mac OS, etc.) style guides are used to give a uniform *look and feel* to different applications. The file-menus look the same; the help-menus look the same, etc. Users familiar with applications of a specific platform

will more easily master new applications because of that. The web breaks that model, because of its diversity and freedom. Everybody can nowadays develop a website and put it on the web. Guidelines are available, some good and some bad. But what makes a good guideline? I suggest it would be a standard guideline, but because the web is so heterogeneous, it makes them practically impossible to apply on every single website. Furthermore, guidelines may not be the best solution when it comes to the web. This will be discussed in **4.2 Interaction Patterns Explained**.

1.3 Performing tasks on the Web

Initially, computers were used to assist people in accomplishing their tasks. Later on computers were also used for entertainment. As the main purpose of using computers was to help businesses accomplish their work, the desktop metaphor was used and still is used for making computers accessible.

The web on the other hand uses some kind of a different metaphor, namely the page metaphor. The page metaphor is an appropriate metaphor for browsing static text with hyperlinks. But as metaphors on their own can seem very suited for accomplishing specific tasks, once they're mixed together people using them can be left astray. Web browsers are today fully integrated in the desktop environment. Moving from the desktop environment to the web environment implies a different approach in how tasks are performed because of the different metaphors used. Even in a web metaphor some desktop elements can be integrated (e.g. applets) which tends to blur the differences even more. As a result users may find it difficult accomplishing their tasks because it is difficult to determine which controls apply to which part of the system.

1.3.1 Classification of tasks on the Web

As pointed in (C. Fellenz et al., 1998) tasks being accomplished on the web can be categorized into 3 classes:

1. Browsing: is the activity of moving from one web page to another (or even from one part of a page to another part of the same page), using hyperlinks or by directly using URLs to retrieve some pages.
2. Performing Transactions: this is a user task which consists of giving information, typically by using a form, and providing information back to the user.
3. Running applications: those applications are essentially desktop style application integrated into a browser. Examples of such application are Java applets, Flash application, or even more recent technologies like Curl[3].

As the distinction between the desktop environment and the web environment tends to blur, as I mentioned earlier, some usability issues arise. I'll give an overview of the most commonly encountered issues with regards to the classifications.

[3] http://www.curl.com

Browsing

Typically when a desktop application starts, people find themselves on the main window of the application. When moving to the web, another story unfolds. People may have entered a web site by following some links or by typing in some URLs. There is no guarantee whatsoever that a user will start on the main page of the given website. This can have the implication that the user doesn't know where he is on the site and thus doesn't know where is has to go to accomplish a specific task.
Frames also tend to have a bad impact on the usability of a site. For instance frames linking to other pages with frames result in a nesting of frames which can be disorienting with respect to the user. Linking or book marking frames can also be a confusing task for not so experienced users. Finally, printing the content of frames can also be difficult.

Dynamic content embedded in a web page also brakes down the page metaphor. A typical example is the use of the back-button inherent to all popular web browsers. A user might expect that using the back-button in a page that embeds a Java applet will back-up in the navigation of the applet. But the ultimate result will be a reloaded applet or the user might find himself within a page he followed to get to the applet. This is a clear example that shows how confusing the mix of the 2 metaphors can be.
Pop-up windows also break the page metaphor in the sense that it is not possible to back-up in the navigation of a newly created window. Instead the user has to go back to the previous window.

Performing Transactions

Like in the previous section, following links during transactions without providing useful information about where they are in the transactions can be quite confusing.
Again the use of the back-button can be quite problematic. When using the back-button a previously cached page is rendered. This can have two major impacts on usability: first of all a user can accidentally re-submit previously entered information; secondly, an out-dated version of the page is viewed which can lead to erroneous information.

I remember, a few years ago, I wanted to buy a computer because the one I used to have wasn't powerful enough to run applications like I use in the context of my studies (compilers, design tools, etc.). I also wanted a reliable computer because the one I used to have was made up of assembled parts and wasn't very stable. After reading some reviews about the available computer I decided to buy a DELL computer. Unfortunately DELL only sold computers in Europe through the web and the web was pretty new to me.
One of the nice features about DELL was that it gave the opportunity to tune a little bit your configurations and gave in real-time the adjusted prices. So I started testing several configurations and checking the prices. While I was testing the configurations, I was also doing several other tasks on my desktop (checking mail, etc.). I inadvertently added one of the configurations I was checking to my shopping basket. I didn't really pay any attention to it and went back (by using the back-button) to one of the pages on DELL's site, thinking it wouldn't be in my basket anymore. Finally, I tested some other configurations and found one that suited me well for the given price. I added it to my shopping basket and once again I didn't really pay any attention to the content of my basket. Then I entered the secured part of the site, in order to make the order. I had to fill in several pages of information to accomplish the order. Finally, I was on the last page where people have to submit the order. I checked if all my entered information were correct and then realized I was just about to buy 2 computers. If I had the money to buy them, I wouldn't probably have minded it. But the poor little student that I was didn't really wanted to pay that much. So I wanted to cancel one of the two orders, but since I was already in the secure part of the site it wasn't possible.
Finally, I had to restart the process allover again.
This is an example of how even an experienced desktop user can be tricked by a combination of the use of the back-button and some distraction.

Running applications

During my training course I was asked to test a new technology known as Curl (which I mentioned earlier). It gives the opportunity to run a desktop like application into a browser (much like Java applets do it). The motivation behind that was the zero client administration. The only thing needed was a plug-in. Once this was installed the user had to go to a given webpage and an application would run into the browser. The main advantage of it is that the deployment of such an application is far easier than having to install the application on different desktops with probably different operating systems, etc. Like I mentioned earlier it breaks down the page metaphor.

One way to solve this is, is to put the application in a pop-up window without the browser controls. This isn't perfect because most browser support keyboard shortcuts. Applications build for novice users should take this into account.

1.3.2 Additional considerations

The creation of applications on the web is the result of two driving forces (P. Pop, 2000):

- Internet portals like Yahoo, MSN, and Netscape are investing in web applications as a way to attract more people that are converted to revenue through advertisement.
- Companies and organizations are using web applications as a way of reducing development, distribution, and maintenance costs compared to traditional applications.

But as Jakob Nielsen, which is considered to be one of the web usability experts, explains in (J. Nielsen, 1998), it is unlikely that a user interface that is optimal for browsing hypertext also will be optimal for every other thing. Furthermore, he explains that optimized stand-alone applications deliver a superior user experience. Even tough he fails to speak about Java applets and similar technologies (which were discussed in previous section); some thoughtful considerations must be paid when moving from the desktop environment to the web environment.

In (P. Pop, 2000) a study was conducted comparing the usability performance between a desktop based application and a web based application which let users doing the same tasks. The application was a calendar. For the desktop based application Outlook was used for conducting the tests. For the web based application Yahoo!Calendar was the application of choice.

The conclusions of the conducted tests were that the performance of accomplishing some basic tasks was degraded when using the web based version. The three main factors were:

- The limited interaction mechanism provided by the web browsers. The web browsers are good for browsing and for filling in forms. When it is used for anything else the limitations compared to a desktop version are substantial.
- A mismatch between the users' mental model (more on this in **3.2.1 Conceptual User Modeling**) of the web application and the actual application. Like I previously mentioned, the distinction between the web browser and application itself is blurred.
- Finally, network delays have a negative impact on the users' experience because of limited performance.

For all those reasons a development team should first decide whether moving in a Web environment is a judicious choice. Applets or similar technologies can be seen as a viable alternative to more conventional navigation based applications. But the team should keep in mind that using those alternatives brings new usability issues into play (e.g. the back-button)

1.4 Conclusion

The Web originally was created to share knowledge, using hypertext. Rapidly, new applications (e.g. encyclopedia, search engines, mail accounts, etc.) were conceived which gave users all over the world the opportunity to enjoy desktop like applications on the web. The characteristics of the web were not specifically conceived with this in mind. We discussed the problems which can occur, due to the shift in the metaphors used. Other characteristics, like network delays, can also result in usability problems. We also addressed the problems introduced when trying to bring more advanced applications, like mail accounts, on the web. These problems can be circumvented by using more advanced technologies (e.g. Curl), but introduces also some new usability issues.

The remainder of this dissertation is focused on designing usable web sites; more precisely on designing the tasks users perform on the web. The first step that has to be taken in this process is gathering the relevant data about the users and their tasks. This is the subject of the next chapter.

2 Task-based Design

"One most unfortunate product is the type of engineer who does not realize that in order to apply the fruits of science for the benefit of mankind, he must not only grasp the principles of science, but must also know the needs and aspirations, the possibilities and the frailties, of those whom he would serve."

-- Vannevar Bush

Before designing a user-interface some attention has to be paid at the people who in the end will use the system (be it a website, a desktop application,…).

In this chapter, I'll discuss why paying attention to the end-user is important when designing a web site and more in particular the user interface of the web site. Such an approach is called a user-centered design and is discussed in section **2.1 User-centered Design** of this chapter. A special case of a user-centered design is a task-based design. Task-based design is a modeling technique that covers the early processes needed to uncover the users' needs to the actual user-interface design. We will discuss task-based design in section **2.2 Why Study Task-based Design**. The actual design of the interface will be discussed in a later chapter. In this chapter we focus on all the steps needed to acquire the necessary information to build the final interface. These steps include task analysis and task modeling. Task analysis is concerned with the gathering of all the necessary data. Several techniques for gathering the data will be discussed like interviews, observation, etc. Task analysis will be covered in section **2.3 Task Analysis**. Task modeling is mainly interested in gaining insight in the data. Gaining insight is achieved using several modeling techniques like task decomposition, etc. Task analysis is explained in section **2.4**

Task Modeling. Also, several representations used in order to document the gathered data will then be discussed.

2.1 User-centered Design

User-centered design methods all share the thought that the system being developed should be designed for the people that use the system (the users). The designer must think in terms of the user and not in terms of the system. The system is designed upon the users' abilities, their needs, their context, work and tasks.

Typically software engineers tend to think about a software system in terms of:

- What can be easily built on this platform?
- What can I create with the available technologies?
- What kind of tools will I use?
- What do I as a programmer find interesting to work on?
- …

Software engineers usually are concerned about the system, not about those who will use it.
However, the last decade has shown a trend towards systematic development of systems with a higher degree of usability (M. van Welie, 2001). This is the part where designers come into play. Their main goal is to design a highly usable and useful system (H. Hartson, 1998). As opposed to software engineers who focus on the implementation of the system itself.

Now, what makes a system usable and useful? Clearly, there isn't a well-defined answer to that question. But from this question we are able to derive some measuring concept about how usable a system is. In user-centered design the word "usability" is used for that purpose. Several definitions exist, some show a high degree of similarity but others may seem very different. Each of those definitions can ultimately be used to measure the usability of a system or a combination of those techniques. In (M. van Welie, 2001) a framework is proposed for dealing with these various views on usability.

Instead of focusing on the different definitions, and how they are measured, we'll focus on the benefits of it and thus why it is important to involve the user in the design of a system.

Usability may not only give benefits to the end-user but has also the potential to give benefits to other stakeholders. Here is an overview of the potential benefits as found in (C. Karat, 1994):

- Increased Efficiency: One reason of introducing software systems is to get more work done in less time. Clearly, if a user struggles with the interface less time will be spend at doing the work.
- Increased Productivity: As a result of the previous point, more time will be spend at doing the work.

- Reduced Errors/Increased Safety: Poorly designed interfaces may cause undesirable side-effects as users prone at making errors.
- Reduced Training: this can reduce the time of training. Hence, reduced costs.
- Reduced Support: a well-designed interface may cause fewer problems to a user, which minimizes the need for support.
- Improved Acceptance: users may reject using a particular system if they don't feel comfortable using it.
- Savings in Development Costs: integrating usability issues early in the design life-cycle may ultimately reduce costs. This is one of the software engineering principles which state that making changes later in the life-cycle causes more costs because extra iterations are needed.
- Increased Sales: happy users are more likely to use your system over other similar systems.

I would like to pay some more attention on the potential benefits. Let me explain why the word potential is used when talking about benefits. As (C. Karat, 1994) mentioned other stakeholders come into play when designing for usability. Benefits like reduced support and reduced training may actually cut some additional revenues of a company, because those services will be less solicited. From this we can deduce that when designing for usability a company may be reluctant about applying this.

I remember reading an article about a usability issue the NASA had. They were confronted to the problem that using ballpoint pens would not work in zero gravity. So they spend a decade and a huge amount of money developing a pen that wrote in zero gravity and had several other benefits. When confronted with the same problem, the Russians used a pencil. Clearly, in this case there were more drawbacks (costs) than benefits.

Further reading about the potential benefits can be found in (R. Bias et al.,1994).
Now let's move on to the next section. I'll introduce you to the concept of task analysis which is the first step taken in a task-based design process.

2.2 Why Study Task-based Design

One might ask why I discuss task-based design (and thus task analysis) as a process of designing interfaces, while there are several other techniques for designing interfaces. This is a fair question, and let me try answering that question.

Many design methods exist but only few of them try to address the entire design process. In Task-based user interface design, we try to cover the entire process from early analysis to prototyping (M. van Welie, 2001). Applying methods that fail to exist during the whole design life-cycle (from analysis to prototyping) may result in a badly designed system because bridging the gaps between the various steps of a design process is not trivial at all. A process that helps a designer bridging those gaps will ultimately result in a better web site UI design.

Furthermore, in terms of the design life cycle, task analysis (part of task-based design) belongs to the beginning in requirements capture, whereas the cognitive models are normally used towards the end of the process during evaluation (A. Dix et al., 1998). […] However, many of these techniques (HCI techniques) have been directed towards analyzing designs, as a form of evaluation, rather than contributing to their construction (S. Wilson et al., 1995). Task-based design is one of the few techniques used for the construction of a user interface (UI). Evaluating a user interface may be valuable but it doesn't say how you could create a usable UI. It can help a designer at tracking the good and bad things about the interface, but it doesn't say how to enhance the interface. Another point is that those techniques depend on an existing interface/prototype. Task-based design can help building an interface from scratch.

2.3 Task Analysis

First of all, I would like to emphasize on the two different existing views about task analysis. During my research I realized there existed two different views on task analysis in the literature. The first, more classical one, describes task analysis as the way of gathering data about users and how to structure the data. The second and more recent way, views task analysis as a part of a bigger process, task based design. In task-based design, task analysis is fined as the process of gathering data and only that. The process of analyzing and structuring the data is part of task modeling. As my thesis is about task-based design I'll use the term task analysis defined by the task-based design.

2.3.1 What is Task Analysis?

Task analysis is the process of analyzing how people perform their jobs. It is the process of gathering data about them. Several techniques, which I'll describe later on, exist for that purpose. Task analysis concentrates on three different aspects which are correlated: users, their tasks, and the environment in which they perform their tasks. Although there isn't a clear distinction between those three aspects I'll describe them separately in order to give more insight about the subject.

Users are the people who in the end will use the system, so not considering them would be a huge mistake. They decide whether they will use the system or not. Users are people with likes and dislikes, habits and skills, education and training that they bring into play whenever they interact with a system (J. T. Hackos et al., 1998).

When talking about users in task analysis we try to answer questions like: who will use the system, what they know about their task, what they know about an existing system they use, what their mental models are, what are the differences among different users, etc.

To build a successful interface you need not only to know about the users. You also need to know about the tasks they perform. Several degrees of analysis are possible here. A combination of all those will bring a better understanding about the tasks.

In order to understand how a particular process is being accomplished *workflow analysis* is used. In workflow analysis some attention is paid about the collaboration between several users in order to complete a task. *Job analysis* on the other hand focuses on the particular job a user has to do. We emphasize on the several tasks a user performs over time. *Task sequences* try to decompose tasks in subtasks and analyses the sequence of the subtasks. Several other techniques (like *task hierarchies*, *procedural analysis*, etc.) also exist.

Another important aspect, one that is more related to the user, is what is called the "stages of use". Stages of use try to grasp the level of expertise the users have/need in order to accomplish their tasks and it also thinks about the evolution of that expertise over time.

Thinking about the users' environment may seem a little awkward but people do not perform their work in isolation, they are influenced by the activity around them. If a system doesn't fit the environment, it may be frustrating or difficult to use.

The physical environment may influence the way people work. Imagine a coal mine worker that has to work with a touch-screen in order to enter some data. The screen will end up with dust all over it. Maybe the screen will stop functioning. This is an extreme example but imagining that more subtle problems can occur is obvious.

The social and cultural environment may be of interest when doing task analysis. Analyzing this may point out some stuff you didn't even think about. For instance, pictograms may be perceived very differently between an American user and a North African user. Taking this into account may avoid some undesirable side effects.

Task analysis is not specific to user-centered design. Task analysis has roots in many study fields. I'll describe some of the techniques used to gather the data in **2.3.3 Techniques**. But in order to give some more insight in those techniques, I'll give a brief history about those techniques.

2.3.2 A brief history

As I mentioned earlier, task analysis borrows techniques from all sorts of fields.
Observation techniques are borrowed from anthropology and ethnography. Anthropology is the study of people and ethnography is the process of immersing oneself in a culture in order to understand that culture. Respect for the people you are observing and the importance of paying attention to their ways of working, their environment and their culture (J. T. Hackos et al., 1998). Of course, differences are noticeable between the different ways the techniques are applied. For instance, in an ethnographic observation, the observation tends to last for some months or even years. In task analysis this is clearly not feasible.

Cognitive psychology is another domain that contributed to the development of task analysis. Cognitive psychology is interested in how people think and learn. Cognitive psychology tells us that it is the user and not the designer who will in the end determine how the product is used. The talk aloud protocol is one of those techniques used in task analysis. Also usability testing is derived from this domain.

Technical communication and instructional systems design (ISD) is about writing documentation and can also be used as a source of information. Technical communication and ISD are primarily focused on documentation for software and hardware. This documentation can be used as a source for task analysis like for instance instruction booklets, training materials, etc.

The list of domains I handle here is not exhaustive. Task analysis as mentioned in (J. T. Hackos et al., 1998) borrows also from fields like scientific management, participatory design, market research, etc. Now that I mentioned the background of those techniques, I'll introduce them more in depth, taking the advantages and disadvantages into account when applying the different techniques.

2.3.3 Techniques

The process of task analysis is not a simple one of collecting data. Ideally, the process should be iterative (more on that in **Chapter 5**) with periods of data collection interspersed with analysis. In practice, limitations of time and cost will cut this short, so the skill of the analyst lies in foreseeing possible problems in analysis and obtaining data as quickly and economically as possible (A. Dix et al., 1998). I'll describe the three main techniques for data gathering: use of documentation to gather data, observation techniques, and interviewing.

- Documentation
  - Description:
    Manuals, instruction booklets, training materials, etc. are one source of data about tasks. Furthermore, there may be corporate rule books and job descriptions which may be used to obtain information about tasks in a wider context. Manuals form a starting point for future analysis, and may be used to structure interviews. Rule books and job description can be used as part of an interviewing process, as a means of comparison against real job activities.
  - Advantages:
    - Documentation is the easiest way to obtain data about tasks
    - They are useful in providing basic actions and objects involved in a task (this will be discussed later on in section **3.2 Bridging the gap on the web**)
  - Disadvantages:
    - Job descriptions tell you how people are supposed to perform tasks, not actually what is being done.
    - Manuals are likely to tell you about the functions of a system rather than the way they are used in performing tasks.

        - Non-system-specific actions may not be described in documentations.
- Observation
    - Description:
      In order to get more insight in how tasks are performed, spending time watching people and maybe chatting with them can be useful. This can be used in combination with some existing documentation. Formal observation can then be used in the field or in the laboratory. Observation can be passive (listening and watching) as well as active (asking questions during tasks people perform).
    - Advantages:
        - A laboratory may be well-equipped (recording facilities, etc.).
        - Another advantage is the fact that the environment is interruption free.
        - In field studies situations may arise that would never occur in a laboratory.
    - Disadvantages
        - Costs for observations are higher than the cost of gathering documentation.
        - In the laboratory there is an evident lack of context. Unnatural situations may be simulated that would never occur in the "real" world.
        - Field observations may be harder to accomplish because of the surrounding environment (noise, interruptions, etc.)
- Interviews
    - Description:
      Questioning may be a way to get information about tasks. One would begin with a general set of questions describing some tasks. Then additional questions may arise like: Why would you do that? ; Wouldn't it be easier do this that way? ; And why (not)? ; etc. Interviewing can be a way to uncover details and to broaden the set of behaviors taking place during the completion of a task. Finally, this can be used to clarify some unclear points found during observation.
    - Advantages:
        - It is a direct and quick way to gather information about tasks
        - It is an easy way to clarify some unclear points (unexpected behaviors)
    - Disadvantages
        - Interviewing experts that are not the end-users may result in erroneous information about tasks
        - Too structured interviews may limit the range of topics discussed, because the interviewer didn't think about them.

All the techniques discussed above have their advantages and disadvantages. Using only one of those techniques may ultimately result in misleading views about the tasks being done. The best way is to use a combination of the three techniques. First of all, use documentation to gain some insight in the domain. Then use this knowledge to prepare your observations. And finally, use interviews to clear out some

ambiguous points. Of course this isn't always possible as time and costs come into play so a selection of techniques, using the list of advantages and disadvantages, may be more appropriate.

2.4 Task Modeling

Task modeling is concerned with documenting the data delivered by task analysis. It is concerned with the structuring of the data and gaining insight into the data. I'll discuss several methods for structuring the data. After that I'll discuss some ways to visually represent the data.

When I talk about task analysis in this section, I talk about documenting and structuring the data and not about task analysis the way it was explained in the previous section.

2.4.1 Techniques

Several techniques exist for doing task modeling among which task decomposition, knowledge-based analysis and entity-relationship-based techniques sometimes called ontology-based techniques. Each of those techniques focuses on certain aspects of the task world being modeled. The purpose of these techniques is not only a matter of documenting the data, but it is also used to communicate with other people and used for restructuring tasks.

As the name may suggest *task decomposition techniques* decomposes the tasks in subtasks. Let me explain it using the most influential task decomposition technique named *Hierarchical Task Analysis* (HTA). HTA outputs a hierarchy of tasks and subtasks (see **Figure 2 An Example of a HTA diagram**). HTA also produces plans describing in what order of sequence the (sub)tasks have to be performed and under which conditions they have to be performed. The plans need not to include all the subtasks. By rearranging the tasks and subtasks and by constructing new plans, one might come up with more appropriate and meaningful hierarchies. This is part of the process of HTA. The process is also an iterative one (more on this in **Chapter 5**). We first imagine the overall task being done. Using the data collected, we then build the hierarchy of subtasks. For each subtask we try to subdivide it further and further. We could go on almost indefinitely, and it is the task of the designer not losing sight on what is relevant for the task analysis. The designer should always have the intended purpose of the system in mind. Finally, the HTA could be presented to a domain expert in order to verify correctness of the hierarchies. But it is the nature of experts forgetting obvious things that should be included, so the designer should verify the HTA also by himself or with the user (as will be discussed in **Chapter 5**). The quality of the output finally will depend a lot on the skills of the designer.

Other techniques are called *knowledge-based techniques*. They begin by listing all the objects and actions involved in tasks, and then building taxonomies of these. The aim is to uncover the knowledge needed to perform certain tasks. Just as with HTA

hierarchies of objects are build and it is hard to know at which level of granularity we have to stop building the hierarchy. However starting with a list of all possible objects and then iteratively removing unnecessary object is the best way to handle this. Sorting the objects may also be hard to do. In this case domain experts or users can be involved to do the sorting using for instance sorting cards (like CRC cards in object oriented programming). Building taxonomies of the actions is done in the same manner as the objects (using hierarchies). The difference between the actions taxonomies and HTA may seem somewhat unclear. HTA is about sequencing subtasks in order to accomplish a task whereas taxonomy is about the similarity of simple task to one another.

Finally a third technique can be used. This technique is called an *entity-relationship-based technique*. More recently, *ontology-based techniques* emerged. The difference between these two only lies in the name used to describe them. Entity-relationship-based techniques are usually associated with database design. The difference is that in database design only the system entities are modeled. In task modeling non-computer entities are also modeled. Those techniques focus on the entities, the relationships between them and their attributes. The set of relationships and attributes considered important may vary from one technique to the other. Let me describe one technique discussed in (M. van Welie, 2001) which is an ontology using a minimal set of concept and relations. Figure 1 The Task World Onthology shows this onthology visually.

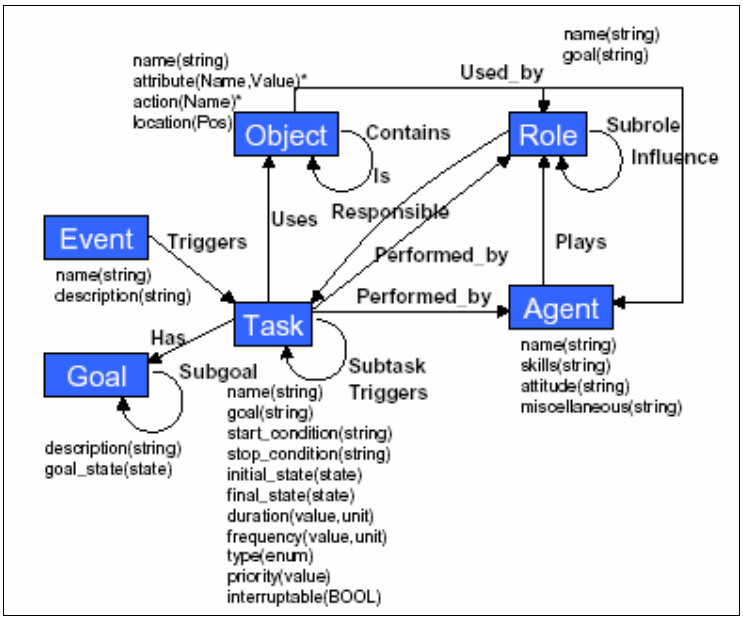


Figure 1 The Task World Onthology

The concepts fall in one of the following categories:

- Task: Tasks are performed by agents to reach a certain goal. Task can be decomposed into smaller subtasks. Tasks occur in a certain order and the

completion of certain tasks can trigger other tasks. Events occurring in the task world can trigger specific task.

- Goal: A desired state in the task world. It can be reached by one or more tasks. Goals may also have subgoals.
- Role: A role is a meaningful collection of tasks performed by one or more agents. Roles may also be hierarchically composed.
- Object: An object refers to physical and non-physical entities. Objects may have certain attributes. Object may perform certain actions. They may also be part of other object.
- Agent: An agent is an entity considered to be active.
- Event: An event is a change in the state of the task world at a point in time.

All these concepts are related in certain ways. The relationships are described as follows:

- Uses: It specifies which object is used and how it is used.
- Triggers: The triggers are responsible for describing the workflow. It specifies that a task is triggered by an event or by a task and how it is triggered.
- Plays: Every agent should play one or more roles. It also describes how the role was obtained (delegation, mandate, etc.).
- Performed by: It specifies that a task is performed by an agent.
- Has: This relationship connects tasks to goals
- Subtask/subgoal: It describes the task/goal decomposition.
- Subrole: The subrole relationship brings roles into a hierarchical structure.
- Influence: A role can influence another role.
- Responsible: It specifies a task for which the role is responsible.
- Used by: It indicates who used which object and what agent or role can do with it.

All the techniques described earlier may have several different methodologies that may differ in their way to describe the task world. Structuring data is not an easy process, different approaches are possible. Ultimately how the data is structured is not so important. What is important, is gaining insight in the task world being modeled.

In order to communicate and to document the knowledge gained representations are used. This is the topic of the next section.

2.4.2 Representations

Representations are used to document the data. Representations are also used for several other reasons:

- In order to communicate the data with other people (with clients, with designers, etc.).
- To analyze the data and to structure it and to be able to restructure it easily.
- It’s an easy way to compare several alternatives.

For task decomposition techniques, obviously task trees can be used. HTA for instance uses two sorts of representations: textual notations and diagrammatic notations. Textual notations should be used with care as long descriptions may become highly unreadable. Diagrammatic notations on the other hand are easy to understand and build. Although task trees can be powerful, they are mainly build on the subtask relationships. Figure 2 An Example of a HTA diagram shows an example of a HTA diagram.

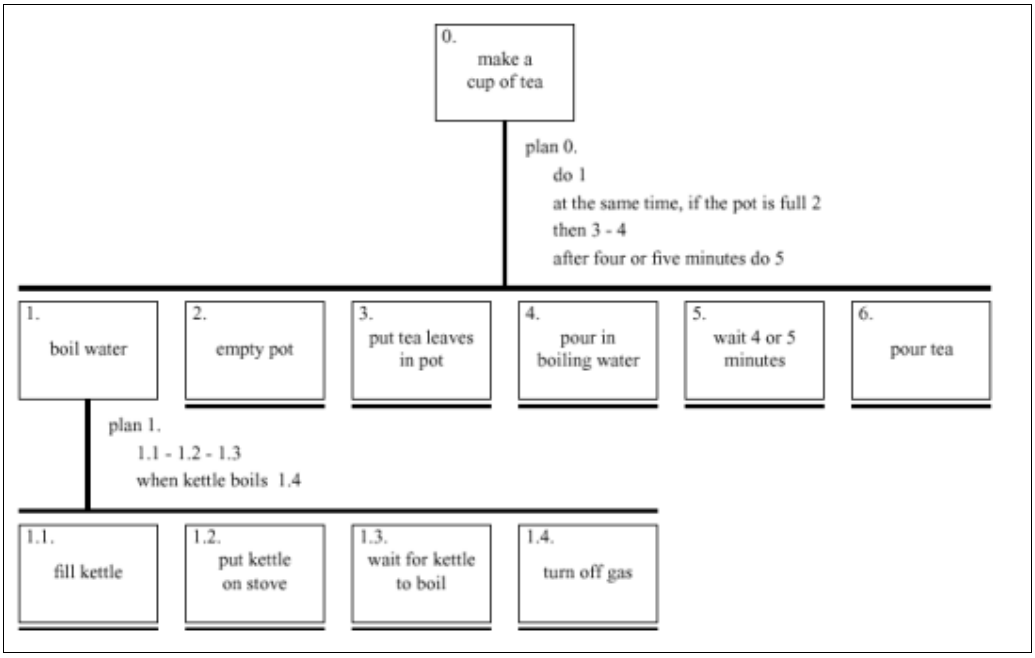


Figure 2 An Example of a HTA diagram

Other modeling techniques (e.g. CommonKads) use worksheets to model the tasks. Worksheets are templates which consist out of several properties (e.g. knowledge needed to accomplish certain tasks) that can be described. Such worksheets are useful when describing properties in detail. As the set of templates can be very large, they are not very useful when communicating at a high level. They are best used as a reference for detailed information about certain aspects of the task world.

When trying to document the data delivered by entity-relationship techniques, database modeling diagrams can be used. Several techniques can be used for this purpose like Entity-Relationship diagrams and Object-Role-Modeling (ORM) diagrams. Note that the UoD (Universe of Discourse) in database design is different from that modeled in task-based design. In database design the UoD is system specific, as in task-based design also non-system specific aspects (e.g. the user of the system) are modeled. Figure 3 Example of an ORM diagram shows an ORM diagram.

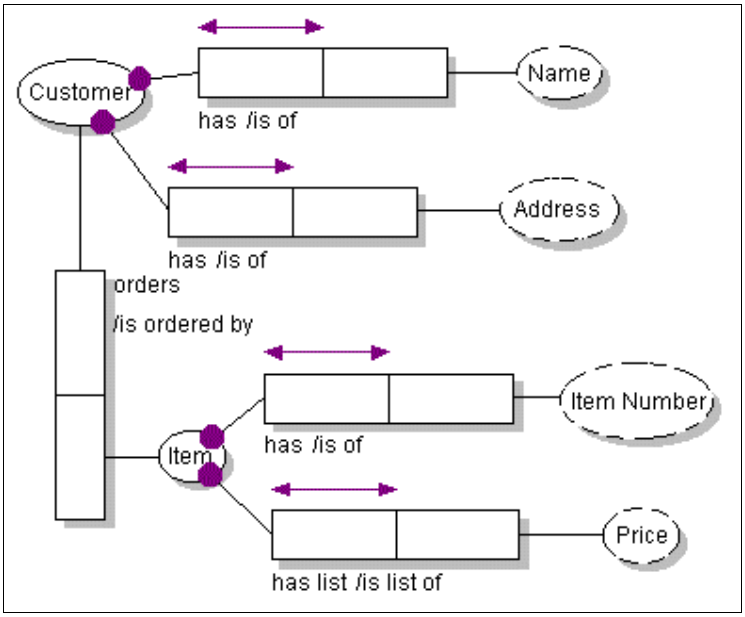


Figure 3 Example of an ORM diagram

Additional diagrams can be used to model the flow/sequence of actions. UML (Unified Modeling Language) wasn't designed with task-based design in mind. Nevertheless, the notational power of UML and its large acceptance can help in documenting the data. Activity diagrams can be used to model the flow of actions. Sequence diagrams can be used to track the sequence of tasks replacing message calls by tasks to be performed. Figure 4 Example of an Activity Diagram shows an example of an activity diagram.

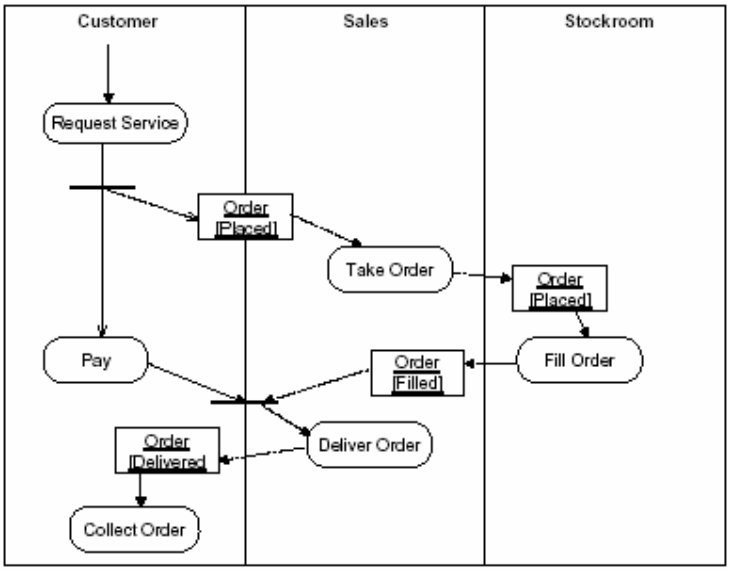


Figure 4 Example of an Activity Diagram

Other modeling techniques exist, and the list of techniques I discuss is not exhaustive and not the "best" techniques. When using some representations, one should always keep in mind the three points discussed earlier for selecting the right technique(s).

2.5 Conclusion

User-centered design is a relatively new research area when it comes to designing applications. It is even more true for the web, which is a vast free and uncontrolled environment. Failing to recognize the benefits of user-centered design may ultimately result in products which aren't used or badly used. In this chapter we also discussed the potential benefits of applying user-centered design principles.

Task Analysis is one of the processes involved in gathering the data about the users and about the task they fulfill. Several techniques for gathering this data were introduced. Although the techniques are very distinct, the best way of applying task analysis is by using a combination of the different techniques. Each technique has advantages and disadvantages. It is up to the designer (and maybe other stakeholders) deciding which are the best suited for a given domain.

Once all the useful information has been gathered, structuring the data becomes inevitable. This is called task modeling. Also here different techniques have been discussed. Each technique has its own way of structuring the data to reveal the interesting information needed in doing tasks.

Representations are useful for several reasons, among these: communication, comparing alternatives, restructuring tasks.

Once an initial Task analysis has been conducted and once the data has been constructed, we can start the design of our web site. This is the second step in the Task-based design process I discuss in this dissertation. Bridging the gap is a technique for mapping the tasks into an actual design. This is the subject of the next chapter.

3 Bridging the Gap

Until now, we've seen a major technique that can help in the design of sites: Task analysis and modeling gather the relevant data needed for the completion of tasks and structure the data in order to get insight in the data, communicate the data, etc.

The question that remains is: How do we translate the obtained data into a design? Interaction Patterns, which I'll explain in **Chapter 5**, can help bridging this gap but these patterns are not enough to make this translation. They can help improving a design; in other words they help making a web site more usable but they don't help translating tasks into the actual design.

So, how do we make the actual transition from the data to the design? This is known as "the gap".

3.1 Existing Methodologies for bridging the Gap

Several methodologies for bridging this gap exist, but the vast majority is focused on desktop interfaces and thus may fail to work in environment with other metaphors (see our discussion in section **1.2 Web vs Desktop**). I'll first give an overview of two well-known methodologies, which are intended for desktop interfaces and then I'll discuss a methodology developed at IBM which takes the hypertext metaphor into account.

3.1.1 The Bridge

The Bridge is an object oriented design methodology which can be divided in three stages as mentioned in (M. van Welie, 2001):

1. Expressing User Requirements in Task Flows
2. Mapping Task Flows to Task Object
    a. Identify which task objects need to be included in the system.
    b. Identify the total set of relevant attributes of these task objects.
    c. Identify relevant actions on the task objects. The actions can be ordered in menus.
    d. Identify groups of attributes so that only the task relevant task attributes are shown while performing a task. The resulting groups are called views.
    e. Identify object containment relationships. These relationships need to be represented in dialog screens.
3. Mapping Task Objects to GUI (Graphical User Interface) objects using a specific platform style.

More information of The Bridge can be found in (L. Wood, 1998).

3.1.2 OVID

OVID (Object, View and Interaction Design) is a similar set of techniques for designing object-oriented user interfaces and consists of three parts as described in (P. Azevedo et al., 2000):

1. Objects: these are objects that users perceive.
2. Views: the views are the views provided for the objects.
3. Interactions: those are the interactions that the user has with the objects.

OVID is applied once user goals, tasks and objects have been identified. Its input comes from requirements and task analysis, whereas its output consists of an abstract diagram that describes the architecture of the desired design, from the users' point of view. **Figure 5 The OVID Methodology** shows this process.

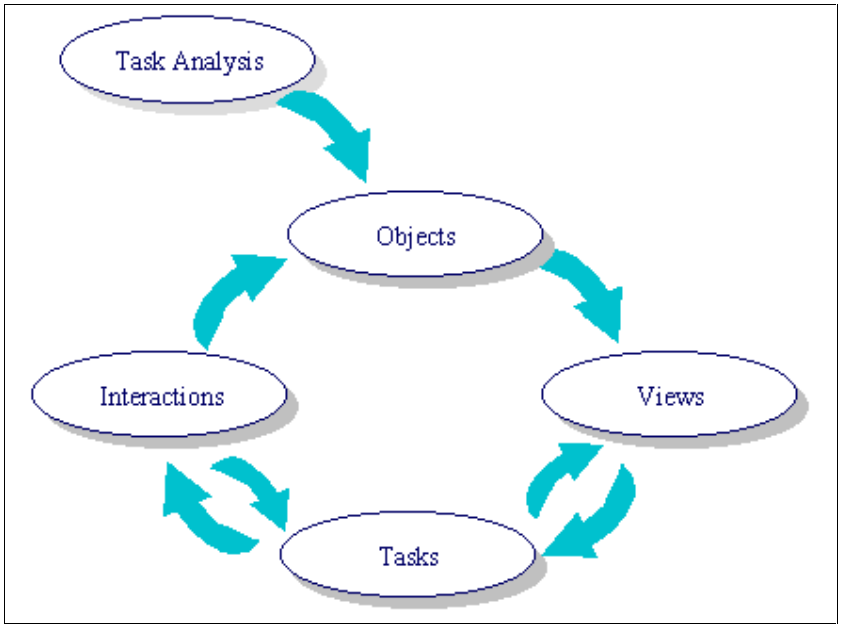


Figure 5 The OVID Methodology

Several other methodologies exist for this mapping but, as mentioned in (J. Kramer et al., 2001); […] however, none have previously been applied to web user experience development. A new methodology has been applied successfully in (J. Kramer et al., 2001) and I'll explain how it differentiates from the others with respect to web design. More information on OVID can be found in (D. Roberts et al., 1998).

3.2 Bridging the gap on the web

In this section I'll describe the design framework described in (J. Kramer et al., 2001). It is an adaptation of the Bridge and OVID, described in previous section, adapted for web site design. It has been proven to work, but at the time of writing this report the details weren't available yet. It has been extensively usability tested and yielded high satisfaction scores.

3.2.1 Conceptual User Modeling

Using the knowledge gathered during the task-analysis and task-modeling phases, we can start to model the users' mental model. Modeling this happens in four basic phases:

1. Each noun encountered is noted as a potential object.
2. Each verb associated with a particular object is treated as a user action (or task) on the corresponding object.

3. The attributes of importance to the user for each object are located among the previously identified object or are added based on the users' real world knowledge of those object.
4. Finally, we have to logically relate the objects with relationships that closely represent the way that the users understand these objects' existence in the real world. The primary relationship that this activity focuses on is containment.

Those steps are done to ensure that the user and interface can communicate well with each other. In order to fully uncover those aspects, the users need to be involved in this process. I'll explain this (the users' involvement) in **Chapter 5**, where I'll introduce an iterative process that unites the three basic steps of website design (task analysis and modeling, bridging the gap, and interaction patterns).

3.2.2 Abstract Views and interaction design

Once the object model satisfies the users' needs, the objects are translated into the interface. This is translated as follows:

- The objects are mapped into views. Views are specifications of the information needed by the users to do their tasks. Views do not specify the actual visualization, giving a designer enough room to use its creativity when designing the actual lay-out, etc.
- The attributes of the objects are then incorporated into the views. For instance, on a computer web site the attributes would be the technical information (CPU speed, amount of memory, etc.) concerning a specific computer (an object).
- Finally, methods are incorporated into the views to allow a user to accomplish a certain task. In the previous computer example a "buy"-method would allow a user to start the process of buying a specific computer.

Since the information in the views is determined by walking through all the interaction sequences that a user might encounter while executing the task flows, the abstract views guarantee that the design will support all the required tasks, regardless of the visuals chosen in the following step (J. Kramer et al., 2001).

3.2.3 The Web Framework

In **1.2** I mentioned the differences between a web interface and a desktop interface. Applying the translations rules of OVID and The Bridge into an actual interface can't be done on the web because of the metaphor shift of the web (J. Kramer et al., 2001). The main reason for this is the lack of a multiple windowing system support of the browsers because of the hypertext-metaphor, which has an impact on fulfilling tasks. Like I mentioned in **1.2** it has an impact on the context of tasks, the structure of tasks, and quitting tasks. Instead of having multiple windows, browsers use view replacements. So we only see one window at the time in fact.

Object Views

The views are translated into a visual view as described in **Table 1 Visual Treatment**.

Abstract element	Visual Treatment
Object Identity: unique characteristics	Color, bold font, signature, background
Object Identity: view delimiting boundary	Explicit or implicit visual boundaries: separator lines, frames, or spacing between colored backgrounds
Composition relationship	Geometric containment
Grouping: similarities, closeness in domain, compatibility	Spatial grouping, other positioning cues
Domain-specific methods	Rounded buttons
View-access methods	Tabs and tab-like buttons

Table 1 Visual Treatment

Those visual cues don't need to be specifically the same as given in **Table 1 Visual Treatment** (e.g. squared buttons instead of rounded buttons), but it gives a fairly good idea what should be focused on. It is to the designer to decide which cues it will use and which it will adapt.

The first two abstract elements are used to differentiate different types of object (views) and to clearly show what belongs to an object (e.g. attributes) and what doesn't. In **Figure 6 Example of Object Identity** a display object (in this case Flat Panel TFT Active Matrix) has a view with some attributes in it. It would be differentiated from a processor object, because of the delimiting boundaries and the object name (Display).

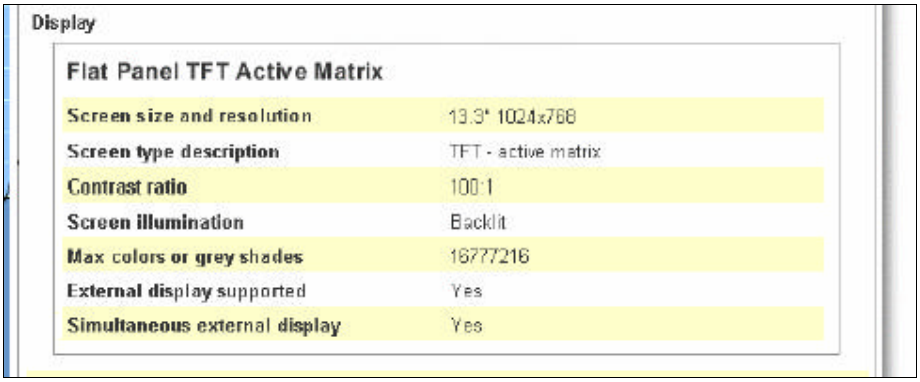


Figure 6 Example of Object Identity

There are also several relationships between information entities (e.g. objects) possible. Among those relationships we find: compositions, aggregations, etc. The relationships between an object and its attributes are treated like compositions (in the UML sense). This is expressed by using geometric containment.

Containment is key to navigation because locating information often amounts to opening a series of container objects until the desired object is located. […] Such a

layout is intuitive, but can be challenging to implement dense, spatially constrained situations (J. Kramer et al., 2001).

For instance, **Figure 6 Example of Object Identity** could be contained in a container object Notebook and would have siblings like Processor, etc. As an illustration I have taken the example found in (J. Kramer et al., 2001) and more information can be found there as well; see **Figure 7 Example of geometric containment**.

Figure 7 Example of geometric containment

Actions on Objects

In **1.2** I mentioned the heterogeneity problem. The file-menu, view-menu, back-button etc. don't belong to the web site, but belong to the browsers. And as previously mentioned, browsers don't provide multiple windowing systems.

So there needs to be some way to perform actions without confusing them with the features provided by the browsers. Those kinds of actions are called methods in object oriented interfaces.

In web design two kinds of methods are distinguished: context specific methods and view access methods. The context specific methods are the actions an end-user performs on a particular object. It should be located next to the object and should be represented as a button. The view access methods allow switching from one view to another (to faint a multiple windowing system). They allow to switch from one view of an object to another view of that object and are represented by tabs.
Actions on objects are explicitly not represented as underlined words, to avoid the hyperlink metaphor and to explicitly evoke the kinds of metaphors used in a desktop application. An example of this is given in **Figure 8 Methods on a web page**.

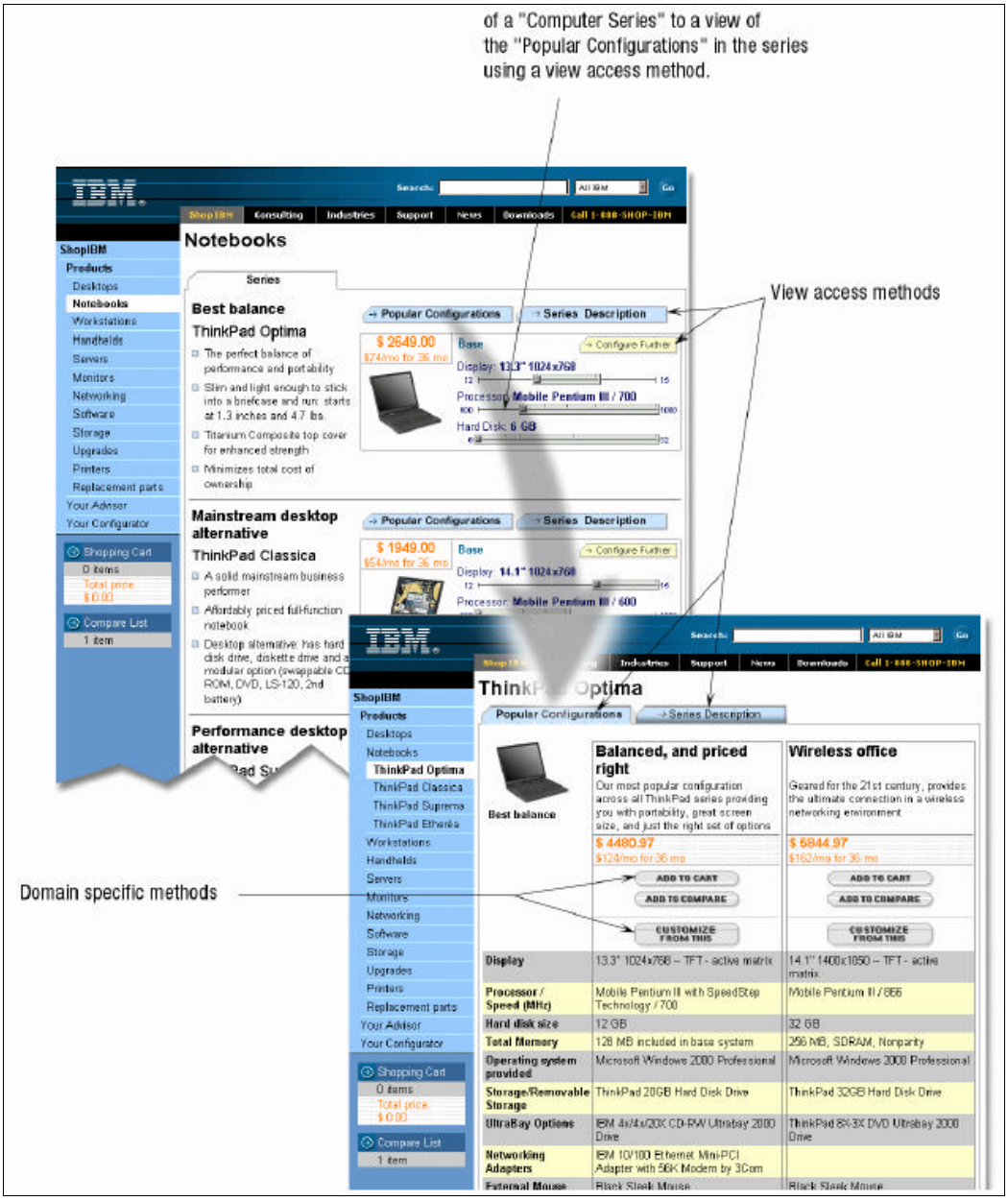


Figure 8 Methods on a web page

3.3 Conclusion

On the web, as I mentioned in **Chapter 1**, the metaphor used is a hypertext metaphor, which is not well suited for completing complex tasks except navigation. The attention of the designer should be focused on the user and not on the navigation of a web site. In this chapter an adapted version of The Bridge and OVID, which are object oriented methodologies for designing desktop interfaces, was discussed where the real focus is concentrated on the user and the tasks he has to complete. This is achieved by defining a user conceptual model, which is the result of analyzing the task modeling phase and its data. Then once the user mental model has been

established, the mapping of this model into an actual design is done which matches the users' mental model.

In the next chapter we will see a technique called interaction design patterns, which helps designers improving the design and which helps designers in making some design decisions based on best-practices.

4 Interaction Patterns

"The pattern is, in short, at the same time a thing, which happens in the world, and the rule which tells us how to create that thing, and when we must create it. It is both a process and a thing; both a description of a thing which is alive, and a description of the process which will generate that thing."

-- Christopher Alexander

The step described in previous chapter helps a designer mapping the tasks into an initial design. In order to improve this initial design, I introduce Interaction Patterns. Interaction Patterns are much like the Object Oriented Software Design Patterns, in that they are pieces of proven knowledge that designers should strive for in order to improve the design.

I'll first introduce patterns to give some background. Then I'll introduce the Interaction Patterns and discuss their benefits, but I'll also consider the critique that has been made by several researchers towards these patterns.

4.1 Patterns

Here I'll start with a little bit of background and then I'll introduce the concept of patterns as known in Computer Science.

4.1.1 A Brief History

The current[4] use of the term "pattern" is derived from the writings of the architect Christopher Alexander who wrote books related on the topic of urban planning and building architecture.

Although these books are ostensibly about architectures and urban planning, they are applicable to many other disciplines. Alexander wanted to create structures that are good for people and have a positive influence on them by improving their comfort and their quality of life. He concluded that architects must constantly strive to produce work products that better fit and adapt to the needs of all their inhabitants and users and their respective communities (B. Appleton, 2000).

The ideas of Christopher Alexander became popular in Computer Science, when, in 1987, Kent Beck and Ward Cunningham were working with SmallTalk and designing user interfaces. Using his ideas, for guiding novice SmallTalk programmers, they developed a small pattern language.

Later on, Jim Coplien compiled a catalog of C++ idioms[5] and published them.
From 1990 to 1992, a few members of the Gang of Four[6] (GoF) had done some compiling a catalog of patterns. From then on several workshops were organized, some of them during the OOPLSA conferences.

Shortly thereafter the book of GoF was published and since then patterns are widely accepted in the Computer Science community.

Beside software design patterns, a lot of other kinds of patterns were elaborated in the field of Computer Science. Martin Fowler wrote a book on Analysis Patterns. Organizational patterns also exist, etc.

More recently, new kinds of patterns have emerged in the field of Human-Computer-Interaction. These patterns are focused on the interaction a user has with a specific user interface. More on this will be given in the following sections.

More history related reading can be found in (B. Appleton, 2000).

4.1.2 What is a pattern?

So, what exactly is a pattern? From the literature it is clear that there isn't a consensus about the definition of a pattern. Instead of trying to give my own definition, I'll give several descriptions found during my research.

[4] The meaning of the term "pattern" as is used nowadays in Computer Science.
[5] A low-level pattern specific to a programming language.
[6] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides.

Description 1:
Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it in the same way twice.
by Christopher Alexander

Description 2:
Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves.
by Richard Gabriel

Description 3:
I like to relate this definition to dress patterns. I could tell you how to make a dress by specifying the route of a scissors through a piece of cloth in terms of angles and lengths of cut. Or, I could give you a pattern. Reading the specification, you would have no idea what was being built or if you had built the right thing when you were finished. The pattern foreshadows the product: it is the rule for making the thing, but it is also, in many respects, the thing itself.
by Jim Coplien

In other words, a pattern can be seen as a best practice to a recurring problem. But it is much more than that:

- It occurs in a given context. In this context competing concerns may occur. So it is also a solution which tries to accommodate these concerns in an appropriate manner.
- It captures the essential elements so that people can learn from it. And by those means, it helps people using them in similar situations.
- A pattern has a name. It helps people talk about patterns and what they represent.
- A pattern also explains why a solution is needed. This will in many respects improve the insight people have in the domain they're dealing with.
- Patterns are also used for structuring parts of a greater whole. For instance, the design patterns of GoF have relationships between them. Those relationships can be used to build bigger parts of a software system.

From these characteristics it may be clear why nobody has come up with a well defined definition of the term "pattern".

4.2 Interaction Patterns Explained

As mentioned in (J. Tidwell, 1999) there is plenty of good literature out there on the high-level principles of good interface design, and it is getting ever better as this young field matures. […] But if you're a novice designer, it's hard even to remember all these principles [gentle error messages, protection from accidental mistakes, etc.], let alone use them effectively! And it's difficult sometimes to make the tradeoffs among these principles when they come into conflict; we often have to figure out the best solution by guessing, or by resorting to other means.

Given the background of previous sections, the use of patterns in human computer interaction should be obvious. The use of patterns helps in applying designs where many requirements compete against each other.

I'll explain in the next section why exactly patterns can benefit the world of human-computer interaction and even more in the context of the web.

4.2.1 Benefits

In the literature (M. van Welie, 2001), (R. N. Griffiths et al. 1997), (J. Tidwell, 1999), etc); the advantages of interaction patterns are expressed by comparing them to guidelines and style guides. On the web it becomes even more important to consider interaction patterns as a viable alternative to guidelines. Essentially this can be explained by the fact that there isn't a consensus about the guidelines to use when moving to the web. The web is also a heterogeneous space, independent of any platform, so there isn't any need to have a common style between sites. Also most of the guidelines on the web are tied to certain types of technologies[7] which make these guidelines difficult to apply over time or there is even a lack[8] of guidelines surrounding a technology.

Other benefits of patterns can be stated as follows:

- **Richer resource:** the information surrounding a pattern is much richer than the information recorded in guidelines. Problems are stated, a context is given, relationships with other patterns are mentioned, etc. It is important to know why, when, and how a particular pattern should be applied. Guidelines don't or merely offer some hints.
- **Focus on the process:** patterns can evolve over time. They can be refined to more closely approach a best-practice. For instance in the world of object oriented design, the patterns of GoF have been augmented and refined in (J. Vlissides, 1998). In the case of guidelines, such things are achieved by simply recording more cases.

[7] The Web Design Guidelines of IBM recommend the use of CSS style sheets: http://www-3.ibm.com/ibm/easy/eou_ext.nsf/Publish/572

[8] Flash MX doesn't provide any guidelines. But due to criticism, guidelines for Flash MX are on their way. Nielsen Norman Group is creating Flash MX Guidelines by now.

- **A pattern language:** the relationships between the patterns make navigating between them easier than with guidelines.
- **Communication:** it doesn't only ease the communication among designers, but it also, by its language style features, allows involving users in their design.

As mentioned in (M. van Welie, 2001) guidelines have often also been reported to have a number of problems when used in practice, e.g. guidelines are too simplistic, too abstract, too difficult to interpret and select, etc. Guidelines are usually defined by closed groups which may be arguable because they aren't the product of some empirically discovering process.

Other benefits of patterns have already been mentioned, this non-exhaustive list is a means of comparing it against guidelines.

Although there hasn't been done extensive research about the usefulness of patterns in the domain of human-computer interaction (HCI), interesting and encouraging results has been made. In (J. Borchers, 2002) statistics were gathered about how the usefulness students of HCI courses found concerning patterns. The questions concerning the patterns were:

1. I remember the following HCI patterns[9]:
2. For the overall understanding and remembering of user interface design concepts, the patterns were (1 = very useful … 5 = completely useless).
3. I was able to find problems and solutions for our own design project in the pattern collection (1 = absolutely … 5 = not all).
4. I can imagine using this pattern concept in future design projects (1 = certainly yes … 5 = certainly not).

Figure 9 shows the results of the survey. The number of patterns remembered had an average $\mu \approx 1.73$ patterns, with a standard derivation of $\sigma \approx 1.65$. According to (J. Borchers, 2002) this looks quite promising considering the relatively short time spent during the lecture[10]. The usefulness of the pattern language was rated with the second-best grade possible, giving an overall $\mu \approx 1.96$ and a small derivation of $\sigma \approx 0.65$ indicating a high-level of consensus. The usefulness for the project was a little bit worse with a second best rate ($\mu \approx 2.23$) and a slightly higher standard derivation ($\sigma \approx 0.89$). Finally, the confidence that the pattern concept would be reused in future projects was again quite high ($\mu \approx 1.94$), with relatively great consensus ($\sigma \approx 0.81$).

Then (J. Borchers, 2002) concludes that patterns also have a great potential for teaching, not just as a topic alone, but particularly as a tool.

[9] The studied patterns collection was J. Tidwell's Common Ground Collection which contains about 50 documented patterns. http://www.mit.edu/~jtidwell/interaction_patterns.html

[10] The survey was made 2 weeks after the lecture had been done. And the students studied the collection during 15 minutes in that lecture.

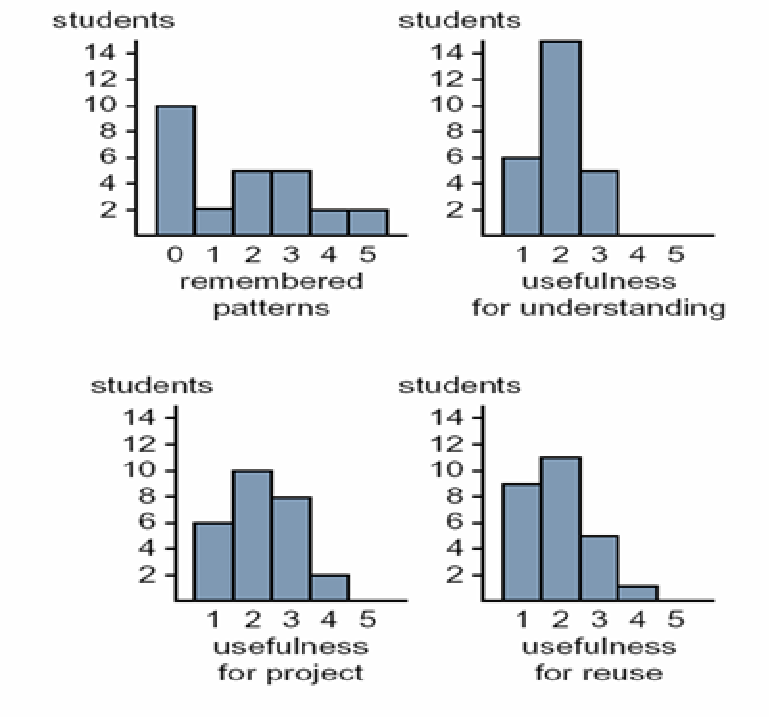


Figure 9 Results of the patterns survey

4.2.2 Critique

Although interaction patterns can be traced back to 1994 (D. Rijken, 1994), some criticism has been made towards interaction patterns. This is partly due to its lack of maturity. The criticism is not specially directed towards their use but more to the organizing principle surrounding those patterns. I'll give an overview of the 3 major obstacles towards an effective usage of patterns as found in (A. Seffah et al., 2002):

1. The human-computer interaction community has no universally accepted standard or format for the documentation of patterns. In other words several languages for specifying interaction patterns co-exist. The presentations of the patterns changes from one writer to another. This has some implications: authors willing to write patterns have to make a choice between the several representations; readers have to get used to the several formats and this may ultimately confuse them.
2. There is no tool support for interaction patterns engineering. Tool support can enhance the patterns user' understandability; decrease the complexity of a pattern language; and eliminate terminological ambiguity. It puts the language into practice in a real environment.

3. Using interaction patterns for different platforms (web environment, desktop, pda[11] environment, etc.) when they're expected to complete the same tasks can be challenging.

The second critique is much more a consequence of the first one. Building support tools should only make sense when a consensus is achieved on a particular pattern language. Furthermore, it takes some time for such a tool to breakthrough. It is not that long ago that the design patterns of GoF made a breakthrough in the most commonly used tools (e.g. TogetherSoft[12]). Finally there is some research made in that area (S. Henninger, 2002) which makes the prospects of such tools not desperate.

I'll investigate the other critiques in the following section, where I'll explain one of the patterns languages that exist and why it is well suited for web design.

Nevertheless, careful attention should be paid to those critiques if a commonly, highly usable pattern language should be created or standardized.

4.2.3 A Pattern Language

During my research I came across several pattern collections. In this section I'll present the pattern language described in (M. van Welie, 2001) and I'll also explain why this language is my language of choice when it comes to web site design. The pattern repository can be found on Martijn van Welie's website[13] and two example patterns can be found in the **Appendix**.

The structure is as follows:

Pattern Name

Each pattern should have a meaningful name that relates to the problem or solution. Ideally they should be abstracted away from user interface details (like widgets, etc.). They should encompass the structure and knowledge of the pattern. They are also a good means of communicating design issues.

Problem Description

While surfing the web users can have all sorts of problems. The description states the user problem the pattern is trying to solve and the objectives it is trying to achieve in the given context. The problems described are typically task-related problems.

Usability Principle

[11] Personal Digital Assistant
[12] http://www.togethersoft.com/products/controlcenter/features/patterns.jsp
[13] http://www.welie.com/patterns/advanced-search.html

Much like the GoF patterns use categorizations to group patterns, the usability principles categorizes the interaction patterns. This categorization is based on usability principles (e.g. metaphors, feedback, error correction, etc.).

Context

The context provides information about when a particular pattern should be applied. It states the conditions under which the problem occurs, and gives insight in the solution which seems to fit in this set of conditions.

Forces

Forces state why a user could use a specific pattern. This can be highly beneficial to designers because they state why a specific pattern should or should not be used. The "why"-part is one of the reasons of using patterns over guidelines. Using guidelines just because they are guidelines, may ultimately result in an unusable design.

Solution

It describes the core of the solution. It also provides visual support that shows how the problem is solved. The solution is abstracted, in order to be able to not specify the specific design details. Finally, the solution starts with a one-liner that summarizes the solution.

Rationale

Rationale in essence motivates the "forces" of a pattern. It states how, and why it achieves the desired goals (or the forces). It gives a reasonable argumentation about the use of the pattern. The argumentation is based on psychological, sociological, ergonomic findings, etc.

Example

It provides examples of interfaces where the patterns have been used (consciously or not). It helps users in understanding a pattern and enforces the fact that the pattern is a proven solution.

Counterexample

Much like anti-patterns can describe situations where given solutions are bad, counterexamples provide information about bad designs. It also motivates the use of the pattern. It is an optional section of a pattern.

Known uses

Besides the example section, this section provides additional examples where the pattern has been used. It gives the opportunity to consult different designs based on the same pattern.

Related patterns

This section references patterns that address related problems. It allows designers to quickly find a set of interrelated patterns. This may ultimately result in a design which solves given problems in a broader context.

Category

This is not mentioned in (M. van Welie, 2001), but is used on the web site and it is not really a sections, but is used primarily to help designers in choosing a pattern by guiding them in the right direction. It states which kind of task-related problem a particular pattern solves. Four kinds of categories exist by now, namely: navigation, searching, page elements and E-commerce. This categorization is not definitive and will probably evolve over time as the collection grows.

As I mentioned earlier, several pattern languages already exist. The most important ones as mentioned in (K. Mullet, 2002) are: van Duyne's collection, Borchers' collection, van Welie's collection, and Tidwell's collection.

The main reasons of using this pattern language over others can be stated as follows:

- **Availability:** at the time I wrote my thesis I only found two available collections online, namely: Tidwell's collection and van Welie's collection. In order to spread good design practices to everyone, to professional as well as non-professional designers, making a repository freely available can contribute to more and better user experiences.
- **Audience:** van Welie's collection is not specially geared towards "customers", but to "users" in general. This makes the collection available to the professional designer designing a corporate web site, but makes it also available to an internet user who wants to build a usable homepage.
- **Size of the collection:** van Duyne's collection and van Welie's collection have specific large web site collections. Tidwell's collection also includes some patterns which could be used in a web environment, but is not sufficiently large.
- **Visuals:** the four collections use visuals to support the patterns. Van Welie's collection uses a specific kind of visual: wireframes. Wireframes are skeleton graphics describing a site's lay-out visually, but not the graphical design of a web site. The fact that it is not a hand-drawn graphic adds clarity to the patterns. Further on, the wireframe can be used as a basic template of a site. Besides the wireframe it also includes screenshots of well-known sites to clarify the patterns.
- **Support for different types of user interfaces:** van Welie's collection not only supports web site patterns, but it also supports mobile and desktop patterns.

Another reason of using this collection is that Martijn van Welie is an active contributor to this community: he organizes workshops, provides a freely available

growing collection, etc. This is definitely something this community needs and will maybe ensure the perennity of the collection or at least it will contribute to the creation of a new standardized pattern language.

4.2.4 Additional Considerations

Computer science compared to older disciplines (architecture, electrical engineering, etc.) is quite new. The discipline of human-computer interaction is even less mature. (R. N. Griffiths et al. 1997) mentions that there are few universally acclaimed design solutions and this may mean that it will be easier to identify problems than to describe the solutions that address them. It may be that the discovered patterns in interaction design may not deserve any attention at all. For instance, Jakob Nielsen criticized the use of frames on the web in (J. Nielsen, 1996). On the other hand some patterns may need to use frames to improve users' usability experience.

Christopher Alexander said about patterns that they should be timeless. Nowadays, we talk about scroll bars, windows, frames, etc. for enhancing the users' experience and for making websites, and software more usable. The interaction patterns don't share this timeless aspect, yet. It may take some time achieving this because it is a quite new discipline and it is even questionable if achieving this can be useful. Only time will tell this.

4.2.5 Conclusion

In a heterogeneous and ever growing environment as the web, it becomes more and more important to consider the users' usability issues. The use of patterns can make users' experience more enjoyable in many ways because of several reasons: patterns help understanding users' problems, which may result in better interfaces; patterns evolve over time, so they can even evolve as new technologies are created; patterns by their interconnectedness can even help designer consider problems they didn't think about; patterns can also spread good design more efficiently because of its language-like features; etc.

Interaction patterns have a lot to offer; nevertheless we must recognize that they aren't mature yet. It will take some time before they get mature, and the best way to achieve this is by using them. Even in architecture the language is still evolving.
Until now we've seen three major methodologies. In next chapter, I'll explain how we unify these three steps into one process. This process is an iterative process, and besides unifying the three steps in this process I'll also explain how the iterative process should be applied with respect to the users' tasks.

5 The Iterative Process

Until now we've seen three major techniques which can help in the design of web sites. Task-analysis is a technique which helps in gathering the data relevant with respect to the user. Then we have seen a technique that helps bridging the gap from the analysis phase to an initial design. Finally, we explained how Interaction Patterns could help designers in making more usable web sites.

In this chapter I'll explain how those three techniques could be combined to form one integrated design process. At the end of the chapter, I will explain how the various steps should be applied from the early phases (the analysis phase) to the detailed design (which is a conjunction of the design delivered by the gap-technique and applying patterns to improve the design). In order to help the designer in the various stages of the iteration process, I will also introduce four criteria he should consider during the process.

5.1 The Model

In this section, I'll describe an iterative process unifying the various steps I discussed until now: task analysis, bridging the gap, and the use of Interaction Patterns. First of all I'll discuss the model behind it, which has been inspired by the water fall model used in software engineering. In subsequent sections, I'll describe the transitions between the various steps.

Doing iterative design allows a designer to set measurable goals with respect to the design. Each iteration refines the prototype, until the final design is ready for implementation. The goals of each iteration are related with the different aspects of the tasks that I'll explain in section **5.2 The Iteration Cycles**. Not every aspect of a

task should be considered in early phases, neither should some aspects be postponed to the end of the design unless necessary.

Here I propose an iterative model which describes how we have to make the transitions between the three methodologies.
I would also like to note that the reason for introducing an iterative process is quite simple; iterative design is introduced because no design team is perfect and no design team can design a web site right from the first time.

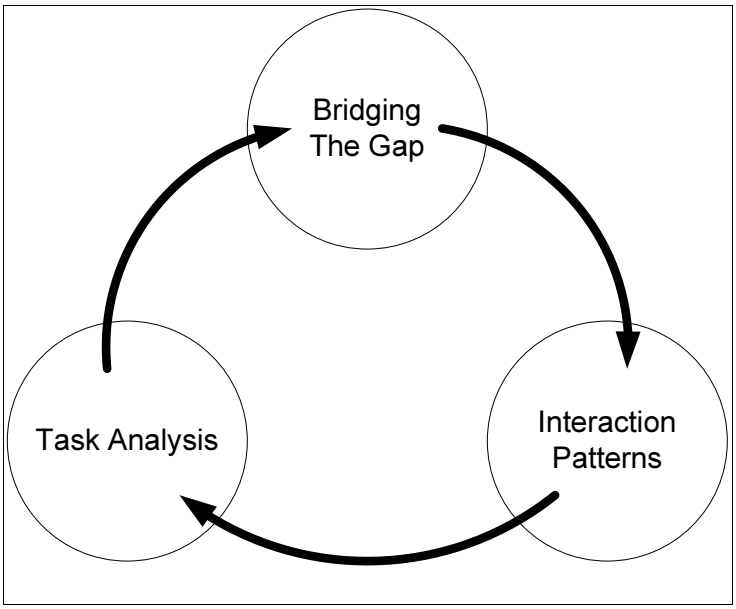


Figure 10 The Iterative Process

The iterative model is represented in **Figure 10 The Iterative Process**. The process is fairly simple and I'll explain the various steps in upcoming sections. I won't focus on the transition from the Task Analysis to Bridging The Gap because it is the mapping technique I already explained in section **3.2 Bridging the gap on the web**.

The first step that has to be taken in this process is the task analysis; following the model, we make an initial design by using the technique described in **Chapter 3**. Then this initial design is improved by applying Interaction Patterns on it. In the last step, we will check the improved design against the gathered data among other things (see **5.4 From the Interaction Patterns to Task Analysis**).

In the remainder of this chapter I will illustrate my process with an example. This example is based on the popular web site known as Hotmail[14]. The Hotmail web site provides three high level tasks:

- Consulting e-mails
- Composing and sending e-mails

[14] http://www.hotmail.com

- Managing an e-mail address book.

Now that we have seen the basics of the iterative process, I'll introduce four task criteria which designers should consider at the various cycles of the iterations.

5.2 The Iteration Cycles

Doing iterative design is not just a matter of applying steps one after another. In order to help the designers focusing on the right tasks during the various cycles of the iterative design, I introduce four criteria that play a role at the various stages of the iterative process.

First of all I'll discuss the various criteria and then I'll introduce a timeline for helping designers focusing on the right criteria during the lifecycle of the iterative process.

5.2.1 The Tasks Criteria

In **Chapter 2** we saw how we could gather the user data and how to structure it. But selecting the aspects of tasks which are relevant is also critical.
Now, we concentrate on those criteria and not the iteration phase they belong to. The criteria described here are: level of detail of the tasks, level of being representative, priority of tasks, and completeness of higher level tasks.

Level of detail of the tasks

One of the relevant criteria of a task during design is its level of detail. Designing a task that for instance doesn't include all the needed subtasks can be problematic, because it may probably interrupt the task flow, which obviously won't satisfy the user.

I'll illustrate this with a short example:
Let's say person X wants to send an e-mail to somebody. Person X unfortunately doesn't know the address by hart, but he has added the contact information into the address book provided by the site he is using for sending and reading e-mail. The scenario goes as follows: He gets on the site and starts composing his mail. Once it is composed he realizes he forgot the contacts' e-mail address. Therefore, he goes to the address book and looks it up, but unfortunately the address book is not on the same page. Once he gets back on the initial page all information is lost. He has to start the process all over again.

This wouldn't have been problematic on a desktop application. For instance Outlook supports multiple windows, there you could easily go to your address book and check the address and add it to the mail. We wouldn't have had this problem if the subtask of checking a persons' e-mail address would have been included in the same window as the one of composing an e-mail. Hotmail for instance solves this problem, by

providing a quick-list of e-mail addresses and also by providing a pop-up box when you ask for it; see **Figure 11 The address list and the pop-up window of hotmail**.

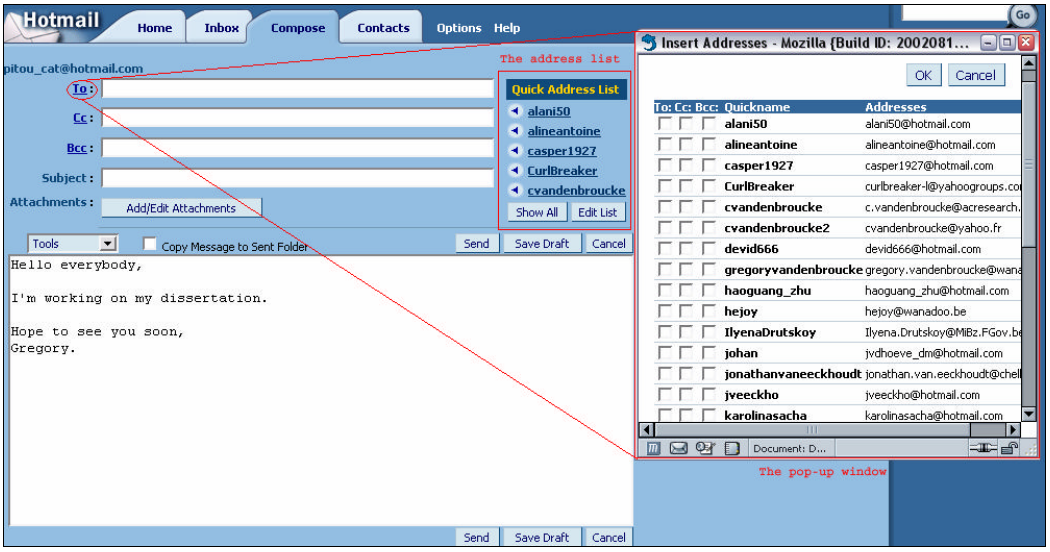


Figure 11 The address list and the pop-up window of hotmail

Other subtasks, like saving a draft of an e-mail, could also have been added if the users wanted to. The level of detail of a task will probably be uncovered after subsequent iterations (more on this in **5.2.2 The Timeline**).

Level of being representative

Tasks included in the design should be representative for what users do or want to do. They should be real tasks that users normally do, or tasks they want to accomplish when the new web site comes into place. Adding tasks because it would be “a nice feature”, isn’t the right way of designing and would probably clutter the interface and therefore it might hinder the normal execution of other tasks which are essential. On the other hand, it is also easy to forget obvious tasks that don’t seem to be part of the task flow. For instance, a company has an intranet system which archives documents. They want to move to the web to give other people the opportunity to consult those documents online. Initially the company needed to send the documents via e-mail, and because they were big, everybody printed those documents out. It would be good to give people the opportunity to generate a printer-friendly version of those documents on the web site, so they could print them out and read them on paper. This simple ad-hoc task could be easily forgotten because it isn’t easy to discover and typically only applies for the web. It is important to always question the level of being representative and to consult the users about this.

Priority of the task

The first steps in the design process should be focused on the most common or important tasks. The first iterations of the design process should be focused on the tasks that are done frequently (see **5.2.2 The Timeline**). The reason for doing this is

to be sure that the design at least supports those tasks that are important. Then as the iterations go by in the design process, the designer can focus on the less important tasks and maybe will even uncover new helpful ad-hoc tasks. During the iteration cycles, the design can then evolve around the tasks that should be supported by all means. Adding the most important tasks in a later phase, will inevitably have an impact on the design and maybe will even force the designer to re-design big parts of a web site, which from a time and money point of view is surely inadequate. For instance, let's illustrate this with the previous example of an e-mail service. One of the important tasks is sending e-mails. So we start designing the text-fields as given on **Figure 11 The address list and the pop-up window of hotmail** and we also add the send-button. A few iterations later, the designers see there is a lack of support for finding the e-mail addresses of contacts without breaking the task flow. Adding this ad-hoc task will be less expensive, than having to re-design a page because the designer didn't concentrate on the important or common tasks at first.

Completeness of higher level tasks

Besides considering the important tasks it is also crucial to think in terms of completeness of tasks. Even if not all the subtasks or ad-hoc tasks have been discovered, it should at least be clear what the complete set of higher level tasks is and how they form a whole. This whole forms in essence your initial site design; it is the back bone of your site. Just like a building needs a solid foundation in order to be able to "grow", a site design needs solid foundation in order to support additions easily. For instance, let's go back to the e-mail service. One of the tasks is sending an e-mail. First of all, I have to go on the site, then I have to log in, then I have to write my e-mail, then I have to send the e-mail and finally I sign out. What about the structure of the site? Obviously I will need a welcome screen (the log in screen), then I will probably need a page to write and send my e-mail, then I'll have to sign out. If another high level task was managing my address book I would have to make various steps and these would help me in uncovering the site structure more easily and also will assure that the various tasks fit more easily without getting in the way of each other in a later iteration. Note that I didn't mention the finer-grained tasks like sending the e-mail by pushing a send-button effectively, etc. the focus is on the relation between the complete tasks. If we didn't take into account those relations and start designing them independently, we might end-up with an awkward design.

In the next section, I will explain how those various criteria play a role with respect to the iteration cycles and especially when those criteria should be considered.

5.2.2 The Timeline

I mentioned several criteria which we should consider when doing iterative design. The various criteria were:

1. Level of Detail of the tasks
2. Level of being representative
3. High priority tasks

4. Completeness of the higher level tasks

One of the questions that come to mind is when we have to consider those aspects when doing iterative design. Here I propose a solution for integrating those criteria in the iterative design.

Figure 12 The Iterations Timeline shows the iterations timeline and how the various criteria fit into it.

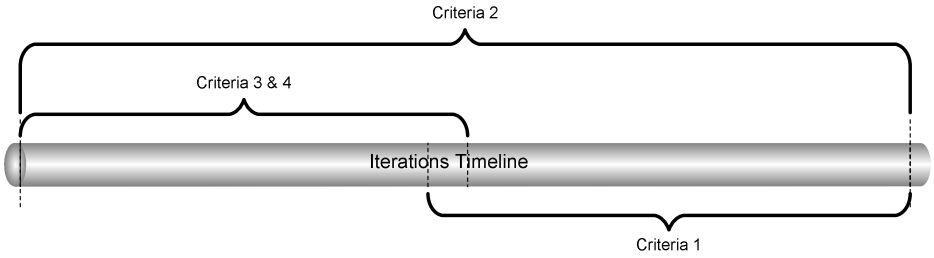


Figure 12 The Iterations Timeline and the criteria

Let's discuss this:

- **Criteria 1 Level of Detail:** Starting the first iteration by trying to address low-level details can be problematic. First of all, the design focus risks heading in one specific direction, and therefore designers will probably miss the overall scope of the design. Secondly, adding finer grained details at the end of the iterations are less expensive than adding higher level tasks. Finally, the designers will probably, during the iterations, uncover new useful details which should be added later.
- **Criteria 2 Level of being representative:** This aspect should be kept in mind during all the iterations. As I already explained adding features just because they seem nice isn't a proper way of doing design. If this feature really supports a users' task or if the designer knows it would help in accomplishing a task, it would be better to wait the next iteration and add it in case the user approves it. As the iterations go it could also be possible that you discover things that weren't part of the initial task analysis, but that should be designed (e.g. the printer-friendly version of documents example).
- **Aspect 3 & 4 High Priority Task and their completeness:** Those two aspects go hand in hand in the early stages of the design process. The first reason of adding high priority tasks in the first phases is that the lower level tasks should evolve around the high priority tasks not the opposite. Imagine first designing the lower level tasks and then adding the important tasks in the end; the low level tasks will get in the way, resulting in an awkward design; which means extra iterations for setting this right. Another reason for paying attention to those two aspects in the early phases is that they will reveal the core of the web site. This is not a bad thing because the main design of the web site will be clear to the user and his feedback will tell us if we're going in the right direction.

5.3 From Bridging the Gap to Patterns

In the iterative design process (see **5.1 The Model**), we first analyze the tasks (task analysis) that have to be supported, then using the bridging the gap technique we come up with an initial design by mapping those tasks. As I mentioned in the chapter about Interaction Patterns, patterns can be used to improve the web site design and also to better understand the users' needs; among other things. Here I will explain how to use the patterns; I'll illustrate this by using the e-mail service example.

I would like to note that the patterns I will use are taken from the pattern repository of Martijn van Welie[15], and these can be found in the **Appendix**. I already explained, in the chapter on interaction patterns, why this is my repository of choice. The reader should also note that larger collections exist like the one in (K. Van Duyne et al., 2002) which probably could helps us more in applying patterns, but as they are not freely available, I will concentrate on this freely available repository.

The web site patterns of this repository are grouped in four categories:

- Navigation: those patterns are concerned with easing the navigation in a website.
- Searching: These patterns are mainly concerned with locating information on large sites by using search function, site maps, etc.
- Page elements: These patterns help a designer in choosing page elements which can support the completion of tasks.
- E-commerce: Those patterns are mainly concerned on recurring problems that occur on e-commerce web sites like identifications, shopping carts, etc.

Considering these categories we can already have some ideas on how to improve our e-mail service described earlier. Among the several tasks that could be supported we found: sending an e-mail, managing an address book. I also mentioned it would be good to consider the relationships between those two tasks and why they would influence the structure of the site (see **5.2.1 The Tasks Criteria**).

We needed a log-in screen, and once we were logged in we should have to make a choice between sending an e-mail, managing the address book or consulting our inbox. So, we got the choice between three distinct tasks. Looking at the repository I find one pattern that seems to match this idea. The problem description of the pattern "Tabbing" (see **Appendix**) is as follows:

> **Problem** Users need to select an element out of a set.

The user goes on the site and obviously wants to accomplish one of the three tasks or even wants to accomplish different tasks. I also read the rationale behind the pattern:

[15] This repository can be found at http://www.welie.com/patterns/index.html

> **Rationale** Tabbing instantly shows the users which sections are available and makes them reachable in one click. An important limitation comes from the fact that the categories are usually placed horizontally. Consequently, the number of categories that "fit" is usually smaller than 10, unless an alphabetical index is used.

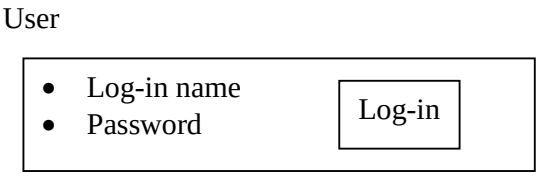
The fact that the different tasks are available in one-click can be useful if I want to accomplish several tasks. The rationale also tells me, the designer, that the pattern probably shouldn't be applied if I had to support more than 10 distinct tasks. If it were the case I should probably apply another pattern (an alternative pattern is provided in the Related patterns section).

This looks like something we could use. Looking at the pattern we can even see a wireframe to get a quick idea of what it should look like. Now we can introduce this into our design. We already got a basis which we can rely on. **Figure 11 The address list and the pop-up window of hotmail** shows this tabbing system to switch between the different tasks. The reader should also note that the "task" of choosing between the three different tasks wasn't considered to be a task on its own. We discovered this structure because we considered the relationships between those three tasks as discussed in **5.2.1 The Tasks Criteria** and by applying one of the patterns on it. Using the bridging the gap technique only we would probably have considered those three tasks distinctly.

Going further on our e-mail example, we also need a log-in screen. A log-in screen enables users to identify themselves. Using the bridging the gap technique, we can map the task to an initial design as follows:

- The task description is as follows: User wants to log in.
- The users' conceptual model would look like this[16] (see **3.2.1 Conceptual User Modeling**):
  - The noun: the user
  - The verb: logging in (this is the task the user wants to accomplish).
  - The attributes which are necessary to complete the task: log-in name and the password.
  - The relationships: there isn't a relationship with other objects. An example of a relationship can be found with the Inbox object. This object will contain e-mail objects which on their turn will contain e-mail addresses as objects, etc.
- The mapping into an initial design:

[16] Note that the users task is based on computer based application and not a real world task.



This is how the mapping of the task would look like. It gives an initial idea of how the structure of the design should look[17]. Using the patterns we could improve this by applying the "Identify pattern" (see **Appendix**). The problem in the pattern is stated as follows:

> **Problem** The users need to identify themselves.

This gives a clear idea why we could use this pattern. The rationale behind this pattern sounds like this:

> **Rationale** Users do not like to be bothered with login procedures. Only if they have reason to return frequently and the benefits of registering are clear will they actually register. Even so, they should only be asked to do so when absolutely necessary. This also facilitates browsing and exploring the site without commitments. Using a combination of email address and password makes it possible to mail the users their password when they loose it and relieves them from remembering again another login name.

This pattern also takes into account services where the log in procedure is not absolutely necessary. In our case it is necessary, but some sites like "De Standaard"[18] provide controlled access to some parts of the site (e.g. the archives of "De Standaard"). One interesting part about this rationale is that it reminds us (the designers) that users are not perfect; therefore it would be interesting to add a feature which allows users to recover their password. Here, obviously, we couldn't use the persons' e-mail address for doing so, but another feature could be used (e.g. use a personal question and if answered right, provide the password). We would probably have uncovered this feature in a later iteration, but if we uncover it later, it might be expensive.

But this is not the only thing the pattern tells us. The pattern also provides an example of a site using a log-in; designers may be inspired by it and improve the design accordingly (see **Figure 13 Example of pattern use**).

[17] I didn't visually treat the information entities. How we could treat this is explained in the "bridging the gap" chapter.
[18] http://www.standaard.be

Figure 13 Example of pattern use

One last thing I would like to add about this pattern is what appears in the "Related patterns" section:

> **Related Patterns** Consider the Registration pattern to register new users.

The patterns' interconnectedness provides links to other patterns; in this case, a link to a pattern which adds support for registering new clients. This will certainly be a pattern we could use.

Now, we'll focus on the last step of the model we discussed earlier (see **5.1 The Model**). This step is used to compare the current design to the data gathered during the analysis phase, in order to start a new iteration if needed.

5.4 From the Interaction Patterns to Task Analysis

In this step we already got a design (maybe not complete) which we can use to verify if it fulfills the goals we wanted to achieve during this iteration.

We go through this step for two main reasons:

1. We check if it meets the expectation of the users.
2. As a result of this check, we can decide what the goals of the next iteration (if one needed) will be.

First, I'll explain the checking procedure and then I'll explain how to set the goals of the next iteration.

5.4.1 Comparing the design with the users' expectations

The main reason for comparing your design with the information gathered during the analysis phase is to check whether the users' tasks are designed as they should be.

The first step that has to be taken is to compare it with the analyzed data and with the user (if this is possible). This obvious step will guarantee that we didn't forget anything during the design phase and that the goals of the iteration were achieved. This can be highly beneficial when designing in teams. A lack of communication among the team members can have repercussions on the design: a team member forgot to design a task because he thought another designer would design it, maybe a task has multiple designs, etc. This step is just a matter of knowing if the designers are heading in the right directions. Note that feedback from the user at each iteration can be very valuable and should be considered. As this is not always possible, it is up to the designers and co-workers to conduct a very good task analysis before starting a project. Secondly, during the design process, a designer might have had difficulties to design a specific task because of a lack of information about the task. If this is the case, the designer maybe should consider conducting a new task analysis with the users in order to clear things out. Finally, the designer might have uncovered details during the design process that weren't specified in the task analysis but that maybe should be considered. Again consulting the user to clear things out might be useful. This will prevent a designer from adding "nice features" as I discussed in **5.2.1 The Tasks Criteria**.

So, the goals we wanted to achieve during the iteration have been checked. Maybe, we have discovered some flaws in the design or found additional tasks to design. The next step is to determine the goals of the following iteration (if one needed of course).

5.4.2 Determining the goals of the next iteration

Having checked the design, we can now focus on the next iteration if one is needed. Using the data from the analysis phase and the current design we can set our goals for the next iterations. In order to set the goals we use the timeline introduced in **5.2.2 The Timeline**.

In the design several tasks are already designed, these tasks won't be considered in setting the goals for the next iteration unless the previous check revealed that some of the tasks were badly or not designed. Having a set of tasks which we can design, we can now, using the Timeline, set the goals for our next iteration.

Once the goals are set, we can start the new iteration.

5.5 Conclusion

In this chapter I presented an iterative process which unifies the three methodologies I discussed during this dissertation: Task Analysis and modeling, bridging the gap, and interaction patterns. Unifying these three methodologies we obtain a process which has as its main focus users and their tasks from the early phases of analysis to the

ultimate design of a web site. There is however one additional step that has to be taken, namely the implementation phase. This step is not a part of my dissertation because it highly depends on the existing technologies (e.g. html, xhtml, css, etc.) used.

Besides unifying the three methodologies, the process also introduces an iterative design process. The iterative process has been introduced because no design team is perfect and because it helps uncovering details we wouldn't have considered if they had been applied in one single step.

Finally, I also introduced a timeline to help designers focus on the right tasks during the various cycles of the iterations. This timeline should be used to focus on the various tasks at the right time. It helps designer heading in the right direction when doing design and by those means tries to shorten the iteration cycles.

6 Conclusion

The World Wide Web (W3) moved from its initial purpose, sharing knowledge using hypertext, to an environment where people can nowadays perform complex tasks that were originally only available on desktop application (e-mail) or only in the real world (shopping). Therefore, it has become more and more crucial to help people in performing their tasks on the W3. A lot of research has been done on how to design user interfaces to help people in performing tasks. Unfortunately, it has been mainly focused on desktop environments or they only addressed parts of the whole design process.

In this dissertation we saw an iterative design process for designing web sites. This process includes three steps which encompasses the process from the early stage of knowing your users, Task Analysis and Modeling, then mapping this knowledge in a initial web site design, Bridging the Gap, then improving the design by using proven design knowledge, Interaction Design Patterns.

The iterative process helps designers taking the three steps from the early stages to the final design. The iterative process refines the design gradually, and ensures that the designers are designing well and are heading in the right direction. In order to guide the designers they have to follow a set of guidelines which helps them focusing on the right tasks at each iteration; and therefore minimizing the number of iteration cycles (which from a money and time point of view is crucial).

Human Computer Interaction is a fairly new discipline and it is also true for the Interaction Design Patterns. Interaction Design Patterns are at the early stages of becoming a fully-fledged language. Several web site pattern repositories exist, but most of them are not freely available. Besides not being freely available the fact that there are several repositories may confuse the designers. However the proven design knowledge they contain must encourage designers of using them, and it is only by using them that they will mature and maybe will become the Object Oriented Software Design Patterns of Human Computer Interaction.

In this dissertation I explained a design process. However it would greatly help designers if tools existed integrating this process. Using the structured data from the task analysis and modeling phase, a tool could generate an initial design using the mapping technique described in Bridging the Gap. These tools could also integrate the repository of Martijn van Welie, and use the pattern's wireframes as basic design templates.

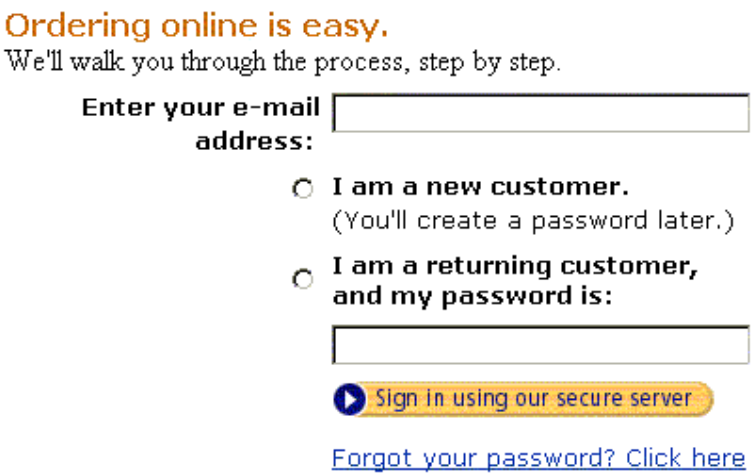
7 Appendix

7.1 Identify Pattern

Identify

Author	Martijn van Welie
Problem	The users need to identify themselves
Principle	???
Context	Sites where users frequently return to perform actions that involves a large amount of "constant" information, or sites where privacy/security is important. (so not for shops where people only buy an item once and never return). Also applicable when the history of the users is important to them.
Forces	• Users do not want to remember passwords and will easily forget them • Users only want to identify themselves if it is really necessary • Users will not register if they don't know what it will bring them
Solution	**Ask the users the identify themselves but only when necessary.** Allow users free access of the site until it is absolutely necessary that they identify themselves. Use a combination of an email address and a password. Optionally the email can be filled in automatically the next time the user returns. By using the email address as "login name" users can retrieve their password if they loose it (which they will...). Offer help to users who forgot their password. Once users are logged in provide feedback that confirms this. Do not suggest that the users are logged in when they are not; for example by welcoming them by name once they enter the site.
Rationale	Users do not like to be bothered with login procedures. Only if they have reason to return frequently and the benefits of registering are clear will they actually register. Even so, they should only be asked to do so when absolutely necessary. This also facilitates browsing and exploring the site without commitments. Using a combination of email address and password makes it possible to mail the users their password when they loose it and relieves them from remembering again another login name.

Examples



At Amazon's, the users log in with their email address and password. They are offered help in case they forgot their password.

Known Uses www.amazon.com; www.bn.com

Related Patterns Consider the Registration pattern to register new users.

7.2 Tabbing Pattern

Tabbing

Author Martijn van Welie

Problem Users need to select an element out of a set.

Principle

Context Typically there is a large amount of items from which the users need to select a specific item. The items is either known already or the users have an idea about the kind of item they are after. The items should be easy to categorize into a small number of categories, typically less than 10.

Forces
- Large amounts of data require a lot of space but the available display space is limited
- Large amounts of data are usually not unrelated and can be divided into categories that match the user's conceptual model of the data.

Solution **Divide the data in sections and present one section at a time. Show all sections and let users switch directly between sections.**

Show the section labes horizontally and place the items for the current section underneath it. The currently selected category is highlight. The sections are typically categories but could also be "points" on a discrete scale, for example the alphabet to form an alphabetical index. When categories are used it is like

having tabular sheets. This effect can be enforced by showing the tabular sheets visually, i.e. connecting the items-area with the current selection.

Rationale	Tabbing instantly shows the users which sections are available and makes them reachable in one click. An important limitation comes from the fact that the categories are usually placed horizontally. Consequently, the number of categories that "fit" is usually smaller than 10, unless an alphabetical index is used.
Examples	Planet Movie Index, CN movies.
Known Uses	
Related Patterns	Consider the Paging pattern when users need to go through an ordered list of items in a linear manner.

8 References

(B. Appleton, 2000)
Patterns and Software: Essential Concepts and Terminology
by Brad Appleton
http://www.enteract.com/~bradapp/docs/patterns-intro.html , 2000

(P. Azevedo et al., 2000)
OVID to AUIML – User-Oriented Interface Modeling
by Pedro Azevedo, Roland Merrick, Dave Roberts
IBM United Kingdom, 2000

(V. Bush, 1945)
As We May Think
by Vannevar Bush
The Atlantic Online, July 1945
http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm

(T. Berners-Lee, 1989)
The World-Wide Web Initiative
by Tim Berners-Lee, 1989

(T. Berners-Lee, 1993)
The World-Wide Web Initiative
by Tim Berners-Lee, R. Cailliau, N. Pellow, A. Secret, 1993

(R. Bias et al.,1994)
Cost-Justifying Usability
by R. Bias, D. Mayhew
Academic Press, 1994

(J. Borchers, 2002)
Teaching HCI Design Patterns: Experience From Two University Courses
by Jan Borchers
Stanford University, 2002

(A. Dix et al., 1998)
Human-Computer Interaction
by Alan Dix, Janet Finlay, Gregory Abowd, Russell Beale
Prentice Hall, second edition, 1998

(C. Fellenz et al., 1998)
Web Navigation: Conflicts between the Desktop and the Web
by Carola Fellenz, Jarmo Parkkinen and Hal Shubin
A CHI 98 Workshop, 1998

(R. N. Griffiths et al. 1997)
Don't Write Guidelines Write Patterns !
by Richard N. Griffiths, Lyn Pemberton
University of Brighton, UK, 1997

(J. T. Hackos et al., 1998)
User and Task Analysis for Interface Design
By JoAnn T. Hackos, Janice C. Redish
John Wiley & Sons, 1998

(H. Hartson, 1998)
Human-computer Interaction: Interdisciplinary roots and trends
by H. Hartson
Journal of Systems and Software, 1998

(S. Henninger, 2002)
Tools Supporting the Delivery and Application of Usability Patterns
by Scott Henninger
Department of Computer Science & Engineering
University of Nebraska-Lincoln, 2002

(C. Karat, 1994)
A business case approach to usability cost justification
by C. Karat
Cost-Justifying Usability, Academic Press, 1994

(B. Kirwan et al., 1992)
A Guide to Task Analysis
by B. Kirwan, L. Ainsworth
taylor and Francis, 1992

(J. Kramer et al., 2001)
Designing Object Oriented E-Commerce Web Sites
by Joseph Kramer, Sunil Noronha, Didier Bardon, 2001

(K. Mullet, 2002)
Structuring Pattern Languages to Facilitate Design
by Kevin Mullet
Reactor Experience Design, 2002

(J. Nielsen, 1996)
Why Frames Suck (Most of the Time)
By Jakob Nielsen
Alertbox, December 1996
http://www.useit.com/alertbox/9612.html

(J. Nielsen, 1998)
Does internet = Web?

by Jakob Nielsen
Alertbox, September 1998
http://www.useit.com/alertbox/980920.html

(P. Pop, 2000)
Comparing Web Applications with Desktop Applications: An Empirical Study
by Paul Pop
Dept. of Computer and Information Science, Linköping University, Sweden, 2000

(D. Rijken, 1994),
The timeless way… the design of meaning
by D. Rijken
SIGCHI Bulletin, 1994

(D. Roberts et al., 1998)
Designing for the user with OVID: Bridging user interface design and software engineering
by Dave Roberts, Scott Isensee, John Mullaly
Macmillan Technical Publishing, 1998

(A. Seffah et al., 2002)
On the Usability of Usability Patterns
by A. Seffah, H. Javahery
Department of Computer Science, Concordia University, Canada, 2002

(H. Shubin, 1997)
Navigation in Web Applications
by Hal Shubin
Interaction Magazine, November 1997, issue IV.6

(J. Tidwell, 1999)
Common Ground: A Pattern Language for Human-Computer Interface Design
by Jenifer Tidwell
http://www.mit.edu/~jtidwell/common_ground.html , 1999

(D. Van Damme, 2002)
Onderzoek naar het ontwikkelen van grafische gebruikersinterfaces met behulp van moderne webtechnologieën
by David Van Damme
Vrije Universiteit Brussel, Departement Informatica, 2002

(K. Van Duyne et al., 2002)
The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience
by Douglas K. Van Duyne, James A. Landay, Jason I. Hong

Addison-Wesley, 2002

(M. van Welie)
Task-based User Interface Design
by Martijn van Welie
Vrije Universiteit Amsterdam, 2001

(J. Vlissides, 1998)
Pattern Hatching: design patterns applied.
by John Vlissides
Addison-Wesley, 1998

(L. Wedeles, 1965)
Professor Nelson Talk Analyzes P.R.I.D.E.
by Laurie Wedeles
Vassar College *Miscellany News*, February 1965

(L. Wood, 1998)
User Interface Design: Bridging the Gap from User Requirements to Design
by Larry Wood
Springer Verlag, 1998

(S. Wilson et al., 1995)
Empowering users in a Task-based approach to design
by Stephanie Wilson, Peter Johnson
HCI Laboratory, Department of Computer Science Queen Mary and Westfield College, 1995