UNIVERSITÀ DEGLI STUDI DI PISA

DIPARTIMENTO DI INFORMATICA

DOTTORATO DI RICERCA IN INFORMATICA

PH.D. THESIS

Criteria to Improve Web Site Usability and Accessibility When Interacting Through Screen Readers: Definition, Application, and Evaluation

Barbara Leporini

Supervisor: *Dr. Fabio Paternò*

September 2003

“Navigare necesse est, vivere non est necesse!"

(Pompeo Magno, 106 – 48 a.C)

A mia madre, con affetto e gratitudine.

Acknowledgements

Thanks are due to my supervisor Fabio Paternò for scientific support and guidance during my PhD research, to Piero Maestrini for encouragement and thoughtful advice, and to my friends and colleagues for daily moral support and for being there every time I needed help: Carmen Santoro, Carmine Ciavarella, Francesco Correani, Giulio Mori, Grazia Piras, Laila Paganelli, Micol Pizzolati, and Silvia Berti.

Thanks are also due to the "SOGEI" for funding me during this year, to Francesco Conversano for his important contribute to the development of the tool, and to all people who participated to the user testing.

Thanks to all my friends, who believed in me and sustained me in difficult moments.

A special thank goes to Ivan Norscia for his immense friendship, encouragement and for practical and moral support. I would like to thank him also for his important and precious contribute to the realization of this thesis: *"Ivan, grazie davvero"*.

I am grateful to my family, in particular to "Tato", for never ending support.

Finally, thanks are not enough to express my infinite gratitude to my mother, who has always been following and assisting me with continuous devotion, and without whom all of this wouldn't have been possible: *"Grazie mamma! Ti voglio bene"*.

Abstract

This research is related to the usability and accessibility of Web sites.

Guidelines for Web site usability already exist, but they only marginally consider the exigencies of “special users”, such as blind people or subjects with high levels of vision deficit.

This study specifically aimed at defining, in a more precise way, the usability of Web sites, in order to improve their accessibility for “special users”, who are obliged to navigate on the internet through screen readers.

First of all, 19 criteria (general principles) and 54 checkpoints defining each criterion (technical solutions) were proposed; then, possible ways of application of such criteria and checkpoints was specified. This represented the starting point to evaluate the usability of Web sites: in this work, the heuristic-based method was proposed and used in order to assign levels of usability to several Web sites of interest.

A user testing was performed by 15 voluntary users, chosen among blind and low vision subjects. Two Web site prototypes were specifically designed for this purpose, only differing for the presence/absence of important usability criteria defined in this study. By comparing the time spent by users navigating and performing assigned tasks on the two Web site (with and without criteria), the impact of the application of the proposed criteria on the quality of the navigation was estimated.

Finally, an automatic tool, whose implementation is in progress, is briefly presented at the end of this work. This tool is the first step toward a complete and definitive automatic procedure able to evaluate real Web site usability, especially considering blind and low vision people’s constraints. Further studies are in progress to reach this final goal.

Index of Content

1
Introduction

1.1 The Problem

In recent years the use of Web sites has been widening, and the number of users accessing them is steadily increasing. For this reason, it is important that the information be easily reachable by all, including people with disabilities. The difficulties in providing such universal access can be addressed through the application of the principles of usability and accessibility.

This work deals with usability of accessible Web sites. Usability and accessibility are two concepts that can be referred not only to Web sites, but also to general user interface. Usually, we can make a distinction between WIMP (Windows, Icons, Pointer, and Mouse) interfaces and Web interfaces, in part because the nature of these interfaces differ and in part because the usability methods developed have often only been applied to one type or the other in the literature. Generally speaking, we use the term WIMP to refer to applications, such as programs written by Java, C++, Delphi, etc.

Web applications feature significant differences with respect to classical WIMP user Interfaces, such as:

- the structure of the Web site (e.g., number of pages, links between pages) is directly related to the information that is available on the Web site (amount, type, etc), while the structure of WIMP user interfaces is usually static;
- Web applications are usually modified often and without any notice to the user, so they can be considered more unpredictable;
- moving from one Web site to another corresponds to switching

from one application to another, but in Web sites no guideline is provided for ensuring inter-application coherence [Scapin et al. 2000B].

More in specific, WIMP applications are characterized by Windows, menu bars, tool bars, dialogue windows, buttons, icons etc, while Web interfaces are formed substantially by text, links, and images. Furthermore, WIMP interfaces tend to be more functionally-oriented than Web interfaces: in WIMP interfaces, users complete tasks (such as opening or saving a file, by following specific sequences of operations), while most Web interfaces offer limited functionality (such as selecting links or completing forms), since the primary role of many Web sites is to provide information. In addition, the presence in Web interface of links leading to remote pages represents one important difference between WIMP and Web. We have mentioned the term task, but what is a task? A task is an activity performed to reach a goal. We can think of tasks at different abstraction levels, ranging from high-level tasks (such as retrieving information on film projections available today) to very low-level tasks (such as selecting a button on the screen) [Paternò 1999].
In both cases we can think of tasks that can or cannot be performed, and of temporal ordering among tasks. In case of high-level tasks the temporal relationships are determined by the logical dependencies among tasks (for example printing a file can be performed only after indicating the name of the file to print) whereas in the case of the low level tasks temporal relationships may depend also on constraints provided by the implementation of the user interface.
Task models describe how activities can be performed to reach the users' goals when interacting with the application considered (see section 2.4.3 for more details). These concepts are important also for World Wide Web applications, because their use is increasing by many types of people. So, more and more features of WIMP interfaces are used. For this reason, it is important to note that WIMP and Web interfaces are closely related, because many features of WIMP are also in Web interfaces. Thus there are different evaluation techniques for WIMP and Web interfaces, but several techniques can be used for both.

A Web site is an interactive software system. It interacts with at least two different kinds of users: end users trying to achieve some goal and developers/maintainers striving to keep the system working and improving it [Brajnik 2000].
End users can be characterized in terms of:

- Goals and tasks: e.g. information seeking, choosing where to buy some specific product, buying it, writing a book review, etc.
- User characteristics: user behaviour during information seeking processes is strongly affected by users (culture, language, previous knowledge on the field, experience in using the Web).
- Technology: end users interact with the Web site through a layer of technology that is not under control by the Web designer, such as browsers, protocols, operating system platforms, interaction devices (screens, speaking devices, reduced telephone keyboards, etc.), network connections.

Information seeking through browsing is a process that almost all Web sites must support. Unfortunately, it is also a difficult task to model and support because it encompasses complex cognitive, social and cultural processes [Allen 1996] spanning through interpretation of textual, visual, audio messages, selection of relevant information and learning.

Amongst developers and maintainers activities, a prominent role is played by actions that include: corrective maintenance (i.e. fixing problems with the Web site behaviour or inserting missing contents), adaptive maintenance (i.e. upgrading the site with respect to new technologies), maintenance (i.e. improving the site behaviour or content), and preventive maintenance (i.e. fixing problems in behaviour or content before they affect users).

Noting that in the last years the use of Web sites is widening, and that the number of users who approach them is increasing more and more, it is important that information be easy reachable by all. More precisely, the user population is expanding in age (ranging from young users to elderly people), in expectations (ranging from private use for leisure to professional use), in information needs (ranging from simple information to compound multimedia resources), in task types (ranging from basic text searches to complex problem-solving methods), and in user abilities (ranging from the able-bodied person to any person with special needs, such as for motor - auditory - or visually - impaired persons). This concepts may be summarized in the sentence "anyone, anywhere". For this reason the idea of accessibility and usability is considered. So, what is accessibility and usability of a Web site? In the next paragraphs we discuss these two concepts.

1.1.1 Accessibility

In order to refer to accessibility aspects, we have to consider that many users may be operating in contexts very different from those of general use: they may not be able to see, hear, move, or may not be able to process some types of information easily or at all; they may have difficulty in reading or comprehending text; they may not have or be able to use a keyboard or mouse; they may have a text-only screen, a small screen, or a slow internet connection; they may not speak or understand fluently the language in which the document is written; they may be in a situation where their eyes, ears, or hands are busy or interfered with (e.g., driving to work, working in a loud environment, etc.); they may have an early version of a browser, an entirely different browser, a voice browser, or a different operating system.

In the context of Web site design, accessibility is a measure of how easy it is to access, read, and understand the content of a Web site. A Web site can be said to be accessible if it can be used from everyone, including people with disabilities. Accessibility is complicated by the fact that a Web site is not a published piece of work so much as a living document that can be interpreted in different ways by different browsers and on different platforms. For example, in addition to people with disabilities who explore Web pages by using screen readers (with voice synthesizer or Braille display), we can include those using low-bandwidth technology like cellular phones, black and white screens, speaking browsers via telephone, etc.

In order to obtain a Web site more accessible, the Web designer should follow some simple criteria and guidelines which do not limit the graphical features, such as images and icons, as a way to improve accessibility, rather provide a text equivalent information combined to each multimedia content.

Most accessibility issues are taken into account especially by W3C (World Wide Web Consortium) in the project Web Accessibility Initiative (WAI). In fact, Tim Berners-Lee (W3C Director and inventor of the World Wide Web) has defined accessibility as "The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect".

In the WAI activities, a set of specific guidelines and recommendations has been defined: "Web Content Accessibility Guidelines 1.0" [WAI 1999]. These guidelines

explain how to make Web content accessible to people with disabilities. The guidelines are intended for all Web content developers (page authors and site designers) and for developers of authoring tools. The primary goal of these guidelines is to promote accessibility. However, following them designers will also make Web content more available to all users, whatever user agent they are using (e.g., desktop browser, voice browser, mobile phone, automobile-based personal computer, etc.).

The 14 defined guidelines address two general themes: ensuring graceful transformation, and making content understandable and navigable (do not rely on colour alone, use mark-up and style sheets and do so properly, create tables that transform gracefully, provide context and orientation information, provide clear navigation mechanisms, and so on).

At the same time as, "Techniques for Web Content Accessibility Guidelines 1.0" explains how to implement the checkpoints (i.e., how someone may apply and verify if the guideline is satisfied) defined in the WAI document.

Moreover, each checkpoint has a priority level assigned by the Working Group based on the checkpoint's impact on accessibility: a guideline "must", "should" or "may" be complied.

In additions to guidelines for Web content, WAI also elaborates User Agent Accessibility Guidelines and Authoring Tool Accessibility Guidelines [W3C].

Noting that accessibility ensures graceful transformation (it should remain accessible despite physical, sensory and cognitive disabilities, work constraints and technological barriers), and it makes content understandable and navigable (it should present its content in a clear and simple language, and should provide understandable mechanisms to navigate within and between pages).

Content developers must consider these different situations during page design. While there are several situations to consider, each accessible design choice generally benefits several disability groups at once and the Web community as a whole.

1.1.2 Usability

Usability was defined by Lindgaard [1989] as “...a quantitative, or quantifiable, statement of the ease with which users can accomplish tasks for which a given computer system was designed”.

Usability of a Web site is determined by user satisfaction, ease of learning and remembering its organization and functionalities, user effectiveness, efficiency and likelihood of errors while performing the tasks the site has been designed for, like finding information needed, or completing e-commerce operation [see also Turk 2001].

In this work, however, we refer to the standard ISO9242, which defines usability as "the effectiveness, efficiency and satisfaction with which specified users achieve specified goals in particular environments", where:

- Effectiveness means "the accuracy and completeness with which specified users can achieve specified goals in particular environments;
- Efficiency means "the resources expended in relation to the accuracy and completeness of goals achieved";
- Satisfaction means "the comfort and acceptability of the work system to its users and other people affected by its use".

The most important properties for Web site usability related to navigability (most of them taken from [Fleming 1998]), which are often taken into consideration, are:

- Consistency of presentation and controls across the site, natural organization of information (clear structure, systematic labels, clear and meaningful labels);
- Contextual navigation, in terms of environment, "type" of users, particular devices, etc; someone often considers also how much information is given for providing a context to the user (where is he, where he can go, and so on);
- Robustness, i.e. how well the Web site handles technology used by users that has not been foreseen by developers;
- Flexibility (e.g. availability of graphic and textual versions, redundant indexes and site maps, duplicated image map links);
- Functionality (i.e. support of user goals) etc.

These properties need to be decomposed into more detailed ones that can be assessed in a simpler and perhaps more standard way in order to be operationalized. They can be considered in the Web site evaluation, in order to determine whether it is usable or not.

Usability is not the only important aspect of the site: obviously its content/functionality, and its popularity contribute even more to the success of a site. However, if users can do the same thing with two different sites, they will choose the one that is more effective, efficient and satisfactory.

1.2 Our Approach to Usability and Accessibility

After having seen the corresponding definitions, we can note that accessibility and usability are closely related, as they both improve satisfaction, effectiveness, and efficiency of the generic user population. In our approach we consider the two concepts to be correlated, but distinct. Accessibility is aimed specifically at making a Web site more available to a wider population of users, (including special categories of people, such as blind users) by removing the technical barriers that prevent the access to the information included in the site. Accessing the information, however, is not enough. In fact, once the site is accessible for a certain typology of users (such as blind people), it can be more or less usable by the same typology: the goal of usability is to make users' experience with the Web site more efficient and satisfying. Of course, in this perspective, a Web site cannot be used if it is not accessible, therefore technical accessibility is a pre-condition for usability.

Usually usability and accessibility issues are treated separately. Our aim herein is to identify their relationships: in particular we want to address the meaning of usability in the context of Web sites accessed by disabled users through screen readers.

In this context, accessibility issues refer to the mere access to the information included in Web pages, while usability issues regard the facility of exploring such available information. For instance, if graphical links do not have `ALT` content, it is an accessibility question. If graphical links have "poor" `ALT` contents (e.g., "click here"), or all the contents are equal among them (e.g., "go to paragraph"), it is a usability issue.

Often, when designers consider special users, they tend to address only accessibility, ignoring usability for such users.

Even if a site is theoretically accessible because it completely complies with technical accessibility standards, it can still be very hard to use for people with disabilities, at the point that they may not succeed in reaching their goals.

Appropriate Web site design and evaluation methods help to ensure that Web sites are usable [Nielsen and Mack 1994], but they are so numerous and hard to differentiate that identifying which ones are suitable for a particular Web site is challenging. This depends on several needs: for a cost-effective method, for remote evaluation, for real-time evaluation, etc.

Hence, an adequate design of Web site, by following rules and guidelines, aids achieving usability and accessibility, but evaluation testing is required too.

In literature, guidelines and evaluation methods for accessibility and usability of the Web sites are given and discussed separately. We would like to combine both ideas, i.e. considering the usability of Web site accessibility. Often, we think of a Web site is accessible if the designer complied with WAI guidelines, but it is not always enough. In fact, if a Web site observes recommendations by W3C, it does not automatically imply that users are able to navigate well, or that they will succeed to reach their goals. Also the Web site structure has some effects on Web usability by all, but overall by special users such as people with visual or auditive impairment, etc. To evaluate usability referred to these cases, we have to consider the context in which these users work, i.e., browsers, particular devices (voice synthesizer or Braille display), programs (e.g., screen readers, magnifying programs) and so on. When we use a screen reader, how the information is set out on the page is very important. For instance, if the page content that changes dynamically is shown in a confused way, those special users can have several problems reading them. This, as well as time saving, can also cause that a user is not able to achieve his goal. Or, the same problems can be caused by layout of links, frames, form fields and so forth. For example, static links are the main cause of not good navigability; some data arrangements can provide difficult patterns in which users have many troubles (and in some cases also impossibility) in order to achieve their ends.

Moreover, when we evaluate Web usability, we tend to consider user actions by mouse, and we do not pay attention to actions performed through the keyboard. Considering usability issues for disabled people, we have also to capture interactions by keyboard, without neglecting commands of the screen readers or special devices. Users can have different problems depending on the screen readers or devices used (i.e., user context), because several commands exist and different modalities (e.g., voice or touch). Thus, the usability evaluation may have different results.

1.3 The Context

Usability is a multidimensional concept, since it can refer to several aspects whose importance depends on the application domain. So in order to define the usability of the Web site accessibility, according to the ISO 9241 usability definition, we have to indicate which are the *specified users*, the *particular context*, and the *specified goals*.
In our work we consider, in specific, accessibility and usability issues for blind or visual impaired people. Therefore, we focus on the context in which these users work, i.e., browsers, particular devices (voice synthesizer or braille display), particular programs (screen readers, screen magnifiers) and so on.
One of the most important computer interfaces for blind people is a screen reader. A screen reader is software program that provides access to computer software applications and the Internet by using a speech synthesizer to read the information on the monitor out loud. At the present time, several screen readers, with different characteristics, are available on the market [for review see: Leventhal and Earl 2000]. All of them follow the focus of the system and support text reading (on the whole, line by line, word by word, etc.). They also support a Braille display. Refreshable Braille Displays are electronic devices that are used to read text that a computer sends to the monitor. The device is connected to the computer and produces Braille output on the Braille display. Braille displays only read one line of text at a time. These displays generally include directional keys which allow the user to navigate through a document. However, in general the content reading is in sequential way also through voice synthesizer. In fact, blind users usually move through keyboard, especially with Tab key, Arrow keys, and shortcuts.

Older versions of screen readers were designed to simply read text on the screen, since they had been developed for text-based operating systems such as DOS or Unix. These older versions sometimes have difficulties with GUI (Graphical User Interfaces) such as Windows, Mac, or graphical Web pages. Newer versions read out what is happening on the screen, such as which dialogue boxes are opening on the screen (including icons, menus, text, punctuation, and control buttons), so that users can use them with GUI.

Not all the screen readers use the same technology for collecting and analyzing data captured by the system and its applications.

In particular, the way screen readers process the Web page can differ: some of them perform this task by directly reading the visualized layout (such as NARRATOR), while others operate by directly capturing and processing the HTML code sent to the browser (such as Jaws for Windows).

This second typology of screen reader looks at the HyperText Markup Language (HTML) file that generates a Web page and read the content using synthetically generated speech. The best readers "understand" specialized HTML and, if a few simple rules are followed, the page content can become accessible to the visually disabled.

Often the underlying structure of a Web page is constructed in an illogical manner, since the major effort is canalized on the production of a visually pleasing Web page. Unfortunately, the aspect of a Web page is secondary when a screen reader is used to accede it: if the content of a Web page is not properly organized, the screen reader output may not make sense to the listener. In fact, the screen reader output depends on the underlying structure of a Web page, rather then on its visual features.

For these reasons, and others that we are going to explain below, some important points should be kept in mind when designing Web contents to make sure that they are accessible to people using screen readers. First of all, limiting use of images, and put all important information as text, instead of graphics with text in them. This is because screen readers read text and can not interpret graphics. Hence, it is important to make sure in using the alt tag function in HTML to add a description of image that the screen reader will be able to read to the user.

This is especially required for graphics that provide important information. For example, if a navigation bar has text links, a screen reader will speak those links. However, if the navigation bar uses graphic buttons and provides no alternative text, the reader will either ignore these links or repeat the word "graphic". Now the user is lost on the page and doesn't know where to go next.

Other problems result from poor or incomplete HTML that confuses the reading software, causing the screen reader to skip content and links or to read text out of order, speaking nonsensical sentences. Another issue for visually impaired Web surfers is columns of text. Some of the screen readers, such as Jaws, read one line at a time, not one column at a time, meaning that information contained in columns can often become garbled.

Low vision people use magnifiers programs. Screen magnification and enhancement allows users to enlarge text/images and to change monitor settings (e.g., brightness, contrast, etc.) to meet individual reading needs. Although visual impaired people are able to see something on the screen, various problems arise in computer using. Colours, partial vision, mouse adjustment are examples of such problems.

When interacting with a screen reader, the way in which the information is set out on the page is very important, because what the blind user hears is very different from what sighted users would read from the screen. Thus, blind and vision-impaired people were involved during the stage of identification of possible criteria, through preliminary tests on some selected examples.

Both the inspection evaluation dynamically conducted on the Web pages and the user testing performed in this research, require a good knowledge of the screen reader adopted. Thus, among the screen readers available, we decided to consider Jaws for Windows [Jaws], which processes the Web page HTML code. This specific choice was made mainly for two reasons:

- Jaws is the most used by blind people, thus it represented the most appropriate candidate for the user testing;
- Checkpoints related to criteria proposed in our work refer to the same HTML code used by Web page designers.

However, since criteria proposed are general principles, and checkpoints are referred to the HTML code, such criteria can be applied to other screen readers than the one considered in this work.

The output devices interacting with the screen reader we considered are voice synthesizer and Braille display. Then, we tested our criteria by using Internet Explorer, because it is the browser which works better with Jaws.

Finally, a good and easy navigability by using a screen reader is the goal of our studies. We would like to deal with not only accessibility but also usability when the Web site navigation is through screen reader. So, if the users are able to access to information in some way, the Web site is accessible; if the navigation is easy through a screen reader, the Web site can be considered usable for the disabled user. So, we consider the usability aspects in the graphical environment (windows), and not only the possible access problems in a textual environment (e.g., by using lynx browser). Assuring the access to information is an indispensable condition, but improving the navigability in a graphical environment is important too.

1.4 Aim of the Research

This work addresses the user interface design in Web sites to improve usability and accessibility issues for blind and visual impaired people. To assist the designers to handle Web pages, well-defined criteria and guidelines must be provided to guide them in the development process of more usable and accessible Web sites.

Several usability guidelines have been formulated [Nielsen and Mack 1994], [Vanderdonckt 1999], [Scapin et al 2000A] and [Nicolle and Abascal 2001]. Most accessibility issues are taken into account especially by W3C (World Wide Web Consortium) in the Web Accessibility Initiative [WAI 1999]. They put forward a number of recommendations and guidelines to promote Web accessibility: "Web Content Accessibility Guidelines 1.0" [WAI 1999]. The guidelines are intended for all Web content developers (page authors and site designers) and for developers of authoring tools. Such guidelines mainly focus on Web accessibility aspects for people with disabilities, though in our opinion, they do not consider usability enough in this context. Hence, we intend to investigate those aspects that affect the

organization and rendering of information in the page. In practice, we would like to extend accessibility guidelines, while also aiming at improving and facilitating task performance.

Our contribution mainly focuses on advancing new design criteria aimed at improving usability and accessibility when the interaction is through a screen reader, paying particular attention to usability issues. On the basis of potential usability and accessibility problems, we formulate possible solutions, which are then tested and systematically classified according to usability principles.

The aim of our proposed criteria is providing new tools that can be used by developers in the working out activities, and by evaluators during test phases, in order to make Web sites more usable and accessible.

Thus, first we would like to provide general usability principles for Web site access through screen readers, which can be useful to developers to focalise specific aspects and issues they can refer to during the handling of the Web site. Furthermore, we would like to give various criteria application examples in order to better clarify their effect on user interface. Then, the aim of our work is to provide technical indications that help both developers and evaluators in the usage of the proposed criteria. So, the criteria are described in more details. All this is aimed at guiding designers in the choice of appropriate solutions, and in their application and evaluation. In fact, often accessibility and usability principles are clear to developers, but how to effectively implement them is a more difficult problem. Furthermore, identifying specific and significant situations to be taken into account is not easy. Therefore, our criteria should focus on those aspects in order to facilitate the developers' task. To this end, some application examples can clarify the usability issues in this domain. By showing how the criteria can be applied, what their effects are on the user interface, and how the screen reader interprets the criteria, we would like to assist developers in their work.

This work would also like to consider the evaluation issues in order to define an appropriate method to assess the quality of a Web site according to our proposed criteria and checkpoints. The method we propose defines several quality levels, which should help evaluators to determine the measure of usability for Web site

interaction with screen readers. Furthermore, the method should allow evaluators to compare different Web site designs.

1.5 Organization of the Thesis

This thesis is structured as follows: in the next chapter we give some basic information about usability evaluation, a taxonomy of usability evaluation methods, and we wish to describe in more details how in general the most used method types are conducted (testing, inspection and inquiry). Furthermore, we discuss the state of the art of existing tools to evaluate usability and accessibility in Web sites.

In chapter 3, the criteria are defined and classified. Besides, potential usability problems and the individuation phases of criteria are mentioned. Then, in the next chapter through various application examples to existing Web sites, we would like to show which is the criteria effect on the user interface. To this end, main situations that developers should take into account in the design phase are considered. In chapter 5, technical solutions (checkpoints) are proposed in order to suggest how the criteria can be applied and evaluated. Then, a short description of the criteria application is discussed. Lastly, evaluation of the proposed criteria is considered. Chapter 7 focuses on a user testing conducted in order to collect some empirical data about the impact of criteria usage on end users. In chapter 8 an evaluation model for inspection-based evaluation is proposed; in addition, a rating of 16 Web sites according that proposed inspection evaluation model is reported. In chapter 9 we draw some conclusions on this work and provide some indications for future work.

2
Evaluation Method Background

2.1 Introduction

In this chapter we summarise some basic concepts on usability evaluation and testing methods. In the next section we provide a definition of usability evaluation and a taxonomy of usability evaluation methods. Then, in section 3 we describe in more detail how in general the most used method types are conducted (testing, inspection and inquiry). In section 4, we focus on those methods which we refer to for our proposed evaluation, describing more precisely the steps to follow. Finally, we discuss the state of the art of existing tools to evaluate usability and accessibility in Web sites.

2.2 Usability Evaluation Methods

In order to achieve a usable (and accessible) Web site, an appropriate design is needed. However, in order to be sure to achieve usability, empirical testing is required. There exist several evaluation techniques by which a Web site can be evaluated. Unfortunately, often this phase is undervalued by designers, also because it requires a big effort. On the contrary, it should be very important to apply this phase during the Web site life cycle, so that likely design errors can be taken out early (the same as for any software development).

As we said in the introduction, we can distinguish between WIMP and Web interfaces, but including the evaluation testing is important for both. Of the several methods which have been proposed, not all can be applied to both WIMP and Web interfaces. Below, after giving some general definitions, we focus on Web evaluation techniques.

2.2.1 Usability Evaluation: Basic Concepts

Usability engineering [Nielsen 1993] concerns the development of systematic methods to support usability evaluation.

"Usability evaluation" can be defined as the act of measuring (or identifying potential issues affecting) usability attributes of a system or device with respect to particular users, performing particular tasks, in particular contexts. The reason that users, tasks, and contexts are part of the definition is that the values of usability attributes can vary depending on the background knowledge and experience of users, the tasks for which the system is used, and the context in which it is used [Hilbert and Redmiles 2000].

Definition of UE

Usability evaluation (UE) consists of methodologies for measuring the usability aspects of a system's user interface (UI) and identifying specific problems .

"Usability data" is any information that is useful in measuring (or identifying potential issues affecting) the usability attributes of a system under evaluation [Hilbert and Redmiles 2000].

Usability evaluation is an important part of the usability interface design process, which consists of iterative cycles of designing, prototyping and evaluating [Dix et al. 1998; Nielsen 1993].

When evaluating a user interface, it is important to know what usability means for the current application. This is why an analysis of users and their needs is important: if we do not know what the user wants and needs, we cannot know which tasks s/he must be able to perform [Lecerof and Paternò 1998]. Evaluation can be performed at different times in the development process. During the early stages evaluations tend to be done to predict the usability of the product or to check the design team's understanding of the users' requirements. Later on in the design process the focus is more on identifying usability problems and improving the user interface.

Empirical and analytical approaches

Various types of approaches have been proposed for usability evaluation purpose.

They can be substantially grouped in two approaches: empirical and analytical approaches.

In empirical testing the behaviour of real users is considered. Thus, end users (extraneous to development) are involved in the evaluation process.

It can be very expensive and it can have some limitations too. It requires long observations, which can take a lot of time to designers, and some relevant aspects can still be missed.

In analytical approaches, the evaluation is conducted by designers or expert evaluators. In these approaches, evaluators can use Model-based or inspection-based techniques.

Model-based approaches to usability evaluation use some models, usually task or user models, to support this evaluation. In inspection-based techniques for usability evaluation designers analyse a user interface or its description.

However, we will discuss better about these methods in next paragraphs.

Evaluation phases

There are many techniques applied to evaluate user interfaces; some techniques can be applied during the development phase (e.g. heuristic analysis), while others have to be used only when the design or prototype is finished (e.g. formal analysis).

Evaluation testing involves several activities which depend on the method used. The activities more often used are:

- Capture, it consists of collecting usability data, such as task completion time, errors, guide- line violations, and subjective ratings;
- Analysis, is the phase in which usability data are interpreted to identify usability problems in the interface;
- Critique, consists of suggesting solutions or improvements to mitigate the problems identified.

Many methods have been proposed because different techniques reveal different usability problems: usability evaluation typically only covers a subset of the possible actions users might take. Furthermore, different testers may detect different problems even if the same evaluation method is used. For these reasons, usability experts often recommend using several different evaluation techniques [Dix et al. 1998; Nielsen

1993]. Thus, further efforts (e.g. increasing the evaluator team) are required. One solution to this issue is trying to automatize all, or in part, the process of testing, such as the capture, analysis, or critique activities. This aspect will be discussed in the next paragraph.

Black-box and White-box testing

Referring to software testing in general, the classical distinction of test techniques is between *black-box* and *white-box* (pictorial terms derived from the world of integrated circuit testing). Test techniques are here classified according to whether the tests rely on information about how the software has been designed and coded (*white-box testing*), or instead only rely on the input/output behaviour, without no assumption about what happens in between the "pins" (precisely, the entry/exit points) of the system *black-box*.

Therefore, the terms that software testers use to describe how they approach their testing are black-box testing and white-box testing [Patton 2001].

In **black-box** testing, the tester only knows what the software is supposed to do (he can not look in the box to see how it operates): he types in a certain input, he gets a certain output. He does not know how or why it happens, just that it does.

In **white-box** testing, the software tester has access to the program's code and can examine it for clues to help him with his testing (he can see inside the box). Based on what he sees, the tester may determine that certain cases are more or less likely to fail and can tailor his testing based on that information.

Since a Web application is just like any other software, these concepts can be applied to Web site testing.

The easiest starting point is treating the Web page or the entire Web site as a black-box. We do not know anything about how it works, we do not have a specification, we just have the Web site in front of us to test: there are all the basic elements-text, graphics, hyperlinks to other pages on the site, and hyperlinks to other pages play videos.

When planning the tests for a Web site, we take care to identify all the features of each page. For example, in order to satisfy usability properties, such as flexibility

and efficient navigation, we have to check for text layout issues by resizing browser windows to be very small or very large (this will reveal design errors where the designer or the programmer assumed a fixed page width or height), or to check if the page has too many links or frames (contextual and efficient navigation properties). Moreover, if we would like to consider accessibility problems, we can verify the presence of a link of textual version, the alternative texts for images and graphical links, etc.

So, we can say that black-box testing is achieved by user testing, in which users, or evaluators, use and observe directly the behaviour of Web site, without considering the code of Web pages.

All examples that we have mentioned above are referred to static Web pages, but more and more the content of Web site are built dynamically. Thus, several features could be more effectively tested with a white-box approach. Of course, they could also be tested as a black-box, but the potential complexity is such that it is almost impossible to be sure to find the important design mistakes without some knowledge of the Web site's system structure and implementation, such as the dynamic content, dynamically created Web pages, server performance and loading, etc.

In addition to HTML code (formed by only tags), a Web page can be created by using java, java script, cgi, perl etc. For this reason, in order to better investigate usability and accessibility problems, it is required to examine the code of the Web site. To do that, we have to analyse this code through a white-box approach.

2.2.2 Taxonomy of UE Methods

There exist several taxonomies proposed. The most commonly used taxonomy distinguishes between predictive and experimental techniques [Coutaz 1995]. Another one is based on the presence or absence of a user and computer [Whitefield et al. 1991]. These two classification schemes, however, do not consider automation aspects. The first taxonomy which weighs up automation is by Balbo [Balbo 1995]. This classification distinguishes among four approaches to automation in dependence on which level the automation is applied. So, Balbo uses the following categories to classify 13 UE methods:

- *Non Automatic* - no level of automation supported (i.e., specialist evaluator performs method).
- *Automatic Capture* - software automatically captures interface usage (e.g., logging). Methods that use this automation rely on software facilities to record relevant information about the user and the system.
- *Automatic Analysis* - automatic identification of usability problems. These Methods are able to identify usability problems automatically.
- *Automatic Critique* - automatic analysis coupled with automated suggestions for improvements. Methods which not only point out difficulties but propose suggestions.

To facilitate our discussion of the state of automation in usability evaluation, UE methods can be analysed through the following four dimensions: Method Class, Method Type, Automation Type and effort level [Ivory and Hearst 2001].

a) **Method Class – T**he type of evaluation conducted at a high level.

In this dimension UE methods can be classified into 5 method classes.

- *Testing*: an evaluator observes users interacting with an interface (i.e., completing tasks) to determine usability problems.
- *Inspection*: an evaluator uses a set of criteria or heuristics to identify potential usability problems in an interface.
- *Inquiry*: users provide feedback on an interface via interviews, surveys, etc.
- *Analytical Modelling*: an evaluator employs user and interface models to generate usability predictions.
- *Simulation*: an evaluator employs user and interface models to mime a user interacting with an interface and report the results of this interaction (e.g. simulated activities, errors and other quantitative measures).

UE methods in the testing, inspection, and inquiry classes are appropriate for identifying specific usability problems, and for obtaining general assessments of usability. Analytical modelling and simulation are engineering approaches to UE that enable evaluators to predict usability with user and interface models. Software engineering practices have had a major influence on the first three

classes, while the latter two are quite similar to performance evaluation techniques used to analyse the performance of computer systems [Ivory and Hearst 1999; Jain 1991].

b) **Method Type. -** How the evaluation is conducted within a method class. Method type indicates how the evaluation (capture and analysis) is conducted. For instance, if we use a user testing, we can record user actions by video types, or we can adopt remote evaluation by log files. Or, if we can proceed by inspection evaluation, we can use heuristics, or guidelines.

c) **Automation Type -** The evaluation aspect that is automated. This dimension specifies which aspects of usability evaluation are automated, that is which phases are automated. We discuss better this aspect in 2.3.

d) **Effort level -** The type of effort required to execute the method.

Even if we consider Balbo's automation taxonomy, a certain human effort is required to execute those non-automated phases of the method. In order to estimate needed effort, an attribute called effort level is added to each method. Thus, this attribute can assume the following values:

- *Minimal Effort*: does not require interface usage or modelling.
- *Model Development*: requires the evaluator to develop a UI model and/or a user model in order to employ the method.
- *Informal Use*: requires completion of freely chosen tasks (i.e., unconstrained use by a user or evaluator).
- *Formal Use*: requires completion of specially selected tasks (i.e., constrained use by a user or evaluator).

2.3 Methods for Automated Evaluation of Web Sites

As we have already shown, Usability evaluation is an increasingly important part of the iterative design process.

In the prior paragraph, we showed that there exist many UE methods, which can reveal several kinds of usability problems, and can be applied to different contexts depending on particular objectives of evaluation.

When we consider evaluation of user interfaces, two important issues rise: when usability evaluation should be applied, and, how many user tests should be made.

About the first question, evaluation should be performed since the first steps of the life cycle so that the possible problems can be removed early. However, when the product is finished, other tests are needed. Thus, there is the second issue: a precise number of evaluations does not exist. For example, if after the first test some problems are revealed, and thus some modifications are made, at least another evaluation is required, in order to see if the problems have been removed, and no others have been introduced.

For this reason, it is advisable repeating iteratively design-evaluation process, to improve the product. This is an example of how the automation can have a significant role during the process of development.

Another question is deciding which methods are more adequate to identify more problems. We showed that there exist many techniques which can reveal different problems. So, the adequacy of a certain technique depends on the context and on the kind of problems to detect: how much users participate in the evaluation, how many phases are automated, where are users/evaluators, etc.

In the prior section we mentioned that there exist several method class and types, we would like to consider the following: user testing, inspection-based evaluation and inquiry.

2.3.1 Automating Evaluation

As we already said, in the evaluation process many efforts are required. One solution to this inconvenient is to try to automating all or some process of the evaluation.

Automation of usability evaluation has several potential advantages, such as reducing the cost of usability evaluation, increasing consistency of the errors uncovered, predicting time and error costs across an entire design, reducing the need for evaluation expertise among individual evaluators, increasing the coverage of evaluated features, etc.

In [Abascal 2002] there is a valuable example of how an automatic support can help designers and evaluators in their work in order to reduce the requested efforts. USERFfit tool is a tool to facilitate design for all. It has been developed to support Userfit methodology. USERfit is a design methodology with a particular orientation towards usability and accessibility. It could be defined as an aid environment for the design has been developed within the area of Assistive Technology. The main goal of USERfit is the capture and specification of user requirements. This methodology uses paper-based forms to store and propagate the design related information. For this reason, some issues, such as the inclusion and elimination of new users or the need to propagate the results between forms, make the specification process tedious. So, USERfit tool has been developed in order to facilitate the use of the USERfit methodology. It automates the most tedious part of the design process. In addition USERFfit tool supports the reuse of previously developed materials and the sharing of design information among remote groups of designers, maintaining coherence and compatibility.

Another tool aimed at accessibility automatic support is the AVANTI browser, developed by Stephanidis et al. [1997]. This browser refers to the concepts defined for User Interfaces for All: it employs adaptability and adaptivity techniques, in order to provide accessibility and high-quality interaction to users with different abilities, skills, requirements and preferences (e.g., users blind and with other disabilities). It incorporates techniques for changing the presentation of the contents of a Web page, i.e. the generation of a list of large push buttons containing the links of a page for motor-impaired users. Moreover, The AVANTI browser embeds some techniques that are similar to some checkpoints of the criteria proposed in this work (see par. 5.2, checkpoints 1.5.2, 3.2.1, and 3.3.1). In fact, in the AVANTI system, adaptability refers to the process of selecting / modifying aspects of the user interface during the initiation of each interaction session, according to the user characteristics that are known prior to interaction (e.g. user abilities). A certain mechanism for the appropriate style creates / modifies specific "portions" of the user interface, such as size, colour, volume, etc., which are similar to the features suggested by us for low vision users.

A certain mechanism for the appropriate style(s) creates / modifies specific "portions" of the user interface, such as size, colour, volume, etc., which are similar features we suggested for low vision users.

An important benefit derived from automation testing is incorporating evaluation within the design phase of development; as we said before, this is important because evaluation with most non-automated methods can typically be done only after the interface or prototype has been built and changes are more costly [Nielsen 1993].
Modelling and simulation tools make it possible to explore designs earlier.
It is important to consider automation as a useful complement to standard techniques, and not as a substitute.
Using Balbo's automation taxonomy [Balbo 1995], we can have:

- *None*: no level of automation; all aspects of usability evaluation are performed only by evaluators.
- *Capture*: The first phase of evaluation is supported by a software tool, which automatically records in log files usability data.
- *Analysis*: since captured data, a specific tool analyses them and automatically detects potential usability problems.
- *Critique*: automating this step is somewhat difficult. Software automates analysis and suggests improvements.

Considering assessments of automated capture, analysis and critique techniques, can be used the following criteria:

- *Effectiveness*: how well a method discovers usability problems;
- *Ease of use*: how easy is a method to employ;
- *Ease of learning*: how easy is a method to learn;
- *Applicability*: how widely applicable is a method to WIMP and/or Web UIs other than those originally applied to.

Although automated usability evaluation has great promise as a way to augment existing evaluation techniques, it is greatly under-explored.

2.3.2 Overview of Usability Evaluation Methods

Ivory [Ivory and Hearst 2001] in her work surveyed 75 UE methods applied to WIMP interfaces, and 57 methods applied to Web UIs. Of these 132 methods, only 29 apply to both Web and WIMP UIs.

Next table shows that automation in general is greatly under explored.

Method Class / Method Type	Automation Type: None	Capt.	Anal.	Crit.	Description
Testing					
Thinking-aloud Protocol	F (1)				user talks during test
Question-asking Protocol	F (1)				tester asks user questions
Shadowing Method	F (1)				expert explains user actions to tester
Coaching Method	F (1)				user can ask an expert questions
Teaching Method	F (1)				expert user teaches novice user
Co-discovery Learning	F (1)				two users collaborate
Performance Measurement	F (1)	F (3)			tester records usage data during test
Log File Analysis			FIM (9)		tester analyzes usage data
Retrospective Testing	F (1)				tester reviews videotape with user
Remote Testing		FI (3)			tester and user are not co-located during test
Inspection					
Guideline Review	F (4)		(5)	(6)	expert checks guideline conformance
Cognitive Walkthrough	F (2)				expert simulates user's problem solving
Pluralistic Walkthrough	F (1)				multiple people conduct cog. walkthrough
Heuristic Evaluation	F (1)				expert identifies heuristic violations
Perspective-based Inspection	F (1)				expert conducts narrowly focused heur. eval.
Feature Inspection	F (1)				expert evaluates product features
Formal Usability Inspection	F (1)				experts conduct formal heuristic evaluation
Consistency Inspection	F (1)				expert checks consistency across products
Standards Inspection	F (1)				expert checks for standards compliance
Inquiry					
Contextual Inquiry	FI (1)				interviewer questions users in their env.
Field Observation	FI (1)				interviewer observes system use in user's env.
Focus Groups	FI (1)				multiple users participate in a disc. session
Interviews	FI (1)				one user participates in a disc. session
Surveys	FI (1)				interviewer asks user specific questions
Questionnaires	FI (1)	FI (1)			user provides answers to specific questions
Self-reporting Logs	FI (1)				user records UI operations
Screen Snapshots	FI (1)				user captures UI screens
User Feedback	FI (1)				user initiates comments
Analytical Modeling					
No Methods Surveyed					
Simulation					
Information Proc. Modeling			M (1)		mimic user interaction
Information Scent Modeling		M (1)			mimic Web site navigation
Automation Type					
Total	26	4	3	1	
Percent	76%	12%	9%	3%	

Table 1 Automation support for 57 Web UE methods. A number in parentheses indicates the number of UE methods surveyed for a particular method type and automation type. The effort level for each method is represented as: minimal (blank), formal (F), informal (I) and model (M). Two software tools provide automation support for multiple method types: Dome Tree visualization - log file analysis and information scent modeling; and WebVIP - performance measurement and remote testing.

Methods without automation support represent 67% of the methods surveyed, while methods with automation support collectively represent only 33%. Of this 33%, capture methods represent 13%, analysis methods represent 18% and critique methods represent 2%. All but two of the capture methods require some level of interface usage.

To provide the fullest automation support, software would have to critique interfaces without requiring formal or informal use. Ivory's survey found that this level of automation has been developed for only one method type: guideline review, which we well discuss later.

Now, in the next paragraphs, we would like to describe how testing, inspection and inquiry for Web evaluation are conducted.

2.3.3 User Testing Methods

Usability testing with real participants is a fundamental usability evaluation method [Nielsen 1993; Shneiderman 1998]. It provides an evaluator with direct information about how people use computers and their problems with the interface being tested. During usability testing, participants use the system or a prototype to complete a pre-determined set of tasks while the tester records the results of the participants' work. The tester then uses these results to determine how well the interface supports users' task completion as well as other measures, such as number of errors and task completion time. Thus, user interface events are elements which can be captured and analysed in order to measure and detect usability issues.

User interface events (UI events) are generated as natural products of the normal operation of window- based user interface systems such as those provided by the behaviour with respect to the components that make up an application's user interface (e. g., mouse movements with respect to application windows, keyboard strokes with respect to application input fields, mouse clicks with respect to application buttons, menus, and lists). Because such events can be automatically captured and because they indicate user behaviour with respect to an application's user interface, they have long been regarded as a potentially fruitful source of information regarding application usage and usability. However, because user interface events are typically extremely voluminous and rich in detail, automated support is generally required to

extract information at a level of abstraction that is useful to investigators interested in analysing application usage or evaluating usability.
Automation has been used predominantly in two ways within user testing: automated capture of use data and automated analysis of this data according to some metrics or a model.

Automated capture phase

Automated capture methods represent important first steps toward User Interface improvements, because they provide input data for analysis and, in the case of remote testing, enable the evaluator to collect data for a larger number of users than traditional methods. When evaluators assess the interface usability, they have to collect the user actions in order to examine them.
This can be done by an evaluator taking notes while the participant uses the system, either live or by repeatedly viewing a videotape of the session. Because both are time-consuming activities, automation is used as it represents a useful instrument.
Within the user testing class of UE, automated capture of usage data is supported by two method types: performance measurement and remote testing.
Performance measurement methods record usage data (e.g., a log of events and times when events occurred) during a user test. Video recording and event logging tools are available to automatically and accurately align timing data with user interface events. Such evaluation requires many efforts and much time by evaluators who have to examine video-types and records. Furthermore, if the user actions are recorded in log files, data records produce voluminous log files and make it difficult to map recorded usage into high-level tasks.
Another way to conduct the capture phase is remote testing.
Remote testing methods enable testing between a tester and participant who are not co-located. In this case the evaluator is not able to observe the user directly, but can gather data about the process over a computer network. Remote testing methods are distinguished according to whether or not a tester observes the participant during testing or not: Same-time different-place and different-time different-place are two major remote testing approaches used in UE methods.
Both these approaches can be employed for remote testing of Web UIs.

In the first case, the tester observes users exercising the application; software makes it possible for the tester to interact with the participant during the test, which is essential for techniques like the question-asking or thinking aloud protocols that require such interaction.

Otherwise, the tester does not observe the user during different-time different-place testing. A method type of this approach is the journaled session [Nielsen 1993], in which software guides the participant through a testing session and logs the results.

Web servers maintain usage logs and automatically generate a log file entry for each request (e.g. the IP address of the requester, request time, name of the requested Web page, etc). Since server logs cannot record user interactions that occur only on the client side (e.g., use of within-page anchor links or back button), and the validity of server log data is questionable, due to caching by proxy servers and browsers [Etgen and Cantor 1999], Client-side logs are used. Client-side logs capture more accurate, comprehensive usage data than server- side logs because they allow all browser events to be recorded. This approach requires every Web page to be modified to log usage data, or else use of an instrumented browser or special proxy server.

Examples of tools which capture client- side usage data, are The NIST WebMetrics tool suite (WebVIP, VISVIP and WebCAT), WebSat, WebRemUsine [Paternò and Paganelli 2001] and so on.

Automated analysis phase

After having captured user data, we have to analyze them. Approaches which we can use in order to analyse log files are:

- *Metric-based approaches*: they generate quantitative performance measurements. In general, performance measurement approaches focus on server and network performance, but provide little insight into the usability of the Web site itself. Service Metrics' tools, for example, can collect performance measures from multiple geographical locations under various access conditions, (e.g. performance bottle- necks, such as slow server response time), that may negatively impact the usability of a Web site.

- *Pattern-matching approaches*: These approaches analyse user behaviour captured in logs, for examples, detecting repeated user actions (e.g., consecutive invocations of the same command and errors), that may indicate usability problems. However, in several evaluation methods, pattern is matched in conjunction with task models.

- *Task-based approaches*: These approaches analyse discrepancies between the designer's anticipation of the user's task model and what a user actually does while using the system. For instance, USINE [Lecerof and Paternò 1998] employs the ConcurTaskTrees notation (see 1.2) to express temporal relationships among UI tasks (e.g., enabling, disabling, and synchronization). Using this information, USINE looks for precondition errors (i.e., task sequences that violate temporal relationships) and also reports quantitative metrics (e.g., task completion time) and information about task patterns, missing tasks and user preferences reflected in the usage data. So, USINE processes log files and outputs detailed reports and graphs to highlight usability problems. Remusine [Paternò and Ballardin 1999] is an extension that analyzes multiple log files. WebRemUSINE [Paternò and Paganelli 2001] is a more recent extension of this approach to evaluating Web sites. The tool also provide quantitative measurements regarding task performance.

- *Inferential Analysis*: Inferential analysis of Web log files includes both statistical and visualization techniques. Statistical approaches include trace-based and time-based analysis. However, statistical analysis is largely inconclusive for Web server logs, since they provide only a partial trace of user behaviour and time estimates may be skewed by network latencies. Visualization is also used for inferential analysis. It enables evaluators to filter, manipulate, and render log file data in a way that ideally facilitates analysis. Visualizations provide a high-level view of usage patterns (e.g., usage frequency, correlated references, bandwidth usage, HTTP errors and patterns of repeated visits over time) that the evaluator must explore to

identify usability problems. However, there is no discussion of how effective these approaches are in supporting analysis.

2.3.4 Inspection Evaluation

Usability inspection is the generic name for a set of methods that are all based on getting evaluators to inspect a user interface.

A usability inspection is an evaluation methodology whereby an evaluator examines the usability aspects of a UI design with respect to its conformance to a set of guidelines. Typically, usability inspection is aimed at finding usability problems in the design, though some methods also address issues like the severity of the usability problems and the overall usability of an entire system. Unlike other UE methods, inspections rely solely on the evaluator's judgment.

Common non-automated inspection techniques are heuristic evaluation [Nielsen 1993]. Heuristic evaluation is a discount usability engineering method for quick, cheap, and easy evaluation of a user interface design.

Heuristic evaluation is the most popular of the usability inspection methods: it is done as a systematic inspection of a user interface design for usability. The goal of heuristic evaluation is to find the usability problems in the design so that they can be attended to as part of an iterative design process. This evaluation involves getting a small set of evaluators to examine the interface and judge its compliance with recognized usability principles (the "heuristics").

However, automation is more and more used, also because it is difficult to follow guidelines and/or standards. Evaluating usability by verifying if guidelines or standards are respected, aids to automate the evaluation. In fact, software tools assist evaluators with guideline review by automatically detecting and reporting usability violations and in some cases making suggestions for fixing them [Balbo 1995].

It is important to note that it does not necessarily imply that a Web site not addressing guidelines is unusable. Nor it is proved that a Web site addressing all guidelines is the most usable site.

Capture, analysis and critique phases

We said several times that automation involves capture and analysis, and in some cases even critic phase.

In the capture phase, a system assists an evaluator with a cognitive walkthrough: during this phase evaluator attempts to simulate a user's problem-solving process while examining UI tasks, and, at each step of a task, he produces documentation about the assessment of whether a user would succeed or fail to complete the step. For analysis phase, several tools have been developed. For instance, The Rating Game [Stein 1997] is an automated analysis tool that attempts to measure the quality of a set of Web pages using a set of easily measurable features (e.g., information feature, graphics feature, gadgets feature, etc). The tool reports these raw measures without providing guidance for improving a Web page. A similar structural analysis at the site level can be applied. A structural analysis can focus on verifying that the breadths and depths within a page and at the site level fall within thresholds. However, the analysis which is more used is by assessing the code according to a number of guidelines. Some tools are able to analyse one page by time, but it would be useful verifying the whole Web site, also in order to identify potential problems in interactions between pages.

Finally, about the critique phase: critique systems give designers clear directions for conforming to violated guidelines and consequently improving usability. Following guidelines is difficult, especially for large guidelines. Automated critique approaches, especially those ones that modify a UI [Balbo 1995], provide the highest level of support for adhering to guidelines.

Conforming to the guidelines embedded in HTML analysis tools can potentially eliminate usability problems that arise due to poor HTML syntax (e.g., missing page elements) or guideline violations.

Several automated critique tools use guidelines for Web site usability checks: Most tools check that HTML code conforms to standards or guidelines (e.g., Consortium's HTML Validation Service, Dr. Watson, WebSAT, Bobby and so on), but not all provide suggestions to improve Web sites. UsableNet's LIFT Online and LIFT Onsite [Usable Net 2000] provide valuable guidance for improving Web sites: LIFT Online suggests improvements, while LIFT Onsite guides users through making

suggested improvements. However, critique approaches cannot assess aspects that cannot be operationalized, such as whether the labels used on elements (e.g., link or images) will be understood by users. For this reason, users are often involved in the improvement process. Thus, it means that fixing automatically the problems or guidelines violations, is quite difficult.

Guidelines

As we have already mentioned above, guidelines review is one of the methods most used in conducting inspection evaluation.

We hereby define a *Web usability guideline* as any statement ensuring some adequacy of a particular user interface of a Web site with respect to a particular context of use where a given user population has to fulfil interactive tasks with a given system [Scapin et al. 2000A].

Generally speaking, guidelines are mostly used during [Vanderdonckt 1999]:

- *The specification phase* - a set of guidelines is delimited as requirements for the future UI;
- *The design phase* - guidelines are exploited in order to decide an appropriate value for each design option by considering the context;
- *The prototyping phase* - guidelines are exploited to obtain as soon as possible a static or working UI prototype that can be showed, tested and evaluated;
- *The programming phase* - guidelines are gathered to guide, orient, decide, ensure a UI development;
- *The evaluation phase* - the resulting UI is evaluated with respect to guidelines that are often ones that have been selected in previous phases;
- *The documentation and certification phase* - guidelines which have been manipulated in previous phases are instructed for documenting an interactive application for communication, reuse, maintenance or commercial promotion purposes.

Many guidelines have been published in document sources throughout the literature. These sources fall into five categories [Vanderdonckt 1999]:

- *Design rules*: they comprise a set of functional and/ or operational specifications that specify the design of a particular user interface. These

specifications are presented in a form that requires no further interpretation, either from designers or from developers.

- *Compilations of guidelines*: they comprise several prescriptions written for a wide range of user interfaces. Each prescription is presented as a statement, sometimes along with examples, with or without clarifying explanations and comments. Each prescription generally results from a human consensus between guideline users. They can range from a small set of guidelines dedicated to a particular usability feature.
- *Style guides*: they comprise a set of guidelines and/ or functional or non-functional specifications aiming at consistency for a collection of distinct user interfaces.
- *Standards*: they comprise a set of functional and/ or operational specifications intended to standardize design. Standards are promulgated by national or international organizations for standardization.
- *Ergonomic algorithms*: these aims to systematize one design aspect by building it in the same way, according to a same base of design rules. Such algorithms are primarily intended to design a series of Web pages that automatically respect some guidelines (usually, design rules or style guides) by construction.

Contrarily to the domain of usability for graphical user interfaces, Web usability guidelines in these categories are often produced by consensus, common sense or observation of best practices. Guidelines validated by experimental results are rare and hard to find out. Web usability guidelines tend to address Web specific usability issues, but not necessarily usability issues for graphical user interfaces adapted or specialized for the Web. However, we can note that “usability guidelines that are valid for graphical user interfaces must also be considered as potential usability guidelines for the Web, unless some contradiction or invalidation is appearing” [Scapin et al 2000A].

Also for accessibility issues, guidelines have been proposed [WAI 1999]. "Web Content Accessibility Guidelines 1.0", published by the Web Accessibility Initiative [WAI 1999], are intended for all Web content developers (page authors and site designers) and for developers of authoring tools (see section 1.2).

Even if guidelines are useful for automated inspection, they are insufficient. They should not be considered in isolation: often guidelines need to be supplemented by a suitable method and a clear process that leads them to unambiguous interpretation.
Although guidelines can be used in conjunction with guidelines checklists, standard inspection, and consistency inspection, a variant of heuristic evaluation has been set up where the usual set of 10 heuristics [Nielsen and Mack 1994] is replaced by a set of specific guidelines. This replacement came from the observation of heuristics which, though cost-effective, have been considered too general and not expressive enough to be interpreted and applied effectively. Conversely, lists of guidelines are potentially long. If they are not specific, a lot of time to interpret them is likely to be devoted since the whole user interface and its components should be checked against them. However, if developing a tool for automated or computer-aided evaluation of these guidelines is the ultimate goal, then such shortcomings are important and hard to solve. If providing people with assistance and guidance in applying and evaluating guidelines is the ultimate goal, then shortcomings are less important [Scapin et al. 2000A].

2.3.5 Inquiry Evaluation

The goal of inquiry methods is to gather subjective impressions (i.e., preferences or opinions) about various aspects of a UI. Hence, this methodology involves users similar to user testing approaches: inquiry methods require feedback from users and are often employed during user testing.
Inquiry methods vary based on whether the evaluator interacts with a user or a group of users or whether users report their experiences using questionnaires or usage logs, possibly in conjunction with screen snapshots. Evaluators also employ inquiry methods, such as surveys, questionnaires, and interviews, to gather supplementary data after a system is released; this is useful for improving the interface for future releases.
It is important making it very easy to provide some kind of feedback to the user (beside the automated thank you notes currently in vogue) [Nielsen 1999]. In any case, the survey should not be long, so the user can fill it out quickly and move on. Evaluator will get many more responses.

Automation has been used predominantly to capture subjective impressions during formal or informal interface use.

Automation support for inquiry methods makes it possible to collect data quickly from a larger number of users than is typically possible without automation.

The Web inherently facilitates capture of questionnaire data using forms: Users are typically presented with an HTML page for entering data (e.g., WAMMI, QUIS or NetRaker), and a program on the Web server processes responses.

Unfortunately, inquiry methods have some limitations: they may not clearly indicate usability problems due to the subjective nature of user responses. Furthermore, they do not support automated analysis or critique of interfaces.

2.4 Automatic Tools for Web Evaluation

In paragraph 2.3, we have generally discussed about Web testing automation, focusing especially on methods to identify usability problems. Now, we would like to report some tools used for accessibility evaluation. Most of them are directed to check accessibility, but tools such as WebSAT and Dr Watson, are also used for usability features.

2.4.1 Developing a Tool Based on Guidelines

To develop any tool for working with guidelines, five development milestones have been identified for reference and comparison purposes [Vanderdonckt 1999], but also for structuring the process to reach that goal.

The five development milestones, through which we must pass to produce a high quality tool for working with guidelines, are the following:

a) **Guidelines collection.**

The goal of this first step is to gather a useful subset of guidelines suitable for designing Web pages. An initial unstructured but comprehensive set of guidelines is formed by collecting, gathering, merging, compiling guidelines from all available world- wide ergonomic sources. These different sources are not especially dedicated to Web usability (i.e., they focus more on

graphical user interface in general), thus requiring some modification, adaptation, extension, and so on.

This first milestone is represented, in this work, by the collection of usability criteria reported in chapter 3.

b) **Guidelines organization.**

In this step, because the initial set of guidelines is copious and ranges over many levels of rigour and credibility, guidelines are classified in a good organizational structure. Since the initial set, guidelines are organized proceeding by two activities:

- *Classifying each guideline by ergonomic criteria.*

  An ergonomic criterion is hereby defined as a well- recognized usability dimension in human- computer interaction whose reliability effectiveness and impact on usability have been experimentally assessed. Each guideline has therefore been classified by a sole ergonomic criterion based on its definition. By this classification designers have a first idea on when and where the related guideline can be applied as well as some first idea of its absolute level of importance.

- *Further classifying each guideline by alternate index keys.*

  As the set resulting by first step is still wide, guidelines are further classified by alternate index keys. This classification allows multiple and flexible access paths to each guideline, rather than merely by ergonomic criteria [Bastien et al. 1999]. Such accesses permit automatic identification of a certain guideline, so that it can be evaluated more rapidly (e.g., Navigation structure, Links/Organization, etc). Each guideline is then assigned to one or many methods and techniques for Web site design/evaluation that can be effectively used to assess the guideline (i.e., to indicate which level can be used for automation). Moreover, to each guideline can be attached a score in order to express the impact of a violated or respected guideline.

This second milestone coincides with the assignment of usability criteria to 3 categories (chapter 3) and their subsequent organization into checkpoint sets (chapter 5).

c) **Incorporation of guidelines into approach.**

It is now important to associate sections of guidelines with the different phases of a development life cycle.

In this way, it is expected that sets of guidelines suitable for each phase can be more easily identified and accessed. Thus, the goal of this step is to locate points within these phases where organized guidelines should be considered, to specify which should be considered by identifying local guidelines (for a phase), global guidelines (for all phases) or pervasive ones (for several continuous phases).

This milestone is represented by the method for heuristic evaluation explained in chapter 8.

d) **Operationalization.**

Guidelines incorporated into a development life cycle are typically used manually. To embody them in a software tool, a further stage of guidelines Guidelines operationalization is thus aimed at re-expressing in a more formal way, so that it results easier to evaluate them automatically.

This forth milestone was realized through a formal definition of evaluation and repair functions (chapter 6) and an overview of the automated tool implementing such functions (chapter 9).

e) **Guidelines use.**

We have no good knowledge about the usability of software tools for working with guidelines. And yet, it seems fundamental for HCI researchers that the UI of software tools for working with guidelines is usable. An empirical study of the usability of such tools would be of great benefit.

This last milestone was followed by giving examples of the application of the criteria to existing Web sites (chapter 4) and by verifying their effect on the interface for end users (chapter 7).

Above, we have reported general description about the five steps to follow in order to develop a tool using guidelines. However, Vanderdonckt in his work [Vanderdonckt 1999] suggests to consider four key aspects of each step:

- *Definition and goals* - the transformation between milestones in terms of its main goals when transforming input resources into output results;
- *Procedure* - the concrete actions that must be carried out;
- *Problems and analysis* - report known problems risen by applying this procedure, and reporting different solutions and perspectives.

Furthermore, he points out another activity which we should do for each step, namely validation: we have to perform some step validation to check its results and their conformity with the previous transformation steps. In order to do that, three properties will be assessed:

- **Completeness** – it should let us check that all step results still hold the desired properties (e.g., check if guidelines resulting from one step are still necessary and/ or sufficient to solve a specific problem).
- **Consistency** – it should let us verify whether the step results are internally contradictory or contradictory with results provided by previous steps.
- **Correctness -** it should let us verify whether the step results are not subject to intrinsic errors and bias.

Many designers complained namely that users and task characteristics are not considered in the guidelines usability testing because guidelines related to these aspects do not have access to relevant information or because they are merely too difficult to interpret or to implement [Löwgren and Laurén 1993]. As this information cannot be retrieved from HTML code, other sources might be investigated. Thus, user data collected in log files could be used to consider the adequacy on the current utilization of a Web site with respect to the context of use, including cognitive capabilities and preferences of users. However, approaches

which consider log files comparing them to a task (or user / dialog) models, do not embed guidelines.

Therefore, we can imagine a progress of dimension of guidelines independently of dimension of involved models, rather then a possible development of both combined methodologies.

2.4.2 The State of the Art of Existing Inspection Tools

In this paragraph we report a collection of tools for accessibility and usability evaluation based on guidelines. Most tools are designed to evaluate Web sites, but the research is aimed to develop tools able to repair Web pages. So there are:

Evaluation tools - Perform a static analysis of pages or sites regarding their accessibility, and return a report or a rating.

Repair tools (*) - Once the accessibility issues with a Web page or site have been identified, these tools can assist the author in making the pages more accessible.

Name	**Description**
AccVerify	"AccVerify" implements programmatic verification and reports all errors/non-compliance with the standards, plus checklist for criteria that can't be verified programmatically. Verifies the "all else fails" text version. Differentiates between 508 and W3C standards. http://www.hisoftware.com/access/
AnyBrowser.com	Tools relevant for accessibility include viewing in various screen sizes, view with images are replaced by ALT text. Also HTML and link validation, search engine tools, and other browser compatibility tests. http://www.anybrowser.com/
Bobby	Developed by CAST, Bobby helps authors determine if their sites are accessible. It does this through automatic checks as well as manual checks. http://www.cast.org/bobby/

Colourfield	Allows designers to model and predict image readability for colour deficient viewers. Developed by Colourfield Digital Media (2000). http://www.colourfield.com/insight.html
Design advisor	It is a critic tool based advisors that do not automate design, but attempt to spot and alert users to design problems which they can then fix.
Doctor HTML	performs minimal accessibility checking ("alt" on IMG) but it also verifies links, spell checks and performs some syntax checking (1997). http://www.imagiware.com/RxHTML/index_noframes.html
Dr Watson	Watson can check many other aspects of your site, including link validity, download speed, search engine compatibility, link popularity, word count, and spelling. No specific accessibility checking. http://watson.addy.com/
EvalIris	EvalIris is a Web Service that has been designed to evaluate the accessibility of HTML pages. The analysis is based on easily updateable accessibility criteria, making this tool able to easily incorporate new sets of guidelines and updatings. It can be used either as an on-line stand-alone application, or either as a Web service. In this last case, it can be used by any other application to perform accessibility analysis and, in this way, to extend its features. http://www.sc.ehu.es/acwbbpke/evaliris.html
InSight	Interactive evaluation tool designed to help developers to create accessible Web pages. http://www.ssbtechnologies.com/
KWARESMI	It is a Tool for Automatic Evaluation of Web-oriented guidelines. It should enable an evaluator to define and redefine evaluation logic of a guideline if needed, again, without the need to modify and recompile the evaluation engine.

	http://www.isys.ucl.ac.be/bchi/research/Kwaresmi.htm
Lift	LIFT Onsite is software that allows Web designers and Web owners to test and repair accessibility and usability issues, including site navigability, download speed, graphic quality, accessibility, searchability, etc (Usablenet, 2001). http://www.usablenet.com/
WebLint	Weblint is a syntax and minimal style checker for HTML: a perl script which picks fluff off HTML pages. http://www.Weblint.org/
WebSAT	The Web Static Analyzer Tool uses a subset of usability guidelines to analyze a page for accessibility, form use, performance, maintainability, navigation, and readability. http://zing.ncsl.nist.gov/WebTools/WebSAT/overview.html
A-Prompt *	from the University of Toronto may be used in several ways. It both identifies problems and helps the author to correct them. http://aprompt.snow.utoronto.ca/
ALT repair kit *	developed by Sonicon, allows ALT text to be added to page inline. http://www.sonicon.com/wai/altsform.html

All of these tools use inspection techniques in according to standards, rules and guidelines. The tool automatically checks the entire site or a single Web page, analyzing html code and comparing tags to a specific guideline.

For example, to check if an alternate text for images has been used, the tool verifies the presence of “ALT” tag, or “longdesc” tag.

Farenc et al. (1996) showed that automatic evaluation of graphical user interfaces against guidelines would reach a plateau effect around 44%, thus meaning that only one part of guidelines can be checked automatically. In Cooper et al., it was showed that more than half of usability guidelines for Web sites could be addressed, automatically, semi-automatically, or in a mixed-initiative fashion. The portion was more important due to the fact that Web sites are more accessible and analyzable in principle than compiled applications running on top of a window manager.

However, the Brajnik's survey [Brajnik 2000] revealed that most tools, including Bobby and Lift Online, address only a sparse set of usability features, such as download time, presence of alternative text for images, and validation of HTML and links. Other usability aspects, such as consistency and information organization are unaddressed by existing tools.

In order to harmonize the ways different tools treat the same checkpoints, some developers (such as Chris Ridpath (A-Prompt) and Michael Cooper (Bobby)) are now working together (Vanderdonckt, pers. com.).

Bobby

Bobby [Clarck and Dardailler 1999] is an HTML analysis tool that automatically checks a Web page or a series of Web pages against accessibility guidelines promoted by the Web Accessibility Initiative [WAI 1999]. Conforming to the guidelines embedded in these tools can potentially eliminate usability problems that arise due to poor HTML syntax (e.g., missing page elements) or guideline violations. Specifically, Bobby focuses on accessibility issues. This tool is available as online or downloadable version.

Although Bobby reports are usually long in order to cover the guidelines completely, this does not necessarily indicate major problems with the page.

Bobby's report highlights the detected accessibility problems catalogued by 3 priority levels defined by W3C: *Priority 1* (problems that seriously affect the page's usability by people with disabilities), *Priority 2* (problems which one should try to fix), and *Priority 3* (problems which one should also consider). In particular, initially in the report there are those problems due to Priority 1 error (marked by a question mark), that cannot be fully automatically checked. This indicates that the user will need to address that question manually. Moreover, clicking on any of the problems that Bobby reports will produce a more detailed description of how to fix the problem.

WebSAT

The Web Static Analyzer Tool (WebSAT) is a prototype tool that inspects the HTML composition of Web pages for potential usability problems. WebSAT allows the

usability engineer to investigate these potential problems so as to determine whether they should be purged from the design of the Web pages.

WebSAT inspects the HTML composition of Web pages against numerous usability guidelines. WebSAT can perform inspections using either its own set of usability rules or those of the IEEE Std 2001-1999. In either case, WebSAT expects as input the URL to a single Web page or to a whole site. The output for a single Web page is almost immediate (depending on the page size). In the case of a whole site, the length of time needed to analyze the site will depend on the number of pages that comprises the site; therefore, an email address must be supplied so that the destination of the result files can be identified upon completion.

WebSAT uses a set of *heuristic rules* to evaluate the use of tags in Web page design. Note: These rules do not form a comprehensive set of guidelines. However, they are a sample set of typical rules to demonstrate the feasibility (and limitations) of an automatic checker. The rules are grouped into six categories as follows: accessibility, form use, performance, maintainability, navigation and readability. Thus, the results are grouped by those categories.

Lift OnLine

LIFT Online tests Web pages against a subset of all usability and accessibility guidelines (rules), and then sends an email with the link to usability report online. More precisely, problems are identified by testing the pages of the site against a vast and growing collection of rules that were developed following the most recent research on Web usability and accessibility. An interesting feature of LIFT Online is that the rules can be customized. In fact, evaluator can customize rule behaviour so that the unique requirements of different types of sites and development stages can be addressed.

After the analysis is completed, LIFT Online shows a list of the pages in the Web site that may have problems. Each problem is ranked by severity and is described in detail.

EvalIris

EvalIris is a component of the IRIS Design Support Environment, developed by Abascal et al. (2002) within the IRIS European Project, with the goal of assisting designers in the production of accessible Web pages. This Web service was created to check the accessibility of Web sites based on the WAI's WCAG 1.0 guidelines. It has been implemented as a Web service in order to allow any other application to use it. The IRIS evaluation module EvalIris can use different sets of guidelines defined through the proposed XML scheme. This structure allows the tool to adequately handle any guideline or checkpoint that each HTML tag has to accomplish. In this way, EvalIris is also useful for the design of Web sites following a particular set of guidelines specified within this XML schema. As EvalIris is a Web service, it can only be used from other applications through a well-defined protocol, though it does not have a human interface. EvalIris receives the HTML mark-up, or obtains it through the Internet by using its URL, and directly evaluates the mark-up of the Web page dividing it into its HTML components (image, table, header…) then verifying, separately, the accessibility of each component. To this end, the mark-up is transformed into a tree format that provides more facility when processing and evaluating the accessibility of each HTML component. In order to perform the verification, EvalIris retrieves the necessary accessibility information (referred to each HTML component in the tree structure) from a native XML database. The application verifies whether the checkpoints returned by the database referring to the evaluated component can be automatically checked and tests the conformance of the component with the checkpoint. Eventual failures are reported as Errors, in case automatically verification can be performed. One of the key features of EvalIris is that the information stored in this repository can be easily updated. Therefore, new versions of accessibility guidelines can be accommodated into the system with no major effort. EvalIris has been evaluated by means of W3C's "Techniques For Accessibility Evaluation And Repair Tools" [http://www.w3.org/TR/AERT], including several text files with recognized accessibility barriers that are useful in verifying the performance of evaluation and repair tools. EvalIris is able to detect all the barriers in these documents.

KWARESMI

KWARESMI (Knowledge-based Web Automatic REconfigurable evaluation with guidelineS optiMIzation), developed by Beirekdar et al. [2002A], is a tool for Automatic Evaluation of Web-oriented guidelines. It is intended to enable the evaluation of any ergonomic guideline (for usability, for accessibility, etc.) as soon as we can find HTML elements that enable the evaluation of this guideline partially or totally. Therefore, the evaluation should not be restricted to specific types of guidelines.

KWARESMI tool is developed to support guideline definition language (GDL) as an automatic evaluation tool of Web ergonomic guidelines. The tool and the underlying GDL would overcome main shortcomings of existing automatic evaluation tools, like bobby, a-prompt, WebSat, etc.: the guidelines' evaluation logic is hard coded in the evaluation engine, leading to many limitations (e.g. difficulties in adding new guidelines).

So, the tool should allow the dynamic evaluation of any guideline that can be expressed in GDL without any code modification of the evaluation engine.

KWARESMI enables the evaluator to express guidelines in an evaluation-oriented manner in a declarative language. Rather than being imperative or procedural, the GDL language allows people to declare the way a guideline should be evaluated, not how it would be evaluated. The new guidelines structure is stored in the tool data base, thus separated from the evaluation engine [Beirekdar et al. 2002B].

GDL is a language aimed at structuring the guideline. Beirekdar et al. [2002A] proposed a framework considering the main concepts needed by an evaluator to structure a guideline towards (automatic) evaluation. This language is able to express guideline information in a sufficiently rich manner, thus enabling an evaluation engine to perform an automated evaluation of any GDL-compliant guideline by static analysis of the HTML code.

2.4.3 Task Models and Evaluation

Use of models can help in the design, development and evaluation of interactive applications.

There are many types of models, depending on the kind of information that they contain, the level of formality that we use, how we represent them, and the level of abstraction. One important design choice is to select the most appropriate model for the current goal.

Different models imply different representations as they have different users and purposes (e.g., the users of task models can be designers, developers, end users, managers and other actors involved in the task modelling phase).

We consider task model designed and developed by Paternò et al: there is a general agreement that task models are fundamental models in user interface design and most model-based proposals include some sort of task models.

Task models describe how activities can be performed to reach the users' goals when interacting with the considered application. They should incorporate the requirements raised by the people who should be taken into consideration when designing an interactive application (designers, software developers, application domain experts, end users, and managers).

More precisely, task models can be useful for different purposes:

- *Understanding an application domain*: as they require a precise identification of the main activities and their relationships, they help to clarify many issues that at the beginning may not be immediately recognized;
- *Recording the results of interdisciplinary discussions*: many people can be involved in the design of an interactive application: user interface designers, software developers, managers, end users, experts of the application domain. It is thus important to have a representation of the activities that can integrate all the requirements raised and supports focusing on a logical level that can be understood by all of them;
- *Designing new applications consistent with the user conceptual model*: because of the lack of structured methods supporting task-driven design it happened in various projects that first people developed task models for a new application and then they did not use them for driving the design. This made the task modelling an exercise with limited advantages whereas if the application was designed following a task-based approach it would have been

more usable because it would have incorporated the user requirements captured in the task model;

- *Analysing and evaluating usability of an interactive system*: task models can be useful in various ways to support the usability evaluation of an interactive application. They have been used to predict the users' performance in reaching their goals, or to support analysis of user behaviour to identify usability problems.
- *Supporting the user during a session*: creating a correspondence between tasks and the interaction objects composing the user interface can be useful also at run-time, for example to provide context-sensitive, task-oriented help systems.

Thus, task models can be useful both to analyse and evaluate an existing system or as a starting point to design new applications from scratch.

There are two main classes of people who receive the greatest benefits from task models, they are:

- *Designers* - task models provide high-level, structured approaches which allow an integrated framework to both functional and interactional aspects and to focus on relevant logical;
- *End users* - task models support the development of more usable systems, where it is easy to understand how the user interface aids the user activities because the physical actions supported by the user interface can be easily mapped onto logical actions and the representations provided can effectively support the possible tasks.

This means that task models can be useful both to improve the process of design and development and to obtain more usable interactive software applications.

Tasks, task analysis and task modelling

Tasks are activities that have to be performed to reach a goal. They can be either logical activities or physical activities.

A goal is either a desired modification of the state of an application or an attempt to retrieve some information from an application. For example, Accessing a flight's database to know what flights are available is a goal which does not require the modification of the state of the application, whereas Accessing a flight's database to add a new reservation requires a modification of the state of the application.
Thus, tasks and goals are closely connected: Each task can be associated with one goal, that is the goal achieved by performing the task. One goal can be achieved by performing one or multiple tasks. In some cases it is possible to choose among different tasks to achieve a certain goal.

We can distinguish between the task analysis and the task modelling phases. In terms of software engineering phases, task analysis can be useful to support the requirements phase whereas task models can be useful for the design phase. Generally speaking, we can say that the purpose of task model is to describe ways to perform the tasks identified in the task analysis phase.

The ConcurTaskTrees Notation

In model-based design the basic idea is to identify high-level models that allow designers to describe and analyze interactive software applications from a more semantic-oriented point of view rather than starting immediately to address the implementation level.
Such models can be represented by using formal or semi-formal notations.
One notation for the task model we can use is expressed by ConcurTaskTrees [Paternò et al. 1996]. In ConcurTaskTrees the task model is represented by a hierarchy of tasks where higher levels are more abstract and logical and lower levels are more refined and concrete and oriented to describe the physical actions required to interact with the user interface.
This notation allows designers to specify the temporal relationships among tasks and other information, useful for the design of the user interface.
The main purpose of this notation is to support the specification of flexible and expressive task models that can be easily interpreted even by people without formal background.

The main features of ConcurTaskTrees are:

- *Focus on activities:* thus it allows designers to concentrate on the most relevant aspects when designing interactive applications that encompass both user and system-related aspects avoiding low-level implementation details.
- *Hierarchical structure:* a hierarchical structure is something very intuitive, in fact often when people have to solve a problem they tend to decompose it into smaller problems still maintaining the relationships among the various parts of the solution.
- *Graphical syntax:* a graphical syntax often (though not always) is more easy to interpret, in this case it should reflect the logical structure, so it should have a tree-like form.
- *Concurrent notation:* A rich set of possible temporal relationships between the tasks can be defined. In ConcurTaskTrees two tasks can synchronize. This happens when they have to exchange information because the output information of one task is the input information for the other task.
- *Task allocation:* how the performance of the task is allocated is explicitly indicated by using icons.
- *Objects:* once the activities are identified, it is important to indicate the objects that have to be manipulated to support their performance.

In the ConcurTaskTrees notation there are four categories of tasks: User tasks (tasks performed by the user), Application tasks (tasks completely executed by the application), Interaction tasks (tasks performed by the user interacting with the system by some interaction techniques), and Abstract tasks (tasks which require complex activities whose performance cannot be univocally allocated).
It is also possible to identify a set of task types for each category. They are useful to distinguish tasks that raise different requirements in terms of presentation and interaction techniques supporting them.
So, we can have, for example, *planning* (a cognitive activity to organize a sequence of actions to perform), *comparing* (when the user has to evaluate some information), *problem solving* (when the user has to find a solution for a problem), and so on.

Furthermore, we can classify Interaction tasks into various types, such as *Selection task* (i.e. user can select one or more items from a set or range of items), *Edit tasks* (i.e. tasks that allow users to specify input data, and this information can be modified before being definitively sent to the application), *Control tasks* (the user triggers actions explicitly).
The same can be made for application tasks, which are used to indicate that the application performs an activity: overview, comparison, locating, processing feedback, grouping, etc.
More, it is possible especially to add further information to enrich the description of a task, such as informal description, information on time requested for its performance, state-related precondition, and so on.

Temporal relationships

The temporal relationships between the tasks are expressed by using operators of an extension of LOTOS notation (ISO 1988) which is a concurrent notation. It allows us to describe concurrent tasks. Two generic tasks, T1 and T2, are considered independently from their category (there is no limitation from this point of view).
By These operators we can describe interleaving (T1 ||| T2), synchronization (T1 |[ ]| T2), iteration (T1*), choice (T1 [ ] T2) and so on.

The hierarchical structure of the task model is represented in a tree-like structure even though the same task can appear in different parts of the model structure. This structure provides a description of possible tasks at different levels of abstraction, ranging from top (more abstract levels) to bottom (more detailed levels). At each level the temporal evolution should be followed reading from left to right. Also it is possible to go back at some point in a level of the structure by using iterative parent tasks.
Designers can be supported by a graphical editor (http://giove.cnuce.cnr.it/ctte.html) for creating the task tree and specifying relationships among tasks according to its syntax and semantics.

How to use task models in evaluation

In the prior paragraph, we mentioned about the concept of task models. As we said, Task models can give useful information for designing, developing and evaluating user interfaces.

Task analysis usually begins with investigating the users' current problem and breaking down the tasks that potential users of the system do or could do.

A task model describes the set of activities required to reach the users' goals and how these activities are related to each other.

We can use the task model to describe both logical activities and the use of the interaction techniques supported by existing user interfaces. Hence, it means that task models are useful during the design and development phases, but they can also give useful support to engineering the usability evaluation phase.

In this paragraph, we show how the task models can be used in UI evaluation describing the Paternò and Lecerof's work [Lecerof and Paternò 1998].

The advantage of the task model is that it indicates the tasks that can be performed at any time. This is due to the description of the temporal relationships among tasks.

When the user performs a task, he may change the set of available tasks to perform according to indications given by the model represented by the task tree.

By analyzing the task tree and considering the tasks the user has performed, it is possible to know which tasks at each time the user can perform.

These tasks, the available tasks, can be found due to the specification of the temporal relationships among tasks.

The available tasks can be used for evaluation purposes to find the errors performed by the user: an error occurs when the user tries to perform a task not allowed according to the ConcurTaskTrees specification. For instance, we can reveal when the user perform one task when the precondition of a task is not satisfied.

In the evaluation of the user interface we can examine the tasks the user tries to perform. If the user tries to perform a task which is not allowed in the current state of the application, we will record this as an error. The reasons for the errors differ. For example, it could be because the precondition of the task the user tried to perform was not satisfied or that the task was already disabled by another task.

2.4.4 WebRemUsine: a Tool Based on Task Models and Logs

RemUsine [Paternò and Ballardin 2000] is a task model-based tool for supporting remote usability evaluation: it gives such support when evaluators and users are widely separated in time and/or space [Paternò 1999].

RemUsine, however, suffers from the limitation of not being designed for or applicable to Web applications.

Thus WebRemUsine [Paternò and Paganelli 2001] is developing, in order to evaluate remotely Web sites. This approach is important because it proves that it is possible to use models to support analysis of real user behaviour whereas usually empirical testing and model-based evaluation have been considered separately.

Many approaches to the analysis and evaluation of Web applications focus on Web server logs. These approaches, however, suffer from many limitations: servers cannot collect data on some potentially crucial user interactions that occur on the client-side only. Furthermore, the validity of the data is highly suspect because of caching by proxy servers and browsers and dynamic IP addressing (see 3.3).

WebRemUsine is an extension of RemUsine, in which the basic concept was that if users perform a certain action (disabled), it means that they probably want to perform a related task. This is moot in Web interfaces: a link is always enabled. Therefore, it is more difficult to identify the user intentions.

WebRemUsine provides an automated support to evaluate Web sites. It provides a certain number of measurements which is useful to detect some critic aspects of evaluated Web site.

For example, the evaluator can see whether there are tasks particularly long and if some of them exist they can identify which pages create problems and whether they occur because of an excessive downloading time. The tool also displays the sequences of tasks performed and the associated sequences of Web pages accessed and provides lists of tasks performed (and pages accessed) ordered by time duration or frequencies of performance (in case of tasks) or number of accesses (in case of pages).

Patterns of tasks, actions and pages accessed are also automatically detected and displayed. The tool provides automatic support for analyzing whether the sequence of tasks performed is the optimal one for achieving the current goal and to count the

number of useless actions performed and pages accessed [Paternò and Paganelli 2001].

The tool is able to analyse not only single sessions but also groups of them, thus allowing the evaluator to understand whether a problem occurs often across many users or is only a problem limited to certain users and circumstances.

3
The Proposed Criteria

3.1 Introduction

In this chapter we discuss the proposed criteria to improve the usability of accessible Web sites. We suppose that a Web site is already accessible (i.e., it complies with accessibility guidelines), and we describe those aspects of Web pages that have an effect on navigability by users with special needs (i.e., blind or vision impaired users). As we already mentioned, our intention is to improve the navigability especially in a graphical environment. We propose a set of criteria that can be used to support both design and evaluation. We have focused on the Web page code, taking into account the HTML/XHTML language, JavaScript and Style Sheets (CSS). Accessibility guidelines advise not using JavaScripts because some browsers may not support them. Even though we share such principles, we aim at providing possible suggestions that also involve JavaScripts, because we do not wish to excessively limit the means at the disposal of developers. Moreover, this is especially important for sites already including scripts, which have to be modified in order to fix potential usability problems. Our goal is to create a semi-automatic environment supporting the designer rather than a completely automatic solution that would be too restrictive. In fact, developers may often decide not to repair Web sites because of the effort required, which depends on the number of changes necessary, and sometimes requires a general reorganization of the Web site.

In defining the proposed criteria, we aimed at identifying the main aspects that can cause usability problems in accessible Web sites. Then, for each criterion we provide more technical solutions to reach that goal, taking into account developers' choices in building the Web site (e.g., frames, JavaScripts, etc.). Thus, usability aspects are referred to in terms of associated criteria, while the technical solutions in terms of checkpoints.

3.2 How the Criteria are Organized

We consider the aspects that can be potential usability problems and the associated checkpoints. A *checkpoint* is a specific construct in a language for Web page development. While a certain criterion can involve only one checkpoint, others can address multiple checkpoints.

The classification of our criteria differs from that of W3C accessibility guidelines, which are arranged in 3 layers: guideline statements (i.e., general principles of accessibility); checkpoint list (i.e., how the guideline applies in typical content development scenarios); and a techniques section (i.e., implementations and examples for the checkpoints). Our approach aims at providing developers and evaluators with a more compact organization of criteria in order to simplify their use.

Moreover, as the three priority levels assigned to checkpoints based on their impact on accessibility, we group our proposed criteria according to the standard usability definition. More precisely, first of all we classify the proposed criteria according to effectiveness, efficiency and satisfaction principles; secondly we catalogue them depending on the type of impact on the user interface.

Classifying criteria according to the usability definition (ISO 9241) means that we identify those that are most important to reach the users' goals (effectiveness), those that allow reaching them more quickly (efficiency), and those that best satisfy users (satisfaction). We consider effectiveness criteria more important than those based on efficiency and satisfaction, because failure to satisfy such criteria could lead to users' not being able to accomplish their tasks. Thus, we consider effectiveness (level 1) more important than efficiency (level 2), and finally satisfaction (level 3).

The other parameter by which we classify criteria is the user interface aspect. A user interface is composed of two main components: the presentation, indicating how the user interface provides information to the user, and the dialogue, describing how the actions that users and system perform can be sequenced. According to this, we as annotate with an "a" the presentation criteria and those related to dialogue with a "b".

Concluding, criteria have been classified in the most objective and complete way possible: "effectiveness" criteria ensure the accomplishment of the task, "efficiency" ones shorten the time required to complete that task, and "satisfaction" criteria

provide Web pages with minor additional characteristics (addressed to improve the navigation) without the necessity of using specific commands.
In spite of the effort of giving an objective classification to the criteria, the inclusion of some of them in a group rather than in another can be moderately (but not significantly) influenced by personal interpretation.

3.3 Identification Phases of Criteria

To identify the criteria our study was conducted in various phases:

- *Empirical phase for identifying potential issues*. First of all we took into account those aspects (see below) that can be potential navigational problems for special users using a screen reader or magnifier. For this purpose we collected the feedback from a number of blind people (including the author of this Ph.D.thesis).
- *Testing phase for analysing potential solutions*. Then, according to those aspects, we analysed HTML specifications and JavaScripts, to determine possible solutions. Then, based on our hypothesis on possible criteria to solve the problems identified, some simple application examples were built and then tested by several blind users using a screen reader.
- *Systematic phase for defining the proposed method*. Lastly, the chosen criteria were classified in a systematic way, according to the usability definition, UI components and page elements involved.

The main potential navigational problems found when using a screen reader or magnifier are the following:

- *Lack of context* – Reading through the screen reader or a magnifier the user may lose the overall context of the current page and read only small portions of texts. For example, when skipping from link to link with the tab key, a blind user reads the link text on the braille display or hears it from the synthesizer, but not what is written before and after. E.g., links like ".pdf", "more details", "continue…", and so on, are depending on the sentences. A

similar effect occurs when using a magnifier: at any given moment, only a small portion of the enlarged text can be visualized on the screen.

- *Information overloading* – The static portions of the page (links, frames with banners, etc.) may overload the reading through a screen reader, because the user has to read every thing almost every time, thus slowing down the navigation. For instance, let us consider the case in which the user wishes to send an sms message. After having filled in the form and clicked on the "send" button, he wants to read the success or failure response. Often that output message is visualized in some position in the page, among other content, which is probably the same as in the preceding page. So, it may take a long time to find it, because the user has to read the information before the message, even if it is still the same as the previous page. Possible solutions to this issue include appropriate frame and link number usage, specific content marking, and a more organized hierarchical structure of pages.
- *Excessive sequencing in reading the information* – The command for navigating and reading can constrain the user to follow the page content sequentially. Thus, it is important to introduce mechanisms to ease the identification of precise parts in the page. An example of this is the result page generated by a search engine. Usually, in the top of such pages, there are several links, advertisements, the search fields and buttons, and so on, and then the search results begin. Furthermore, if the Web page contains more information blocks (e.g., paragraphs, short news, review lists, etc.), in order to read a specific block, the user has to read also the previous ones. An adequate partitioning or structuring of the content could allow special users to find the desired information more quickly.

As result of this process, 19 criteria have been identified and grouped into three sub-sets. To identify each criterion we use the format I.J.L where: I denotes the criterion kind, that is 1 for effectiveness, 2 for efficiency, or 3 for satisfaction; J is a progressive number to enumerate the criteria; L can be *a* (presentation) or *b* (dialogue) to indicate the aspect type on which the criterion acts.

3.4 The Proposed Criteria

In our work we determined 19 criteria to improve accessible Web site usability for users who access Web pages via a screen reader. As mentioned above, the criteria have been grouped into three sub-sets: effectiveness (5 criteria), efficiency (10 criteria) and satisfaction (4 criteria). For each criterion several checkpoints are proposed in order to indicate how it can be applied.

3.4.1 Effectiveness Criteria

We consider a criterion as belonging to the effectiveness sub-set if its application is important for reaching the user's goal. This means that if it is not adopted, users may not be able to accomplish their tasks because they will encounter difficulties in identifying important information.

Logical partition of interface elements *(1.1.b)*

This criterion aims at grouping information, links, fields and so on in logical categories, in order to allow users who read the page content through special devices to localize the essential parts in the page. In fact, as Web pages usually contain a lot of data, often users are not able to find a specific part or piece of information. Moreover, some screen readers allow skipping from section to section. This can be obtained by using markers, frames or headings to group texts, links, forms, and so on, according to a logical division, for instance, by defining frames such as 'index', 'search' and 'search results'. If the Web site does not use frames, it can be obtained by marking each part with hidden labels having similar strings to those above.

In order to apply this criterion, developers can use various methods:

- Headings: applying appropriate tags `<H1>…<H6>`;
- Hidden markers: putting non-displaying descriptive labels before significant "blocks";
- Tables: assigning useful values to `SUMMARY` attributes;
- Frames: assigning proper strings to `NAME` and `TITLE` attributes.

Proper link content *(1.2.a)*

The link labels are important for special users who use screen readers and keyboard commands (e.g., tab key) and, for this reason, may not be able to see the overall context (i.e., text written before and after a link). For this reason, it is important that links be clear and context independent. Thus, avoiding the use of such general texts as “more details”, “download”, “.pdf” is suggested. We must warn the designer that such texts can lower the site’s usability, because they are not very clear or they are too uninformative. So, designers can work on text, alternative text or `TITLE` attribute in order to improve this aspect:

- *Text links*: using more a meaningful description (i.e., text enclosed between `<A>` and `</A>`), and/or `TITLE` attribute;
- *Graphical links*: specifying a significant description for the `ALT` attribute value, referring to the meaning of the link rather than describing the image in itself;
- *Graphical and textual links*: when both graphical and textual links to the same resource exist, both `<IMG>` and a text string should be enclosed between the `<A>` and `</A>` tags, so that they are recognised as a single link by screen readers.

Messages and dynamic data management *(1.3.b)*

A significant difficulty that special users often encounter is represented by poorly designed system feedback. This may be confirmation or error messages about instructed operations or information requested from a database, which is not presented in a fashion interpreted well by screen readers, for example messages in the middle of the page or amongst a lot of information and links. In addition, often these dynamic messages are so “short” that they are not easy to focus on. This may actually lead to users’ wasting time or executing screen readers commands in order to reach these messages and have them read, if they manage at all. Developers have some possible ways to improve this situation:

- *Focusing on new messages*: including an anchor in correspondence to dynamic messages, and using a method to automatically position the browser on that anchor (e.g., by passing #anchor-name);

- *Using a dialogue window*: applying JavaScripts like ‘alert’, ‘confirm’ and ‘prompt’;
- *Opening a new window*: using JavaScripts, such as ‘window.open’ with larger messages.

Loading suitable style sheets *(1.4.a)*

Browsers can load specific sheets for different items using a particular tag (@media types). This style sheet feature allows specifying how a document is to be presented on different media, such as screen, paper, speech synthesizer, braille devices, etc. This enables the definition of a style sheet for each requirement, thus improving the layout of Web pages. Therefore, it is useful specifying that a style sheet, or a section of it, applies to certain special media types. In order to improve navigation of blind or visually impaired people, voice synthesizer, display and printer braille devices are taken into account by a specific section for each type of device.

Layout and terminological consistency *(1.5.a)*

Layout and terminological consistency is a usability feature that allows users to better understand the context and available functions. So, button features have a very strong impact on reducing situations that may be unclear for users, especially when their overview of the content is restricted. It is important that all the pages of the Web site adopt the same labels for buttons performing the same tasks (e.g., OK/Yes, quit/exit, next/forward), and all those pages should also have the same layout (e.g., dimension, form and colour). These two aspects have important consequences both for users of screen readers with speech synthesisers or braille displays, as well as for visually impaired users, who rely mainly upon dimension/colour references. Developers must carefully consider some aspects:

- *Button labels*: using the same labels (`VALUE` attribute) for buttons having same functions;
- *Button dimensions*: applying the same dimensions (`WIDTH` and `HEIGHT` attributes) for all buttons;

- *Link to home page*: applying a standard terminology like ‘home page’ or ‘home’ (possibly including it in the `TITLE` attribute).

3.4.2 Efficiency Criteria

An efficiency criterion is a rule that allows users to find the desired information more quickly. We consider this rule less important than effectiveness, because if such criterion is not satisfied, users can still perform their task, although it may take more time.

Number of links and frames *(2.1.b)*

It is important that a page does not contain too many links or frames because this makes it difficult for the user to skim through them all. In fact, the number of frames and links in a Web page affects navigation with special devices. Pages should have a balanced number of links: lots of links take a long time for readers to get through, too few links may imply too many levels in the Web pages’ hierarchy structure. In defining the range of values, the possible types of Web site and page should be taken into account. For instance, for Web sites relating to online books or guides, few frames are advisable (e.g., navigation bar, index and content), while for Web sites providing several services, more frames can be useful to distribute the various information and services. Similar rationale can be applied to the number of links: it is advisable to keep the number of links down in pages containing mostly reading material (paragraph, chapter or paper), whereas pages with indexes, search results or lists of services, more links may be necessary. Concerning Web sites structured by frames, in judging the number of links we suggest to consider each frame as a single page, that is to say, each should be considered separately. So, in a page having both an index and a content frame, the index can have numerous links, whereas their number in the content page should be limited.

- *Number of frames*: considering suggested ranges according to type of Web site;
- *Number of links*: considering advisable number of links on the basis of Web page categories.

Proper name for frames, tables and images *(2.2.b)*

It is important to insure that all frames, tables, and images have names and descriptions, which are appropriate and significant. E.g., frames with names such as "top frame", "middle frame", or "Left frame" are not very helpful to the user. On the other hand, names such as "index", "search" and "content" can make it easier for users to reach their goals, because they provide them with the option to skip some frames and cut down the amount of information to read. Similar considerations apply to text summaries of tables and alternative description for images and graphics. Descriptions such as 'this is a layout table' or 'logo.jpg (100kb)' are of no use to users, who must listen to the read out of extra text without the benefit of receiving significant information. To summarise, the relevant suggestions are:

- *Frames*: using meaningful values for the `NAME` and `TITLE` attributes;
- *Frames*: careful consideration of the `<TITLE>` tag of pages loaded into frames;
- *Tables*: providing a meaningful description through the `SUMMARY` attribute;
- *Image description*: using the `ALT` and `LONGDESC` attributes to indicate the type of picture in question (e.g., logo, decorative, link use, and so on).

Location of the navigation bar *(2.3.a)*

The so-called navigation links (i.e. the links appearing on each page and enabling users to reach the main parts of the site) represent a source of delay and inefficiency for the screen reader users. Since such links appear on each page (and often even twice), the users who are forced to read the contents in an almost sequential way (by means of a speech synthesizer or a braille display) are always compelled to skim them, before they can individuate the contents of the current page. Helping a more logical and organized development of this aspect can increase navigation efficiency for these users. Therefore, highlighting the navigation bar at the top and/or the bottom of the page, if any, can be useful to make it more understandable to users who are unable to see its visual features (e.g., horizontal or vertical position, colour or font types, etc.). Thus, other features can be used in order to localize the

navigation bar through a screen reader. We need, therefore, to select criteria that make navigation easier for people using a speech synthesizer (e.g., by graphic and/or text references or frames), and, on the other hand, for visually impaired people (e.g., by different colours and dimensions). The suggestions are:

- *Hidden markers*: putting specific hidden text in order to mark the start of the navigation bar;
- *Frames*: defining a specific frame dedicated to navigation links, such as 'navigation' or 'navigation bar';
- *I-frame*: defining a frame reserved for navigation bar, also when frames are not used for all Web site;
- *Menus and submenus*: using list tags (e.g., `<UL>`) in order to obtain a hierarchical structure easily identifiable with the screen reader; applying font and colour features in order to visually differentiate the multilevel menu elements;
- *Popup menu*: using appearing or disappearing blocks by means of `<DIV>` tags appropriately organized in precise sequential order, and by setting `VISIBILITY` properties.

Importance levels of elements *(2.4.b)*

In order to facilitate the navigation, especially with the keyboard, it is possible to assign different importance values to interaction elements, such as links, buttons, fields and so on. In this way, when users move through the 'Tab' key (element by element), they visit at first the most important, and later the less important, regardless of their location on the page.

This helps users to find more quickly the new elements. For instance, in the case of filling in a form, it would be useful to reach, through a tab, first the obligatory fields and then the optional ones. Links belonging to the navigation bar could have a lower level, whereas those of the current page have a higher level. Therefore, developers have to decide how to assign the various levels. Our suggestion is to use tabbing order position: using `TABINDEX` attribute to assign different importance levels to interaction elements (navigation goes from lowest tabindex value to the highest one).

Assignment of shortcuts *(2.5.b)*

It is advisable to assign hot keys to the most important buttons, links and fields, so that the user is able to reach them quickly through a simple key combination. This feature may be useful especially when users frequently visit the Web site, and they learn by heart the key combinations. In addition, a specific (possibly hidden) link to a page of combination list should be added to navigation bar. Note that we classify this aspect among the "efficiency" criteria because it enables reaching an object more quickly, but we could also include it in the "satisfaction" group, mainly for those users accustomed to using a lot of key combinations. The suggestions are:

- *Main links, buttons, and fields*: applying `ACCESSKEY` attribute with mnemonic shortcuts (usually first letters);
- *New interaction elements*: using same `ACCESSKEY` value (e.g., letter 'n') to skim through new elements.

Proper form layout *(2.6.a)*

In forms with several groups of data, we must properly lay out group titles and fields to achieve a major clearness. In fact, the way the elements are formatted can cause confusion with the screen reader. For example, in some cases the voice synthesizer or braille display could read before the 'checkbox', 'combobox' or 'field' item, and after its value, or vice versa. Often blank characters not appropriately used can cause difficulties in reading. Thus, a correct application of layout elements (e.g., simply by using the return tag in the proper place) is recommended.

- *Button labels*: giving a text to `VALUE` attribute;
- *Groups of control elements*: combining appropriately `<FIELDSET>`, `<LEGEND>` and `<BR>` tags;
- *Onchange event*: the use of `ONCHANGE EVENT` can create difficulties to users navigating especially through keyboard; so, using other elements (e.g. onclick event) to obtain a similar effect;
- *Matching labels and input elements*: using `<LABEL FOR>` tag in order to associate correctly each field label with its input element (this is useful especially when layout tables are used).

Specific sections *(2.7.b)*

In sites with frequent information update and/or new resources to download, we can help the user to find more rapidly the new elements by providing a specific section listing the new elements by date, sparing the user the trouble of going all over the site. Furthermore, a specific page listing all short keys associated to more important links of the Web site should be considered.

- *"Last update" section*: creating a specific Web page containing the list of recent updates, in inverse chronological order (so that users read first the recently updates), and for each of them putting the link to real Web page updated. Then, a specific link to the updates Web page would be added to navigation bar or to home page;
- *List of assigned keys*: defining a specific page containing shortcut list used in the whole Web site, and adding to navigation bar a (in case hidden) link to that page.

Indexing of contents *(2.8.b)*

In pages containing information of different kinds (e.g., paragraphs, news, etc.), we can help the user to find information more efficiently by indexing and pointing out different blocks, so sparing the need of a page skimming. First of all, it is useful to organize the information in logical way (i.e., by using headings, markers or frames), and then to apply a kind of index (i.e., local links), which points to every 'section'. The suggestions are:

- *Highlighting each block*: to this end it is possible to insert a link that is not displayed but can be detected and interpreted by the screen reader. The title block can become a sham link: this can be obtained through the insertion of an anchor at the beginning of the text following the title, and then the creation of a link with the title text, which points to that anchor (so, moving through tab key the screen reader can detect all titles);
- *Index at the top of page*: it can be done by putting an anchor in correspondence of each block, and then adding a list of links at the beginning of the page (in case enclosing links in a specific i-frame);

- *Index frame*: if the Web site uses frames, a specific frame containing the link to every section can be inserted.

Navigation links *(2.9.b)*

In order to reach more easily some location of the page (or of the site) we can insert local navigation links, referring to bookmarks in the ambit of the page (e.g., 'skip to content', 'go to top', 'go to navigation bar', etc.). These types of links are useful for the navigation with the screen reader, but, especially, with browsers that have not specific movement commands. The suggestions are:

- *Skip to content*: including an anchor in correspondence to the new page content (i.e., after the navigation bar or advertisement data), and then adding a local link to that bookmark at the top of the page;
- *Go to the top of the page*: inserting a link that allows going back to the beginning of the page, the bottom of the page or after each section (if sections are present);
- *Go to specific section*: relating to some special sections (e.g., navigation bar, search field, index, and so on), defining specific links on the bottom or on the top of the page.

Identifying the main page content *(2.10.b)*

One of the most important consequences of blind users not being able to view a page overview is that they are unable to identify new content on a page. In fact, Web pages often have diverse information, objects, elements, and so on, which are static and only a part changes when a new page is loaded. The modified part can be called 'new page content' or with a similar name. So, blind users may not be able to identify the 'new page content' because the screen reader deals with the page sequentially, and hence too much data (sometimes poorly organized) may create confusion and render the content unclear. For this reason, we suggest some possible strategies to use in order to address such issues.

- *Positioning at page loading*: the focus is automatically moved to the new content when the page is loaded;

- *Using heading levels*: the first line of new page content (usually its title) should be the first text to which a heading level `<HI>` tag has been applied;
- *Using frames*: when the Web site has a frame structure, a specific frame named 'content' should be used.

3.4.3 Satisfaction Criteria

Satisfaction criteria help to produce a Web site that is more pleasant and easier to navigate. Often the application of satisfaction criteria provides useful information without the need of any special command. For example, applying a short sound to the page loading means to provide an additional piece of information that allows users to know, exactly and in real time, when the page is completely loaded. When no sound is applied, users can still accomplish their task, but doing that will probably require more effort.

Addition of short sounds *(3.1.b)*

Associating a short sound to different elements and different kinds of multimedia environment can make the user more "satisfied", e.g., providing each page with a short sound indicating when the loading of the page is completed, spares the user the need to repeatedly check the state bar. Associating different sounds to different links makes it easier to identify the link type during the skimming.

Colour of text and background *(3.2.a)*

This aspect can make navigation easier for visually impaired people who, with a particular type of contrast, may feel less tired by navigation. It is therefore advised to avoid colour combinations giving a poor contrast. Furthermore, changing colours in correspondence to some events, or particular areas, can be a way to get attention. The suggestions are:

- *By passing mouse*: using css specification (or JavaScripts) to change colour when the mouse pointer goes on elements (links or buttons);
- *Getting focus*: using css specification (or JavaScripts) to change colour when elements get the focus;

- *Specific areas*: applying different background and text colours for particular areas (e.g., navigation bar or index frames, search or result area, and so on).

Magnifying at passing by mouse *(3.3.a)*

The use of this feature can help people with a good visual residue to better focus on the pointed object. The idea is to enlarge particular elements such as images, navigation links and buttons, not all text. We suggest magnifying the elements only when the mouse pointer goes over them, because this feature can be useful especially for users who have a visual residue and therefore can use the mouse.

- *Images*: using JavaScripts to associate an "enlarge" function when the picture is pointed (onMouseOver) and a "reduce" function when the pointer goes out (onMouseOut);
- *Links and buttons*: using css instructions (or JavaScripts) to magnify the elements when the mouse points them.

Page information *(3.4.b)*

Using adequately the content of the title and the last line of the page can be a useful care to improve the reading by screen reader. In fact, defining the beginning and ending of the page by adding useful information can make the navigation more pleasant. For instance, the title of the page, which is read as first line, could contain not only the Web page title, but also additional indication like the current path. Also the last line of the page could contain information so that users are able to understand that is the last line by sequential reading.

- *Page pathway*: in addition to page title, the tag `<TITLE>` could contain information about page pathway (e.g., home::news::monthly bulletin); however an hidden label could be applied too.
- *Repeated end page line*: using a same text for the last line of all the pages help users to better recognise when the reading (by arrow keys) is at the end of the page. For example, each page could have the date of last update as last line.

3.5 Impact of the Criteria on the Web Pages Code

Different criteria have different impact on the Web pages code. Some of them require changing specific elements such as the links or the frames. Others have an impact on the pages structure, and others have an impact on the entire site. Table 3.1 summarises the impact of each criteria.

Objects	Criteria
Links	1.2.a, 2.1.b, 2.4.b, 2.5.b, 2.7.b, 2.8.b, 2.9.b, 3.1.b, 3.2.a, 3.3.a
Frames	1.1.b, 2.1.b, 2.2.b, 2.3.a, 2.8.b, 2.10.b
Forms, buttons and fields	1.5.a, 2.4.b, 2.5.b, 2.6.a, 2.8.b, 3.2.a
Pages	1.1.b, 1.2.a, 1.5.b, 2.1.b, 2.3.a, 2.4.b, 2.5.b, 2.8.b, 2.9.b, 2.10.b, 3.1.a, 3.2.b
Sites	1.1.b, 1.3.a, 1.4.a, 1.5.b, 2.1.b, 2.7.b, 2.8.b, 3.1.b
Java scripts	1.3.b, 2.3.a, 2.8.b, 2.10.b, 3.1.b, 3.3.a

Table 3.1 - List of criteria classified according to the objects they affect.

3.6 GUI and Web UI: shared criteria

Since our criteria identify general principles for improving the interface use through more simple navigation, they can be also referred to GUIs. More precisely, around half of criteria can be adapted to GUIs (e.g. the graphical links of a Web page become graphical buttons in a GUI). Adaptable criteria are listed below:

1.1.b (Logical partition of interface elements);
1.2.a (Proper link text);
1.3.b (Messages and dynamic data management);
1.5.a (Layout and terminological consistency);
2.5.b (Assignment of shortcuts);
2.6.a (Proper form layout);
2.9.b. (Navigation links);

3.1.b (Addition of short sounds);

3.2.a (Colour of text and background),

3.3.a (Magnifying at passing by mouse).

4
Web Analysis of Web Sites through the Proposed Criteria: Some Examples

4.1 Introduction

In this section we show some examples of usability issues in accessible Web sites detected when users interact through the screen reader Jaws. The examples are taken from three Web sites, evaluated by us through the set of proposed criteria. In particular, firstly, we considered the Web site of the Florence City Council (http://www.comune.firenze.it), which provides information and services to citizens, paying particular attention to the social security Department (i.e., 'The services' section); secondly, we tested a Web site on public access (http://www.pubbliaccesso.it), which aims at providing both developers and users with information regarding accessibility topics. Lastly, the third Web site we analysed is about the Environment Department (http://www.minambiente.it/). More precisely, in this case, an accessible prototype has been developed for supporting the search of information concerning public and private institution. After that portion has been made accessible according to the WAI W3C guidelines, we applied our criteria.

These studies have been done to assess how developers deal with usability when developing accessible Web sites and how the criteria adopted are suitable to be applied to existing Web sites. For each example, we show how the page is visualized on the screen, how that page is read by the screen reader before and after our suggested changes, and the code fragments involved. In the table showing how the Web page is interpreted by the screen reader, the bold text with 1 indicates changes, while the italic text with 2 refers to the parts that are read by the synthesizer, but not visualized in the Web page.

4.2 Proper Text: Frames, Links and Tables

An important issue of usability for special users who read the Web pages through a screen reader is the text associated to links, name attribute of frames, summary attribute of tables, and so on. The reason is that, while exploring a page with a synthesizer or braille display, the user may not have a general view of the content. Therefore, if the name of frames or the text of links are significant, users can better orient themselves.

4.2.1 Names of Frames

Unfortunately, often the names of frames are not very meaningful, such as SX, DX, CENTRAL, MAIN, etc.. This is particularly important when users read through a screen reader. For a better usability, the name of a frame is important for an easier understanding of both the page structure and the frame content. Usually this is not important, because the names of frames are not displayed in the page visualized in the browser and so developers do not pay attention to them.

If we consider the Web page in Figure 4.1, where services offered by the City Council of Florence are listed, the structure of that page is composed of several frames which are not entirely visible.

In that page information and links to skip to other sections, are logically grouped in more nested frames. The picture shows how the page content is rendered on the screen. A screen reader reads that content in different ways. In order to try to understand how the synthesizer or braille display considers the page, we provide a fragment of the Web page text read by the synthesizer (see Figure 4.2). As we can observe, the screen reader identifies clearly the beginning and the end of every frame, thus the use of appropriate names is important.

Figure 4.1 - Page of the Web site of Florence's Council where services offered to citizens are listed (http://www.comune.fi.it/inglese/)

main *frame*[2]	***main*** *frame*[2]
top[1] *frame*[2]	***navigation bar***[1] *frame*[2]
Florence's council Web site	Florence's council Web site
link[2] italian version	*link*[2] italian version
link[2] the mayor	*link*[2] the mayor
...	...
link[2] home	*link*[2] home
top[1] *frame end*[2]	***navigation bar***[1] *frame end*[2]
left[1] *frame*[2]	***navigation sub-bar***[1] *frame*[2]
link[2] documents found	*link*[2] documents found
link[2] immigrants	*link*[2] immigrants
...	...
left[1] *frame end*[2]	***navigation sub-bar***[1] *frame end*[2]
central[1] *frame*[2]	***content***[1] *frame*[2]
...	...
central[1] *frame end*[2]	***content***[1] *frame end*[2]

bottom[1] *frame*[2] search in our server *edit*[2] *button*[2] search ***bottom***[1] *frame end*[2] ***main*** *frame end*[2]	***search bar***[1] *frame*[2] search in our server *edit*[2] *button*[2] search ***search bar***[1] *frame end*[2] ***main*** *frame end*[2]

Figure 4.2 - Fragment of Web page content read by the screen reader: The left part shows the reading of the page before changes, and the right part after the changes

As we can see in the figure, the screen reader distinguishes the frames of the page and also their beginning and end. Therefore, it is useful to have significant names. Note that the "main" frame is no very important, because it is actually used to contain the others.

A similar situation occurs also in the part related to 'ricerca enti' of the Environment Department Web site (See Figure 4.12). In that section there are five frames that originally had non very appropriate names (see Figure 4.3).
The user can read the content of the page in a sequential way, or skip directly to a certain frame. In fact, some screen readers (like Jaws) have special commands to list the available frames, and to activate one of them. The user can choose one frame to access it. For instance, selecting "search results", users access the corresponding frame so that they are able to find the search results more quickly.

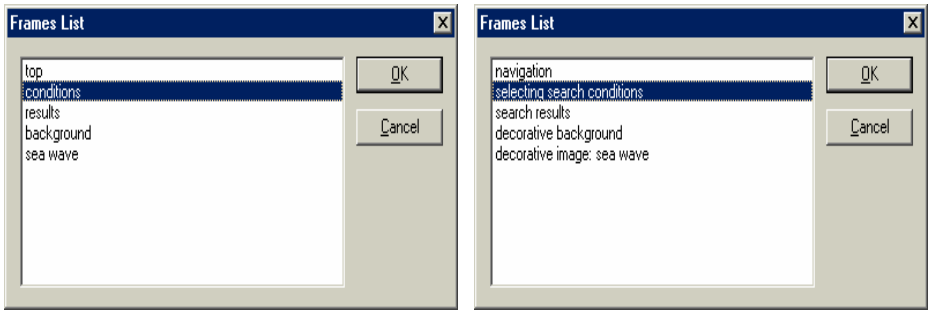

Figure 4.3 - Frame list (produced by a specific command of the screen reader) related to the Web site of the Environment Department (http://www.minambiente.it/).

In order to show the code related to the frame names, we consider a fragment of the page of the Florence's site taken into account (Figure 4.4). In particular, we consider the FRAME tags: first of all, the presence of both NAME and TITLE attributes, then their content.

```
<HTML>…<frameset cols="*,750,*">
<frame name="main"[1] src="main3ing1.htm" title="main"[1] frameborder="0">
<noframes>…</noframes> </frameset> </HTML>
```

```
<HTML>…<frameset rows="84,*">
<frame name="navigation bar" title="navigation bar"[1] frameborder="0"
scrolling="NO" noresize src="top5ing.htm"> <frameset rows="*,29" > <frame
name="content" title="content"[1] src="home3ing.htm" scrolling="yes"
frameborder="0">
    <frame name="search bar" title="search bar"[1] scrolling="NO" noresize
src="bottom2ing.htm" frameborder="0"><noframes>…</noframes> </frameset>
</frameset> </HTML>
```

Figure 4.4 – Code of Web page at http://www.comune.fi.it/inglese/ (top), and at http://www.comune.fi.it/inglese/main3ing1.htm **(**bottom)

Furthermore, it is important that the strings associated to name and title attributes be the same. In fact, if the values are different, the screen reader could behave in different ways, depending on the type of interaction technique used (i.e., it reads the name if the user interacts through the Ctrl-tab browser command, and the title if the user uses the screen reader commands).

In order to evaluate the frame name we could use a dictionary in which the evaluator can store unsuitable terms, such as SX, DX, CENTRAL, MAIN, etc.. Using an external dictionary we can have two advantages: first, the evaluator can add new terms customizing the evaluation, second, there may exist different dictionaries for different languages.

4.2.2 Link Content

A similar problem to frame name issue occurs with links: links like "CLICK HERE", ".PDF", or "GO TO PARAGRAPH" are not very useful. Often the link text is referred to a

specific context, so that if we consider links separately, we might not be able to understand the related content.

A user who cannot see the screen, often uses the Tab key to search in the page the link wanted without reading the whole content. Another way to select a specific link is to use a particular command of the screen reader to open the link list. In both cases, the user reads only the text of the links, so a significant content is important.

In Figure 4.5 an instance of this issue is showed. The picture contains a list of links referring to the descriptions about the WAI checkpoints. As we can note, the link content is not very explanatory: the link text is only the checkpoint number. Hence users are not able to immediately get the actual checkpoint name pointed from that link; probably they will not activate a link from the special list produced by the screen reader, but they will have to move through TAB key and explore the content through arrow keys.

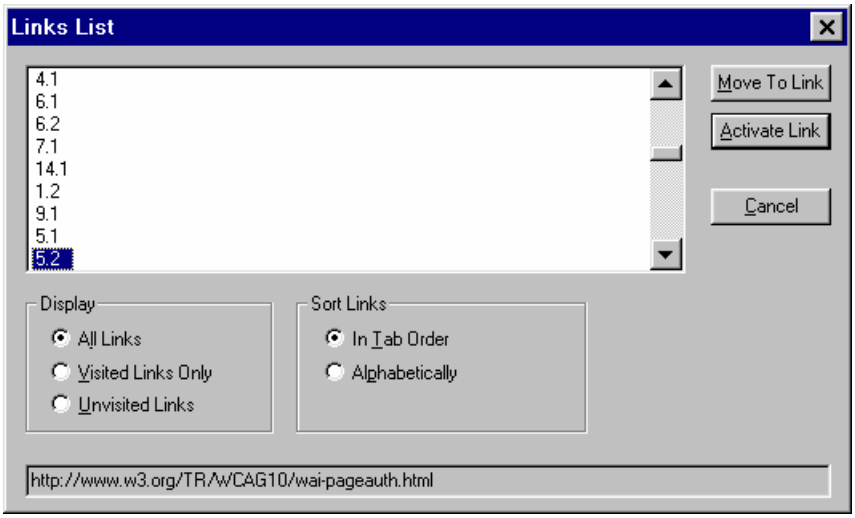

Figure 4.5 - Portion of the link list produced by a special command of the screen reader about WAI checkpoint list at http://www.w3.org/TR/WCAG10/checkpoint-list.HTML

So, to improve link usability we can suggest some solutions. First, for graphical links the alt attribute is handled by putting a significant description referring to the meaning of the link rather than describing the image in itself. Second, for textual links we can change the entire text, or we can use the TITLE attribute. This second alternative may be used if developers do not want to modify the writings visualized

on the screen. In fact, the text associated to alt or title attributes is read from the screen reader, and it is visualized in the status bar when mouse is passed over links.
A more particular case occurs when both graphical and textual links point to same resource. Let us consider some examples.

Graphical links

In the two pictures of Figure 4.6 there is the list of links associated to each topic belonging to an online guide organized in more paragraphs. There is one graphical link before every paragraph item. Since the links are images, the use of ALT is necessary. In the original version all links have the same text: 'go to paragraph'. In the fixed version, the ALT attributes contain also the name of paragraph (e.g., 'go to paragraph presentazione').

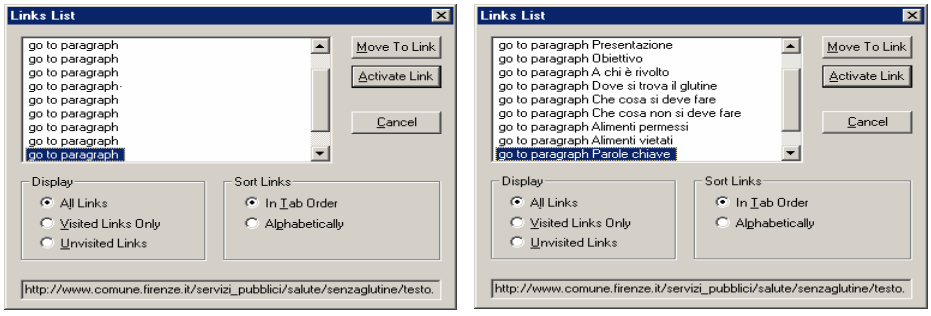

Figure 4.6 – List of links (produced by a specific screen reader command) available at http://www.comune.firenze.it/servizi_pubblici/salute/senzaglutine/.
On the left the original version, on the right the fixed version.

Reading the Web page by a screen reader, the original links are like "*link*[2] *go to paragraph*[1]", while those modified are like "*link*[2] *go to paragraph presentazione*[1]". So, in the original page there are too many links with similar text, such as 'go to paragraph'. In presence of many links of this kind, if users skip from link to link using the tab key or a special command of the screen reader that provides a link list, they read similar texts without knowing the context to which they refer. Therefore, in order to know which is the appropriate link, a user has to explore the page reading line by line or almost. The same problem is encountered also from low vision users who can read little portions of the screen through a magnifier program.

In order to solve this problem, we should modify the text of the links by adding the name of the chapter to which the link points (e.g., “presentazione”, “obiettivi”, etc.) to ‘go to paragraph’. This effect can be obtained changing the link text, or using the ALT and TITLE attributes. This second possibility, among other things, allows two rendering: one visual and one for the screen reader. In our example, the links are images, therefore we have to modify the alt and title attribute text: on the screen the links are still images, while the screen reader reads the text ‘go to paragraph presentazione’, etc.. In addition, for each link the evaluation has to check whether the text is non-recommended or not. Similarly to frames, we suppose that external dictionaries in which non-recommended terms are listed exist. The evaluation criterion extracts text of links - including ALT and TITLE value - and checks they do not belong to that terms.

```
… <A target=content
href="http://www.comune.firenze.it/…/testo.htm#presentazione"> <IMG
height=19 alt="go to paragraph Presentazione"[1]
src="sommario_file/spunta2.gif" width=20 border=0></A> Presentazione <BR>
<A target=content
href="http://www.comune.firenze.it/…/testo.htm#obiettivo"> <IMG height=19
alt="go to paragraph Obiettivo"[1] src="sommario_file/spunta2.gif" width=20
border=0></A> Obiettivo <br>
...
```

Figure 4.7 – HTML code of online guide Web page; the piece of code corresponding to contents evaluated and already changed is in bold

Textual links

Another example of a different content between link text and its TITLE attribute is shown in the following figure. Many links of that page have a short text, and a longer content is assigned to TITLE attributes. Figure 4.9 reports how the screen reader interprets those links. From this example we can observe that a considerable difference between link text and title attribute may exist.

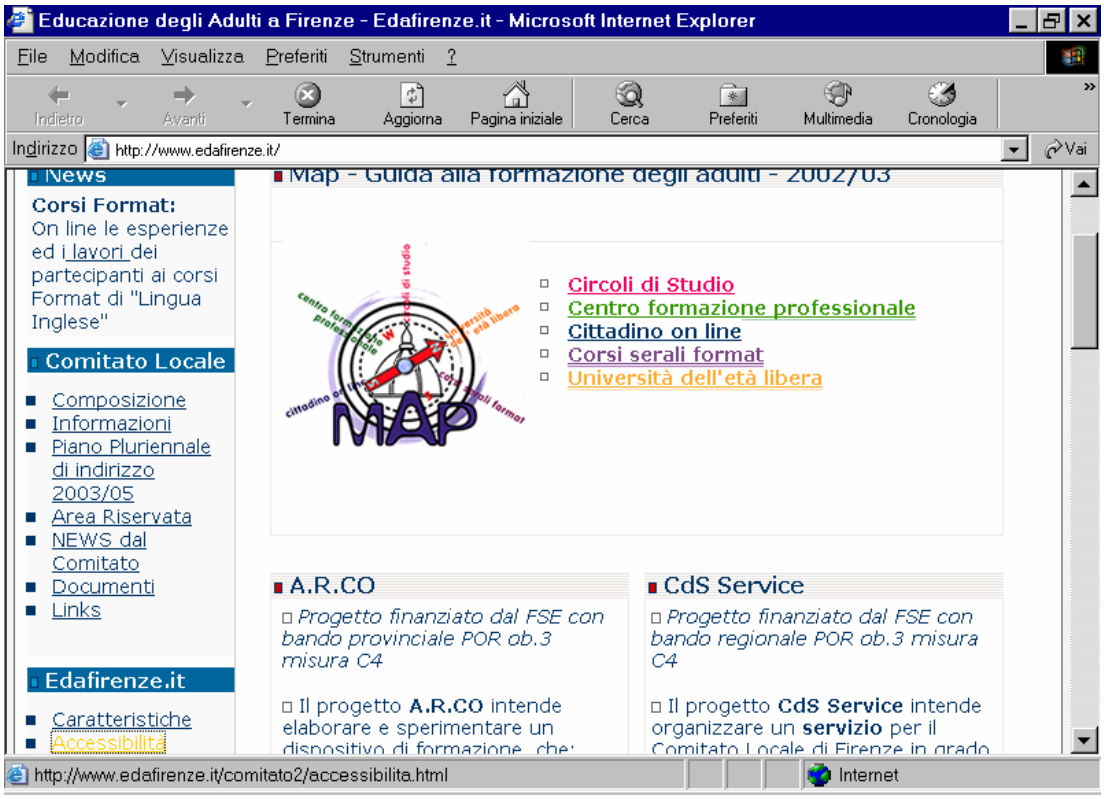

Figure 4.8 - Page of Florence's educational Web site available at http://www.edafirenze.it/

... Comitato Locale *List of 7 items*[2] • *Link*[2] ***Composizione1*** alt+s • *Link*[2] ***Informazioni1*** ... • *Link*[2] ***Documenti1*** ... *list end*[2] ... Edafirenze.it *List of 5 items*[2] • *Link*[2] ***Caratteristiche1***	... Comitato Locale *List of 7 items*[2] • *Link*[2] ***Composizione del Comitato Locale di Firenze1*** alt+s • *Link*[2] Informazioni sul ***Comitato Locale1*** ... • *Link*[2] ***Documenti relativi all'Eda, Educazione degli Adulti1*** ... *list end*[2] ... Edafirenze.it *List of 5 items*[2] • *Link*[2] ***Caratteristiche del nuovo sito1***

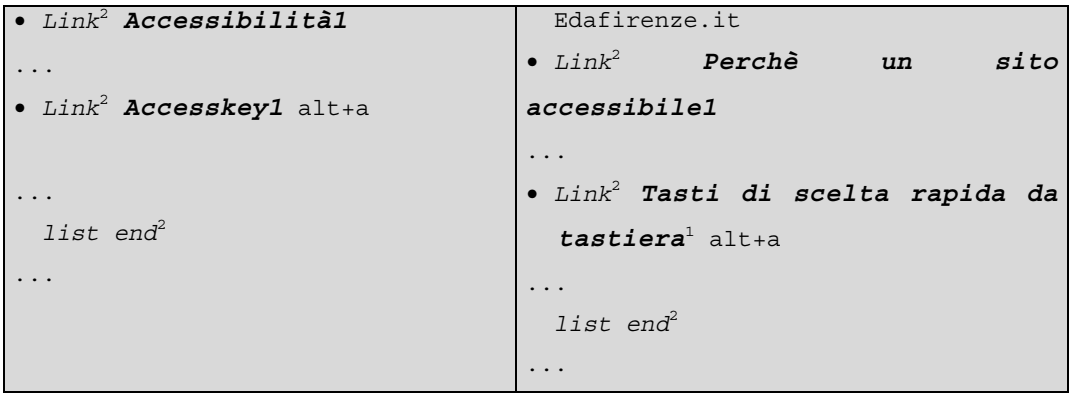

Figure 4.9 – Fragment of page read by the screen reader where link texts (at left) and link title attributes (at right) are shown

Graphical and textual links

Let us consider the navigation bar shown in Figure 4.10. The navigation bar is composed of four links pointing to the next and previous page; two links are graphical and two textual. So there are four links, but the resources referred to are only two (see the HREF values in the picture). Moreover the screen reader interprets them as four different links (see Figure 4.11). Thus users could have some difficulties in understanding that the first two links point to "previous page", and the other two to "next page". This happens because the links are treated as four distinct links: two <A> tags with graphical content, and two with textual content. Since the content of ALT attribute differs from that textual one, despite the resource linked is the same, the caused effect might create confusion.

A suggested solution to this case is enclosing both graphical and textual content to the same resource in an only <A> tag. In this way the screen reader recognises two links rather than four. Let us see Figure 4.11 for the fragments of reading and requested code.

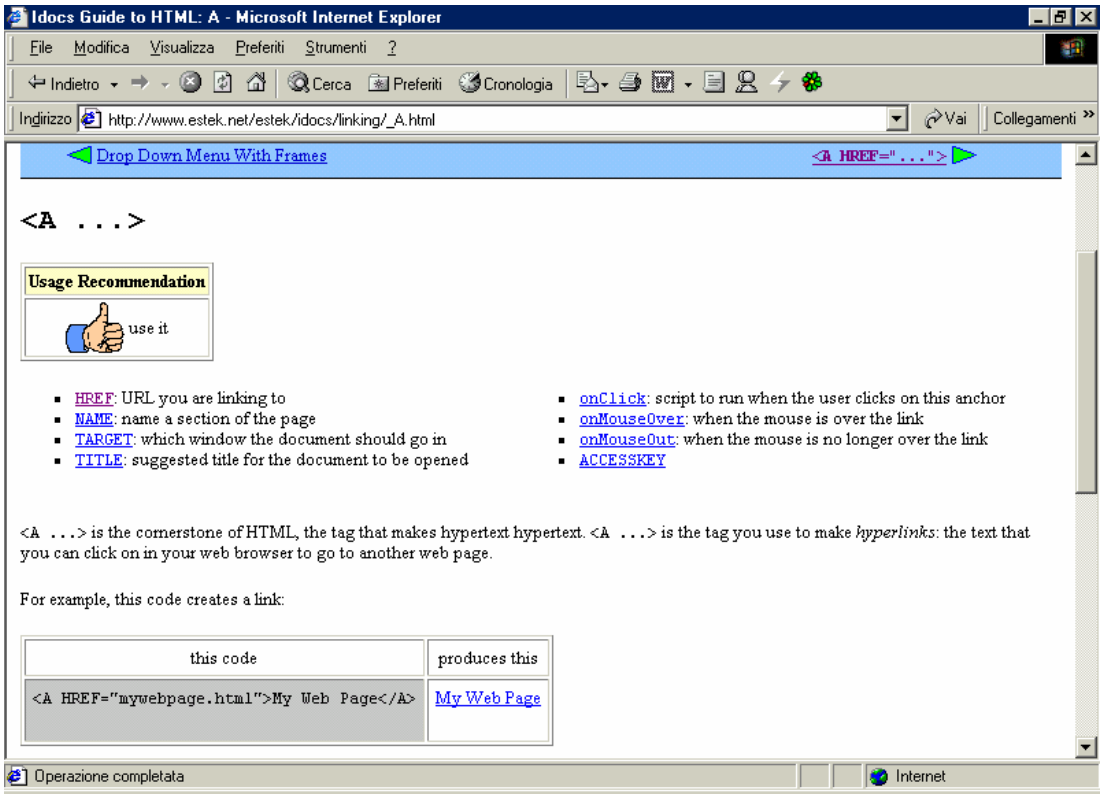

Figure 4.10 - A page of the HTML online guide at http://www.estek.net/estek/idocs/linking/_A.HTML

... Link² ***previous page1*** Link² ***Drop Down Menu With Frames1*** Link² ***<A HREF="...">1*** Link² ***next page1***	Link² ***previous page: Drop Down Menu With Frames1*** Link² ***next page: <A HREF="...">1***
<A HREF="linking_famsupp_137.HTML"> <IMG SRC="../arrow_prev.gif" ***ALT="previous page"1***> </A> <A HREF="linking_famsupp_137.HTML"> ***Drop Down Menu With Frames1*** </A> <A HREF="_A_HREF.HTML"> ***<A***	<A HREF="linking_famsupp_137.HTML"> <IMG SRC="../arrow_prev.gif" ***ALT="previous page:"1***> ***Drop Down Menu With Frames1*** </A> <A HREF="_A_HREF.HTML"> <IMG SRC="../arrow_forward.gif" ***ALT="next page:"1*** ***<A HREF="...">1*** </A>

HREF="...">1 `</A>` `<A HREF="_A_HREF.HTML"> <IMG SRC="../arrow_forward.gif"` ***ALT="next page"1***`> </A>`	

Figure 4.11 – At top, the screen reading fragment before the changes (left) and after (right); at bottom, the necessary HTML code in original version (left) and modified version (right).

4.2.3 Tables: Proper Summaries

Using tables to format the content in the pages is usually not suggested. However, developers often prefer to use this technique because it allows them to obtain a more precise content layout. Therefore, in this case our criteria suggest to define useful strings for summary attribute. Let's consider now the example shown in Figure 4.12 in order to give an example about this aspect.

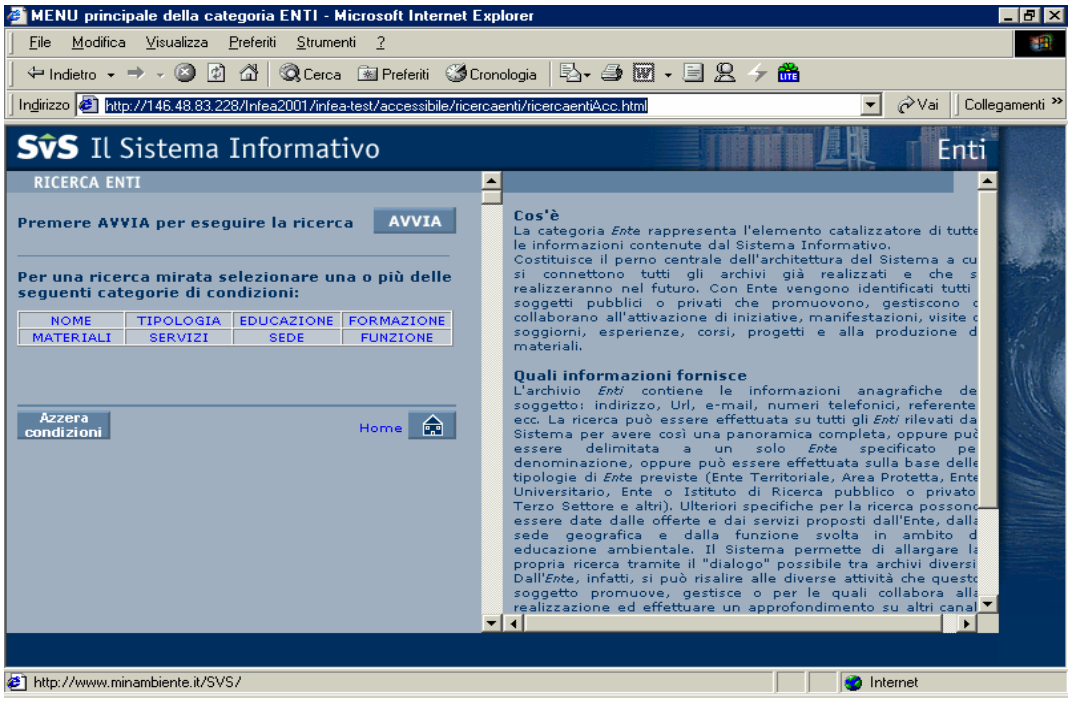

Figure 4.12 - Web page containing organization search criteria available at http://146.48.83.228/Infea2001/infea-test/accessibile/ricercaenti/ricercaentiAcc.HTML

That page, structured by using frames (see 4.2.1), has a specific frame named ‘Selection search conditions’ where more tables are used in order to format the search criteria. In particular, in addition to the ‘search parameters’ table (see Figure 4.12) where there are local links to move the focus to a specific criterion, a layout table for each search criterion is used. Actually, several nested tables have been used to visualize checkboxes, combo boxes, radio buttons and text fields that are shown when the associated link is selected. Since too many nested tables have been used, it was not easy to modify appropriately the corresponding code. Figure 4.13 shows how the screen reader interprets that content, before and after our suggestions. As other cases, the italic text with 1 indicates changes, while the italic text with 2 refers to the parts that are read by the synthesizer, but not visualized in the Web page.

... 1.1.1.Table with 4 columns and 2 rows[2] *Summary:*[2] ***Categories of selection conditions1*** *Link*[2] Name *Link*[2] Tipology ... *table end*[2] Institution name *Edit*[2] *Graphic*[2] Ending of Name category *Link*[2] Back to selection *Table with 2 columns and 40 rows*[2] Radio button No conditions Checkbox Central public institution ... *Graphic*[2] Ending typology category *Link*[2] Back to selection *table end*[2] Title *Edit*[2]	... 1.1.2.Table with 4 columns and 2 rows[2] *Summary:*[2] ***Categories of selection conditions1*** *Link*[2] Name *Link*[2] Tipology ... *table end*[2] *Table with 2 cols and 4 rows*[2] ***Summary: by name1*** Institution name *Edit*[2] *Graphic*[2] Ending of Name category *Link*[2] Back to selection *Table end*[2] *Table with 2 columns and 40 rows*[2] ***Summary: By typology1*** Radio button No conditions Checkbox Central public institution ...

Table with 2 columns and 12 rows[2] Check box Experiences ... *Graphic*[2] Ending of education category *Link*[2] Back to selection *table end*[2] ...	*Table end*[2] *Graphic*[2] Ending typology category *Link*[2] Back to selection *table end*[2] *Table with 2 cols and 4 rows*[2] ***Summary: By Education1*** Title *Edit*[2] Check box Experiences ... *Graphic*[2] Ending of education category *Link*[2] Back to selection *table end*[2] ...

Figure 4.13 - Fragment of Web page content read by the screen reader: The left part shows the reading of the page before changes, and the right part after the changes

Practically, by simply pressing the letter 't', the screen reader skips table by table, reading the type of selection category each time (e.g., by name, by typology, by education, and so on). In order to obtain such an effect, the page code should contain <TABLE> tag with appropriate SUMMARY attribute for every category of selection.

```
…          <table width="328" border="1" cellspacing="0" cellpadding="0"
bgcolour="#C0C8D8"
      summary="Selection conditions"1><!--#7F99B2-->
<!--Begin of selection filters table-->
…
<table    width="350"    border="0"    cellspacing="0"    cellpadding="8"
summary="by name"1>
…
```

Figure 4.14 – HTML code fragment related to the page in the frame "Selection conditions"

4.3 Navigation Bar and Menus

Pointing out the navigation bar, if there is one, at the top and/or the bottom of the page, can be useful for users who are not able to see its visual features (e.g., horizontal or vertical position, colour or font types, etc.). Thus other features can be used in order to localize the navigation bar by a screen reader: we could mark the beginning of the static links, insert a particular link to skip the bar, and so on. Specific labels and strings can be used in order to identify the navigation bar. A particular application is a pop-up menu having disappearing submenus. In the next three paragraphs three examples of this issue are described.

4.3.1 Navigation Bar

Let's consider the page in Figure 4.15 where a specific section of the PubbliAccesso Web site is visualized. Two navigation bars are used: one at the beginning, and one at the end of the page. In addition, a hierarchical structure menu items is available on the left (see next paragraph).

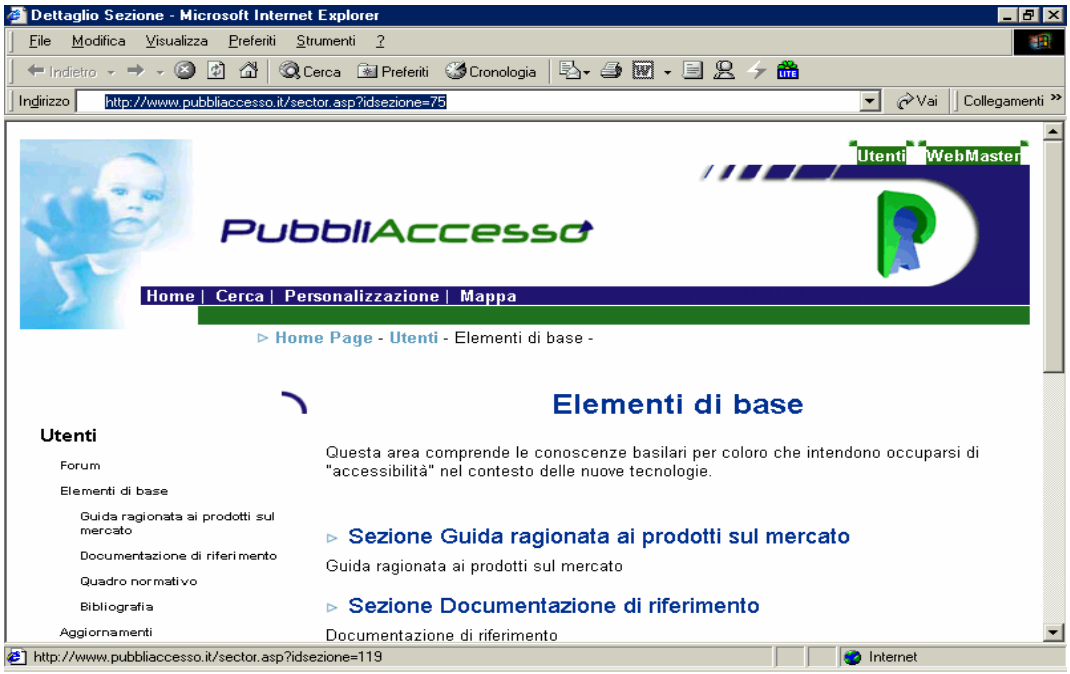

Figure 4.15 - 'Elementi di base' section of the PubbliAccesso Web site available at http://www.pubbliaccesso.it/sector.asp?idsezione=75

To format the navigation bars, layout tables have been used. In this case we suggest the application of an appropriate summary attribute to those tables, such as ‘Navigation bar’ instead of the misleading ‘Right part of the head’. In this way users are able to immediately understand the meaning of the table. However, we suggest inserting a link like ‘skip to content’ in order to avoid the navigation bar content (which is always the same) and go directly to the new content. In the next two figures, we can see how users read the page content (i.e., how they can individuate the navigation bar) and then which is the code involved.

Table with 2 columns and 5 rows[2] *Summary*[2]*:* ***Right part of the head1*** *Link*[2] Utenti *Link*[2] WebMaster *Link*[2] Cerca | *Link*[2] Personalizzazione | *Link*[2] Mappa *table end*[2] … Page content …
Link[2] ***Skip to content1*** *Table with 2 columns and 5 rows*[2] *Summary*[2]*:* ***Navigation bar1*** *Link*[2] Utenti *Link*[2] WebMaster *Link*[2] Cerca | *Link*[2] Personalizzazione | *Link*[2] Mappa *table end*[2]
Link[2] ***Skip to content***[1] ***Graphic***[2] ***Navigation bar:***[1] *Link*[2] Utenti *Link*[2] WebMaster

Link[2] Cerca |

Link[2] Personalizzazione |

Link[2] Mappa

Graphic*[2] *Page content1

Figure 4.16 – How to identify the navigation bar by screen reader: on the top, a reading fragment before the changes; in the middle, a change obtained by using a table; on the bottom, a second method using hidden labels

So, we provided two ways to localize navigation bar: the first consists of the table use with a specific summary attribute (i.e., navigation bar); the second one applies hidden labels to mark the beginning of navigation bar and page content. Both ways provide a specific link to skip to content that, if chosen, frees the user from having to read the navigation bar links.

```
<TABLE style="WIDTH: 100%" cellSpacing=0 cellPadding=0 border=0
summary="Right part of head"1>
<TBODY>
...
<TD class="" style="BACKGROUND-COLOUR: #297618">
<A class=menu_bar title="Go to user section"
href="http://www.pubbliaccesso.it/sector.asp?idsezione=119">Utenti</A>
</TD>
<TD class="" style="BACKGROUND-COLOUR: #297618">
<A class=menu_bar title="Go to Webmaster section"
href="http://www.pubbliaccesso.it/sector.asp?idsezione=120">WebMaster</A>
...
</table>
```

```
<a href="#content" title="skip to content"> Skip to content </a>1
<TABLE style="WIDTH: 100%" cellSpacing=0 cellPadding=0 border=0
summary="Navigation bar"1>
<TBODY>
...
<TD class="" style="BACKGROUND-COLOUR: #297618">
<A class=menu_bar title="Go to user section"
```

```
href="http://www.pubbliaccesso.it/sector.asp?idsezione=119">Utenti</A>
</TD>
<TD class="" style="BACKGROUND-COLOUR: #297618">
<A                          class=menu_bar title="Go to Webmaster section"
href="http://www.pubbliaccesso.it/sector.asp?idsezione=120">WebMaster</A>
...
</table>
...
<a name="content"> </a>1
```

```
<a href="#content" title="skip to content"> Skip to content </a>1
<img src="" alt="navigation bar:" width=1 height=1>1
...
<A class=menu_bar title="Go to user section"
href="http://www.pubbliaccesso.it/sector.asp?idsezione=119">Utenti</A>
<A                          class=menu_bar title="Go to Webmaster section"
href="http://www.pubbliaccesso.it/sector.asp?idsezione=120">WebMaster</A>
...
<img src="" alt="Page content" width=1 height=1>1
<a name="content"> </a>1
...
```

Figure 4.17 - HTML code of original page and of two solutions suggested

4.3.2 Menus and Submenus

When in a Web page there are several links organized into various levels, lists can be used. When we build a menu composed of sub-items, it is important to distinguish the main elements and the sub-items. So, an unordered list (i.e., <UL>) can solve this issue.

In this paragraph we show an example. In the PubbliAccesso Web site, a multilevel submenus is positioned to the left side of the page.

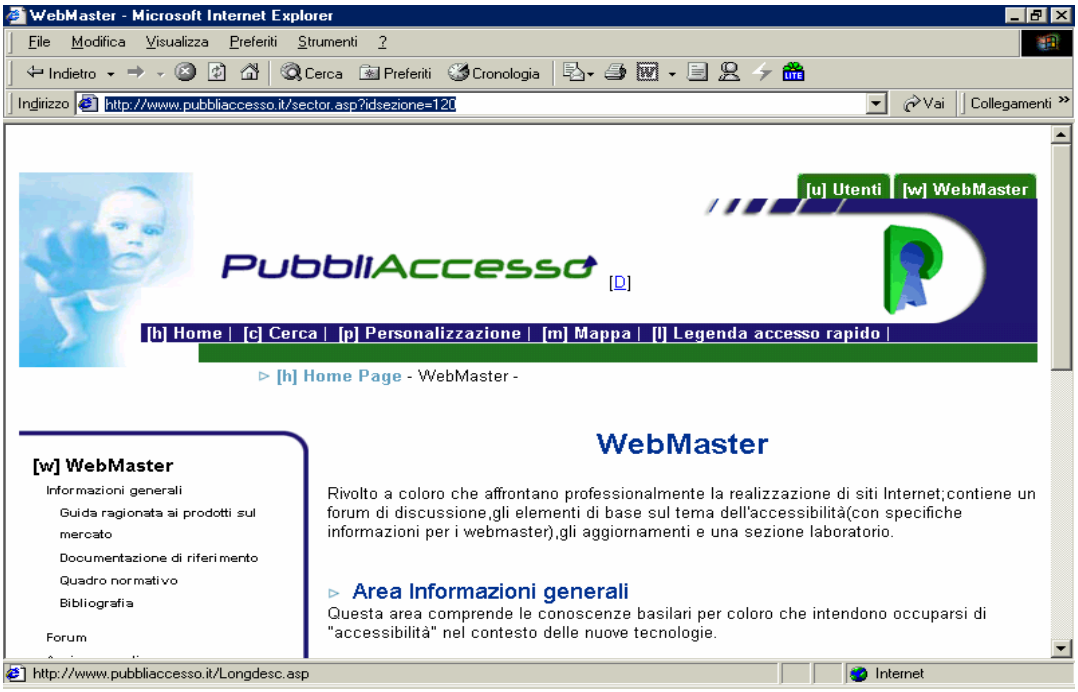

Figure 4.18 - Webmaster section (at http://www.pubbliaccesso.it/sector.asp?idsezione=120) where the multilevel menu is focalised

In order to point out the hierarchical structure of that multilevel menu, two different methods are applied: with style sheets, different font sizes and indenting allows visually impaired users to understand the hierarchical structure; by using `<UL>` tags, blind users are able to get information on multilevel menu (i.e., items and sub-items). The fragment of code is shown in Figure 4.19. The part enclosed between `<STYLE>` and `</STYLE>` refers to visual features, whereas that between `<UL>` is useful for screen reader interpretation.

```
<STYLE type="text/css">
...
A.menu { FONT-SIZE: 112% }
A.sub_menu { FONT-SIZE: 78% }
A.menu        {        FONT-WEIGHT:        bold;        COLOUR:        #000000;
         TEXT-DECORATION: none }
A.sub_menu       {       FONT-WEIGHT:       bold;       COLOUR:       #000000;
             TEXT-DECORATION: none   }
A.menu:hover { TEXT-DECORATION: underline }
A.sub_menu:hover { TEXT-DECORATION: underline }
```

```
UL.menu      {      MARGIN-BOTTOM:      10px;      MARGIN-LEFT:      10px;
          LINE-HEIGHT: 140%; LIST-STYLE-TYPE: none }
...
</style>
<body>
...
<UL class=menu>1
<LI><STRONG><A                        class=menu                        title=""
href="http://www.pubbliaccesso.it/…">Users</A></STRONG>
<UL class=menu>1
                <LI><A class=sub_menu title="Vai all'area Forum-Utenti"
href="http://www.pubbliaccesso.it/forum_utenti/…">Forum</A>
                <LI><A class=sub_menu
                title="Vai all'area Elementi di base-Utenti"
ref="http://www.pubbliaccesso.it/sector.asp…">Elementi di base</A>
...
</ul>1
<LI><STRONG><A class=menu title=""
href="http://www.pubbliaccesso.it/sector.asp…">WebMaster</A></STRONG>
...
</ul>1
</body>
```

Figure 4.19 - HTML code used to provide the multilevel menu of the page at http://www.pubbliaccesso.it

As we can observe in the next figure, the voice synthesizer (or braille display) reads the menu multilevels such as 'List of 2 items', 'List of 5 items nesting level 1', where 'nesting' indicates the submenu. We marked the bolded portions by *1,2* because those parts are both read by the screen reader (and non-visible on the screen) and involved by criteria modifications (i.e., result of `<UL>` tags).

List of 2 items1,2

- *Link*[2] Utenti

List of 5 items nesting level 11,2

- *Link*[2] Forum

• *Link*[2] Elementi di base

...

List end nesting level 11,2

• *Link*[2] WebMaster

List of 4 items nesting level 11,2

• *Link*[2] Informazioni generali

• *Link*[2] Forum

...

List end nesting level 11,2

List end1,2

Figure 4.20 - Example of hierarchical menu reading by screen reader

4.3.3 Popup Menu

A special kind of menu is the so-called pop-up menu with submenus appearing and disappearing at passing mouse. In order to make it accessible and usable, some rules should be applied. So, the purpose is building a menu that is accessible by both textual and graphical browser, and at the same time it is well organised. Obviously it should have those typical features of a popup menu (i.e., opening/closing by passing mouse, graphical features, etc.).

In the next figure an example of a popup menu is shown; the menu 'Research topics' is open.

As we can see in the picture, the menu is composed of six menu titles. Each menu title, when receives the system focus or the mouse pointer, opens and the submenu items appear. Each menu and submenu item is a link so that the screen reader is able to recognise it immediately. The submenu items should be enclosed in a list tag following the correspondence menu title; like so, users can understand that those elements are sub-items. A hidden label to mark the popup menu is advisable: the “popup menu” label makes the beginning of the popup menu known to blind users. Figure 4.22 shows a portion of a popup menu reading through the screen reader.

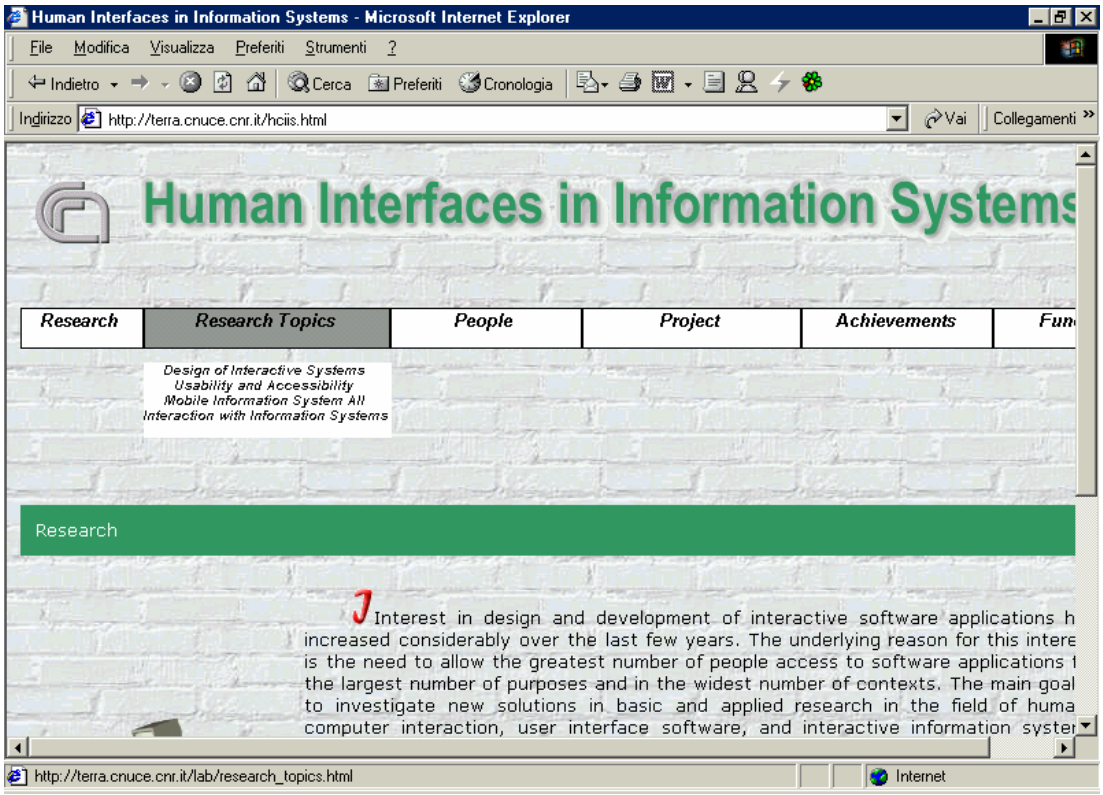

Figure 4.21 - Example of a popup menu where one submenu is open

```
Human Interfaces in Information Systems

Graphic² Popup menu:²
Link² Research
Link² Research Topics
List of 4 items²
• Link² Design of Interactive Systems
• Link² Usability and Accessibility
• Link² Mobile Information System All
• Link² Interaction with Information Systems
list end²
Link² People
List of 5 items²
• Link² Permanent Staff
• Link² Research Assistants
• Link² P.h.D. Students
• Link² Guests
• Link² Position available
list end²
…
```

Figure 4.22 - Portion of the popup menu reading through the screen reader

When a popup menu is built it is important to use a methodology that assures that all the menu and submenu items are accessible also for textual browsers and not only graphical browsers. Unfortunately some functions necessary for submenu appearing and disappearing could be built by using script languages (e.g., JavaScript), which are not supported from all the browsers. For instance, a JavaScript function may be used for the submenu opening, and another for its closing. Then some DOM methods could be applied for changing the `VISIBILITY` CSS property.

Therefore, it is important to build all the menu and submenu items as list of links. In this way information is available even if the browser does not support the JavaScript language or DOM structure.

Next figure shows a portion of code used to develop the popup menu visualised in Figure 4.21.

```
<head>
<style type="text/css">
#menu2       {
             position: absolute;  visibility: visible;
             top: 120px; left: 100px; width: 180px; height: 30px;
             }
#submenu2    {
             position: absolute ;  visibility: hidden;
             top: 160px; left: 100px; width: 180px;
             }
.hiddenlabel { display: none; }
</style>
</head>

<body>
<!--Start of pop-up menu code -->
<img src="" alt="Pop-up menu:" class="hiddenlabel">
<div id="menu2" class="menu">
<a   href="lab/research_topics.HTML"   onmouseover="overMenu('menu2',   1)"
onmouseout="outMenu('menu2',     1)"     onfocus="overMenu('menu2',     0)"
onblur="outMenu('menu2', 1)"> Research Topics </a>
</div>

<div   id="submenu2"   class="submenu"   onmouseover="overMenu('menu2',   1)"
onmouseout="outMenu('menu2', 1)">
<ul>
```

```
<li> <a href="lab/research_topics.HTML#res1"> Design of Interactive Systems
</a> </li>
<li> <a href="lab/research_topics.HTML#res2"> Usability and Accessibility
</a> </li>
      ...
</ul>
</div>

<div id="menu3" class="menu">
...
</div>

<div id="submenu3" class="submenu">
...
</div>
...
</div>
...
</body>
```

Figure 4.23 - HTML and CSS code used for building the popup menu

An important rule to follow when developing a popup menu as in the previous figure, is the logical order of the <DIV> blocks.

Considering the code, we can observe that each menu title and submenu item list is enclosed in a single <DIV> block. This is used for applying the necessary CSS styles, and for making visible and hidden each submenu block. Such a logical order is necessary to make a precise sequence of menu and submenu items. That means if DOM structure and properties are not supported, all the menu and submenu links are listed in correct way. Note that as said before, each submenu is obtained by enclosing all the link items in a list (i.e. <UL> tag). The "MENU" and "SUBMENU" classes are used to apply various and different visual features for the elements of menus and submenus. Finally, the "OVERMENU()" and "OUTMENU()" functions are called by ONMOUSEOVER, ONFOCUS, ONMOUSEOUT, and ONBLUR events to open and close submenus. Their purpose is changing visual features (e.g. the background colour of menu title), and making visible or not the submenu items (e.g. STYLE.VISIBILITY = "VISIBLE" or "HIDDEN").

4.4 Identifying Information and Elements

Structuring content and elements, so that users can be able to find more quickly the information wanted, is an important goal of our criteria. Examples of solutions of this issue are: logical organization of the information and association of shortcuts or indexing levels to elements.

4.4.1 Using Headings

A useful method by which information can be logically organized is to use appropriately heading elements. In paragraph 4.2.3 we discussed about the possibility of using appropriately layout tables by associating them to meaningful summary attributes. The same effect can be obtained by using heading tags. In fact, several screen readers have special commands to list the headings presenting in the page, so that users are able to understand, with a general overview, which is the content of the page, and they can reach some information more quickly. The next figure shows heading use applied to the Environment Department Web site: a single heading has been used for each selection category, as when using table summary attributes. We evaluated that also in the PubbliAccesso Web site headings were applied correctly (i.e., to group logically information).

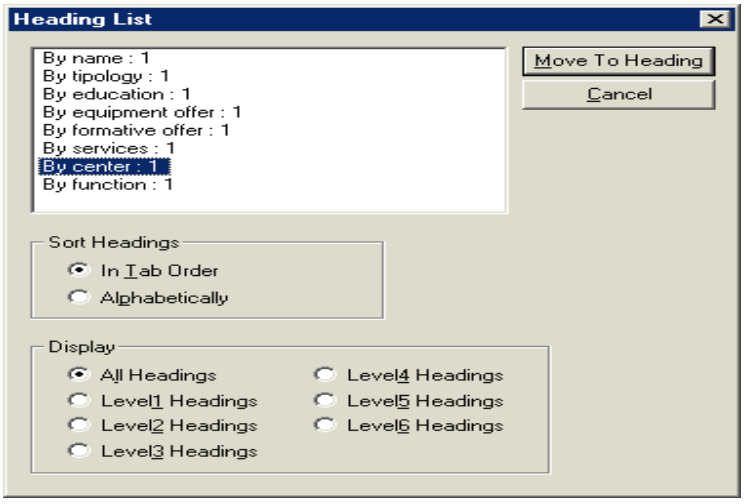

Figure 4.24 - Heading list of selection categories in the Environment Department Web site produced by a special command of screen reader

In the next figure, the HTML code which creates the layout tables of setting parameters is shown. Practically, <H1> tags have been used with image tags: the screen reader produces heading list getting values in ALT attributes. By associating these tags with image tags, there is no effect on the screen: they are interpreted only by the screen reader.

```
<table width="330" border="0" cellspacing="0" cellpadding="0"><!--INIZIO
TABELLA FILTRI NOME-->
<tr> <td colspan="2" valign="middle" height="55">
<h1> <img src="../Denominazione.gif" height="16" width="330" alt="By
name"> </h1>1
</td> </tr>
...
<table width="330" border="0" cellspacing="0" cellpadding="0"><!--INIZIO
TABELLA FILTRI TIPOLOGIA-->
<tr> <td colspan="2" valign="middle" height="55">
<h1><img src="../Tipologia.gif" height="16" width="330" alt="By
typology"></h1>1
</td> </tr>
...
<table width="330" border="0" cellspacing="0" cellpadding="0"><!--INIZIO
TABELLA TITOLO OFFERTA_EDUCATIVA-->
<tr> <td valign="bottom" height="55">
<h1><img src="../OffEdAcc.gif" height="16" width="330" alt="By
education"></h1>1
</td> </tr>
...
```

Figure 4.25 - HTML code of layout tables about selection parameters with heading tags

4.4.2 Shortcuts and Indexing Levels

Shortcuts and indexing levels associated to interaction elements are two useful tools for users navigating mainly through the keyboard. In fact, these features allow users to find and reach more quickly the elements wanted. We suggest applying shortcuts only to main links (e.g., those of the navigation bar) and not to all the links,

otherwise users cannot remember them and shortcuts are not useful anymore. In addition, it would be better to assign the letters with the initials of the link to witch the shortcuts refer. The pubbliAccesso Web site has assigned some shortcuts to the navigation bar links (see Figure 4.15), choosing first letters of the words. To assign a shortcut to a link, the accesskey attribute can be used as reported in the next figure.

```
<TD class="" style="BACKGROUND-COLOUR:#297618"><A class=menu_bar
title="Vai al settore Utenti" accessKey="u"1
href="http://www.pubbliaccesso.it/..." tabindex="10"1>[<B>u</B>]
Utenti</A></TD>
...
<TD class="" style="BACKGROUND-COLOUR: #297618"><A class=menu_bar
title="Vai al settore WebMaster" accessKey="w"1
href="http://www.pubbliaccesso.it/…" tabindex="10"1> [<B>w</B>]
WebMaster</A> </TD>
...
<TD style="COLOUR: #ffffff; BACKGROUND-COLOUR: #211771" colSpan=1> <A
class=menu_bar title="Torna all'Home Page" accessKey=”h”1
href="http://www.pubbliaccesso.it/…" tabindex="10"1> [<B>h</B>]
Home</A> |  
<A class=menu_bar title="Vai alla pagina di ricerca" accessKey=”c”1
href="http://www.pubbliaccesso.it/cerca.asp" tabindex="10"1> [<B>c</B>]
Cerca </A> |  
<A class=menu_bar title="Vai alla personalizzazione" accessKey=”p”1
href="http://www.pubbliaccesso.it/…" tabindex="10"1> [<B>p</B>]
...
<DIV class=sezione>
<A class=sezione href="http://www.pubbliaccesso.it/..." tabindex="5"1>
Area Informazioni generali </A> </DIV>Questa area comprende le
conoscenze basilari...<br>
<DIV class=sezione>
<A class=sezione href="http://www.pubbliaccesso.it/..." tabindex="5"1>
Area Forum </A> </DIV>L'area contiene... <br>
```

Figure 4.26- Example in the PubbliAccesso Web site

In order to inform users of the hot keys available during the navigation, we advise to insert a specific section 'list of available shortcuts' and add a link to the navigation

bar. We do not suggest indicating the hot key with the element name (e.g., [s] Search), because screen readers could be able to recognize the accesskey value. Consequently, in this case, users would read the associated letter twice, which would be a source of confusion. Figure 4.27 shows this situation: every link has the indication of both '[letter]' (at the beginning) and 'alt+letter' (at the end). The text relating parts of accesskey associations is marked by 1.

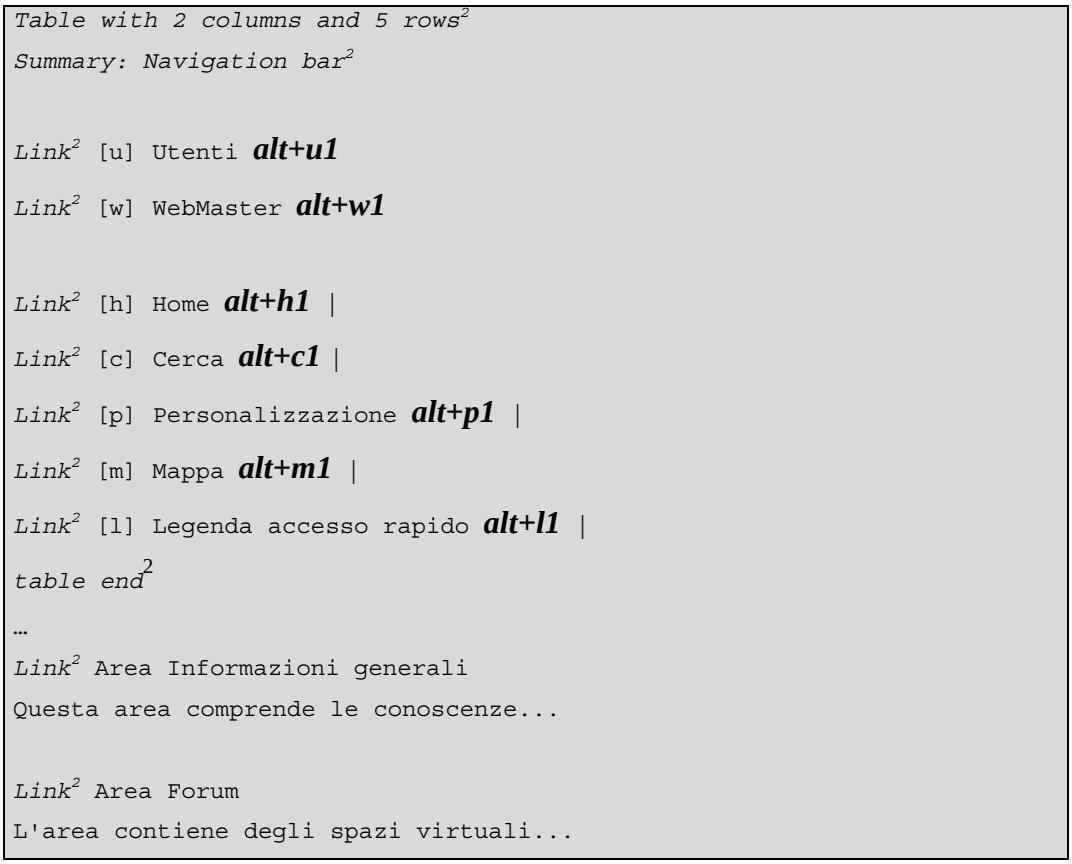

Figure 4.27 - Navigation bar links with shortcuts associated.

Another tool that allows moving more quickly on the interaction elements is the indexing level use. We suggest assigning higher values to navigation bar links, and lower values to other links in the current Web page. In this way, users (through 'tab key') visit the new elements first, then those of the navigation bar. Specifically, in our case, users first move to 'Link Area Informazioni generali', then to 'Link Area Forum', and so on (See Figure 4.27). Subsequently, through the 'tab key' the focus is placed on the first link with the next lowest value, that is 'Link [u] Utenti', 'Link [w]

WebMaster alt+w', and then on all the others. The code to implement different indexing levels is shown in Figure 4.26. Concluding, shortcuts allow users to reach immediately one element, whereas the indexing level makes a certain "order" in the interaction element navigation.

4.5 Adding Short Sounds: Using Earcons

One of our suggested satisfaction criteria is adding short sounds, so called earcons, in order to have an aural feedback that can be particularly useful for blind users. In fact, earcons are abstract, synthetic tones that can be used in structured combinations to create sound messages to represent parts of an interface. This technique models an interaction in terms of event, status and mode information and then categorises this in terms of the feedback needed to present it. A more accurate investigation is necessary in order to understand the kind and features of sounds to use (i.e. pitch, timbre, intensity, length, single or complex earcon, etc.).
The proposed criteria advices to add a sound to the page loading so that users can better understand when the page is loaded. Other application is using different sounds for different types of links or document files, and so on.

4.5.1 Different Document Formats

Let's consider the example of several application forms downloadable in different formats (See Figure 4.28). As we can see in the picture each type of format is marked with a graphical symbol: the format .DOC with a red circle, .RTF with a green circle, and .ZIP with a blue circle. The user on the basis of colour can immediately individuate the format type. For visual impaired users to have a similar utility, we could use different sounds. When the user moves the focus on links, a specific sound is reproduced.

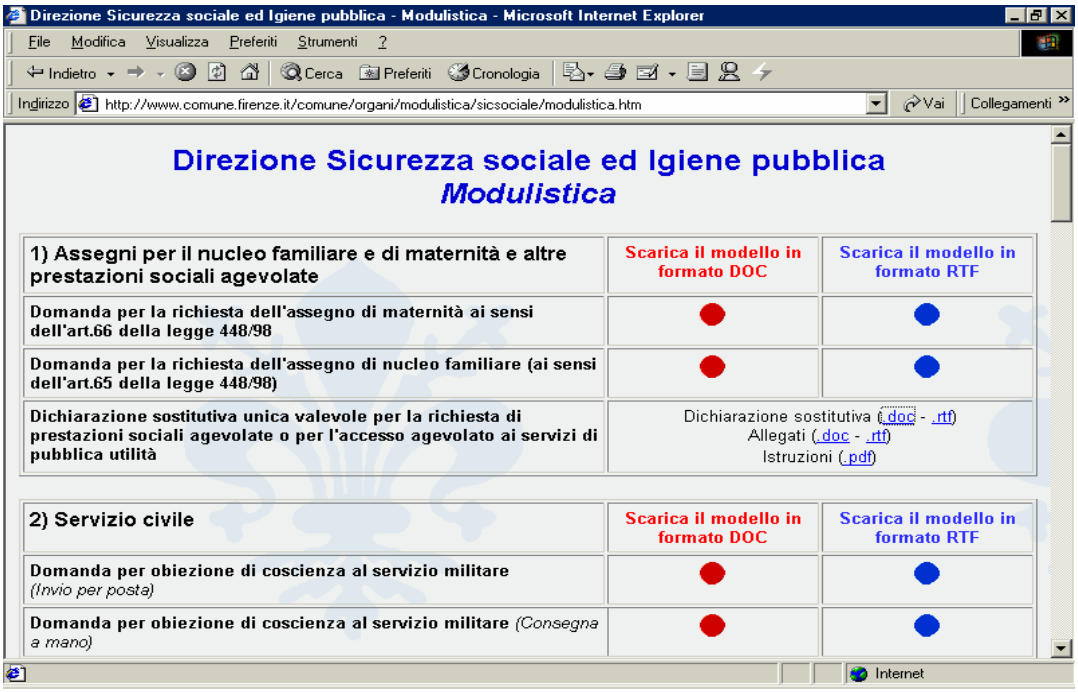

Figure 4.28 - Social security page where several downloadable application forms are available at http://www.comune.firenze.it/comune/organi/modulistica/sicsociale/modulistica.htm

Next table shows a possible implementation. By number 2 we marked the parts which are read by the screen reader and which belong to ALT or TITLE attribute values, or link text.

Visualized link	Link read by screen reader	Riproduced sound
	Formato doc - …[2]	Doc.wav
	Formato rtf - …[2]	Rtf.wav
	file zip entrambi i formati - …[2]	Zip.wav
.DOC	Formato doc - …[2]	Doc.wav
.RTF	Formato rtf - …[2]	Rtf.wav
ISTRUZIONI (FORMATO PDF)	Istruzioni (formato pdf)[2]	Pdf.wav

Table 4.1 - Correspondence of visualized links read by the screen reader, and reproduced sounds

To obtain such an effect, we have to embed sounds in the HTML document, and a JavaScript function to play those sounds.

In order to have a sound when the mouse pointer is on a link, we can use a JavaScript which plays a sound depending on the link type. So several sounds have to be embedded in the document. To discuss the evaluation and repair of this aspect, we can consider the code of the security social Web page previously discussed. In the figure a portion of code that produces the correspondence of format type and sound is copied.

```
<HTML> <head>
<script language=”JavaScript”>
function playsound(sound)1
{
document.embeds[sound].play();1
}
</script>
</head> <body>
…
<!-- download of the different formats -->
<td VALIGN=TOP WIDTH="14%" align="center"> <a
href="assegno_maternita.doc" onfocus="playsound(0)"1
onmouseover="playsound(0)"1><img src="/immagini/bot_doc.gif" border=0
alt="formato doc - assegno di maternità"3></A> </td>
<td VALIGN=TOP WIDTH="14%"> <div align="center"><a
href="assegno_maternita.rtf" onFocus="playsound(1)"1
onMouseOver="playsound(1)"1><img src="/immagini/bot_rtf.gif" border=0
alt=”formato rtf - assegno di maternità"3></a></div> </td>
<td VALIGN=TOP WIDTH="14%" align="center"> <a
href="assegno_maternita.zip" onFocus="playsound(2)"1
onMouseOver="playsound(2)"1><img src="bot_zip.gif" width="24"
height="21" alt=”file zip di entrambi i formati – assegno di maternità"3
border="0"></a> </td>
…
<td VALIGN=TOP colspan="3" align="center"> <font size="2" face="Arial,
Helvetica, sans-serif">Dichiarazione sostitutiva (<a
href="dichsost_prestsociali.doc" onfocus="playsound(1)"1
```

```
onmouseover="playsound(1)"1 title="formato doc – dichiarazione sostitutiva"3>.doc</a>
…
<a href="istruzioni_dichsost.pdf" onFocus="playsounds(3)"1
onMouseOver="playsound(3)"1> Istruzioni (formato pdf)3 </a><br>
…
<embed src="doc.wav" hidden=true autostart=false> </embed>1
<embed src="rtf.wav" hidden=true autostart=false></embed>1
<embed src="zip.wav" hidden=true autostart=false> </embed>1
<embed src="pdf.wav" hidden=true autostart=false> </embed>1
</body> </HTML>
```

Figure 4.29 - HTML code of security social Web page where the portion to reproduce a sound when the focus or mouse are on the links is in bold style and marked by 1

It is quite difficult to evaluate if this a kind of criteria has been applied by the developer, because the JavaScript function can be named by any name, as well as any other JavaScript. However, we could check the presence of code like 'DOCUMENT.EMBEDS[SOUND].PLAY()', or '<EMBED> SRC=".WAV" </EMBED>.

Then, to aid developers to apply this criterion, we insert a function like that in Figure 4.29, and for each link determining the typology (e.g., by HREF value) and add the code 'ONFOCUS="" ONMOUSEOVER=""'.

Such a repair procedure involves the entire Web site, because that criterion is useful if applied to all links of each page.

4.5.2 Different Sounds for Different Link Types

In the previous paragraph an application example of earcons usage has been shown. A similar solution could be used for various categories of links, and not only for document formats. In this way users, while they explore the page, can understand types of link more promptly.

In the prototype we built to carry out a user testing (see chapter 7), we used various sounds to differentiate the types of links. In more details, our idea was to provide some aural information about three types of links:

- *Local link*: we consider to be local a link pointing to an anchor in the page (e.g. #content);
- *Internal link*: we consider to be internal a link to a Web page belonging to the same considered Web site; for example “page.htm”, or “http://SameDomainWebSite/page.htm” in case of absolute paths;
- *External link*: we consider to be external a link pointing to some a resource out from the considered Web site (i.e., the http domain is different).

To each kind of link we associated a type of sound having different lengths:

- Short beep to local link;
- Sound like “browsing” to internal link;
- Double short sound to external link.

In practice our idea is to associate a sound length on the base of distance between pointed resource from link, and current position.

The next figure shows the portion code used to associate sounds to diverse links. Unlike the solution adopted in the previous example, in this case, in order to play a sound, we used a property of DOM structure.

```
<script                                                    language=”JavaScript”>
//       Code       to       be       called       from       each       page
//             Definition             of             sound             names
// sndpath variable needs to resolve pathways from current page position:
// its value has to be defined in each page
var snd= new Array();
snd[0] = sndpath+"onlocal.wav";
snd[1] = sndpath+"oninternal.wav";
snd[2] = sndpath+"onexternal.wav";

function playsound(sound)1
{
       document.all.music.src=snd[sound]1;
}
```

```
…
<!-- Local link -->
<a href="... " onmouseover="playsound(0)"1 onfocus="playsound(0)"1> ...
</a>

<!-- Internal link -->
<a href=" " onmouseover="playsound(1)"1 onfocus="playsound(1)"1> ...
</a>

<!-- External link -->
<a href=" " onmouseover="playsound(2)"1 onfocus="playsound(2)"1> ...
</a>
…
```

Figure 4.30 - JavaScript and HTML code to associate different sounds with different links

5
How To Implement The Proposed Criteria: Checkpoints

5.1 Introduction

In previous chapter 3, we described our proposed criteria to which developers can refer in order to improve the accessibility and usability of the Web sites. We also discussed how we intend to organize criteria and checkpoints. As we said in the paragraph 3.2, we consider an organization in two layers: the first one, the criteria set which focuses on concepts about accessibility and usability issues encountered by using special devices; and the second one, the checkpoint set which explains the specific constructs that should be considered in order to apply the criteria.

In this chapter we discuss about possible technical solutions (checkpoints) that can be used to apply the proposed criteria. A *checkpoint* is a specific construct in a language for Web pages development. While for a certain criterion only one checkpoint may exist, for another one there could be several. More precisely, the criterion application can differ over Web site implementation (e.g., usage of frames or not).

Herein we propose 54 checkpoints for the 19 criteria previously discussed. Before drawing up technical solutions, each criterion is summarised.

5.2 The Checkpoints

As mentioned in 3.3, to identify every criterion we use the notation like I.J.L (where I denotes the criterion kind, J is a progressive number to enumerate the criteria and L can be *a* or *b)*. The checkpoints associated to a certain criterion are marked with I.J.K.L (where K denotes checkpoint numbers for a same criterion). For example, for the criterion 1.1.b we define the checkpoints 1.1.1.b, 1.1.2.b and so on.

Below we describe our proposed checkpoints, by organizing them according to the criteria classification. Furthermore, for each checkpoint the 'To be applied' phrase is used to denote the code fragment that can be applied by developer, and the 'To be checked' phrase will be used for indications that can be followed by evaluator.

1. Effectiveness checkpoints

1.1.b Logical partition of interface elements

This criterion aims at logical grouping information, links, fields and so on, in order to allow user who uses special devices localizing his position and the essential parts existing in the page. In addition, with some screen readers, it is possible to skip section by section.

1.1.1.b Logical partition of interface elements: Grouping by headings

A first way to partition the content of the Web page consists of applying appropriately tag <H1>…<H6>. 'Appropriately' means that such tags should be used for the main parts of the content, and not for rendering some information in some font type (e.g., often to format heads, addresses, etc.). Thus, <H1>..<H6> tags should be applied to chapters or paragraphs titles, to the beginning of the page content, to heading of news, to different sections, and so on. With certain screen readers, it is possible to list all the headings that are in the page (i.e., similar to index), and skip heading by heading through special commands. For example, the first line of the Web page content, after any navigation bar and banners, should be a 'title' (with tag <H1>) related to the content of that page.

- *To be applied*

  <H1> … </H1>

  … OTHER CONTENT …

  <H2> … </H2>

  …

  <H2> … </H2>

- *To be checked*

Usually when the <h1>…<h6> tags are not correctly applied, we may find a possible sequence of some consecutive `<HI>` tags. However a proposed method to check if headings are used for grouping information can be searching `<HI>` tags in the page. Obviously it is a possible method, but we are not sure that headings are used correctly.

1.1.2.b Logical partition of interface elements: Marking page parts

In order to identify specific and important parts of a Web page, we can mark the beginning of each part with a hidden label that can be read by special devices. For example, it is important that user is able to recognise and to find quickly the content of the page, the navigation bar, search section (if any), and so on.

In order to mark those parts, it is possible to insert the hidden labels that are captured by a screen reader. Furthermore, the user can also look for a specific label to find precise information or position in the page (e.g., Web page content, task response, etc.). Possible labels to use can be:

Alt text	Meaning
"Navigation bar:"	To identify the navigation bar
"Sub navigation bar:" or "Menu:"	To mark a list of links which allow to skip quickly to various sections of the site
"Content" or "Page content"	To mark the beginning of the main content in the page
"Search"	To mark the area of search fields
"Results"	To mark the area of operation or search results

Table 5.1 - Possible texts for hidden mark labels

- *To be applied*

A possible way to insert hidden labels which can be captured by screen reader, but which do not appear on the screen, is by using `ALT` attribute in the `<IMG>` tag as follows:

```
<IMG SRC="" ALT="LABEL"...>
```

Noting that the attribute SRC can be empty.

- *To be checked*

In order to understand if developer has applied this checkpoint, we should verify the presence of <IMG> tag with attribute SRC empty and ALT containing a string.
In addition we should analyse all the alt text looking for typical labels, such as we listed before.

1.1.3.b Logical partition of interface elements: Grouping by frames

If Web site has been developed by using frames, grouping information means defining a specific frame for each Web page part. In addition, every frame should have an appropriate name. For more details, see the specific 'Proper name for frames and tables' criterion (checkpoint 2.2.1.b).

1.2.a Proper link content

The text of links is quite important for special users who, by using screen readers and keyboard commands (e.g., tab key), can lose the overview of the content (i.e., text written before and after a link). For this reason it is important that the links are clear and independent of context. For instance, links like '.pdf', 'more details' are dependent on the information around. So, we can work on text, alternative text or TITLE attribute in order to improve this issue.

1.2.1.a Proper link content: textual links

In order to improve the content of textual links we can change the entire text, or we can use also TITLE attribute. We suggest to use an adequate content for link text, and then to add a more precise description by using TITLE attribute. For example, about '.pdf' link, we can define a text like 'download pdf format', and then 'download pdf format of html specification'. That means the TITLE attribute should contain some more information related to link itself.

- *To be applied*

```
<A HREF="" TITLE=""> LINKTEXT </A>
```

TITLE attribute is not necessary, but it is advisable.

- *To be checked*

In order to evaluate the text associated with a link, we have to consider the <a> tag, and analyse the content of textual link and title attribute.

```
<A HREF="" TITLE=""> LINKTEXT </A>
```

It is required the analysis of the text contained between <A> and </A>. TITLE attribute is not indispensable, but if any, it is necessary to evaluate it too. As other similar situations, we could build a list composed of non-advisable texts, in order to do a semi-automatic evaluation. In addition, we could store that list in a file, so that we could have more files for more languages.

1.2.2.a Proper link content: graphical links

About graphical links, we have to apply the ALT attribute, putting a significant description which has to explain the meaning of the link rather than describing the image itself. Furthermore, similar to the textual link case, also the alternative content should be clear and should contain adequate information.

- *To be applied*

```
<A HREF="" TITLE=""> <IMG SRC="" ALT=""> </A>
```

Note that TITLE attribute in this case is optional.

- *To be checked*

```
<A HREF="" TITLE=""> <IMG SRC="" ALT=""> </A>
```

Both ALT and TITLE (if any) contents have to be analysed. As other similar situations, we could build a list composed of non-advisable texts, in order to do a semi-automatic evaluation. In addition, we could store that list in a file, so that we could have more files for more languages.

1.2.3.a Proper link content: Graphical and Textual links

In several cases developers decide to apply both textual and graphical links pointing to same resource. In fact, graphical links are often added in order to get visually more attention. In such a situation, despite the graphical links have alternative text (equal or not to textual one), could generate confusion to blind users who navigate through screen reader. We suggest putting together text and image related to same link (i.e., in the same tag <A>), so that the screen reader recognizes it as an only one link.

- *To be applied*

Generally, image, with or without ALT content, and text of link are both enclosed between the same <A> and </A> tag as follows:

```
<A HREF=""> <IMG SRC="" ALT=""> LINKTEXT </A>
```

- *To be checked*

In order to evaluate if both graphical and textual links to the same resource are used, first the presence of <A> tag containing both image and text is checked; if two near links, one graphical and other textual, having the same value associated to HREF attributes are found, this checkpoint is considered not applied.

1.3.b Messages and dynamic data management

A significant aspect pointed out by special users, who can not have the overview of the page content, concerns some difficulties in identifying new "dynamic" messages, especially those related to system response (success or failure) after having executed some operation. It depends on the fact that often the "short" message is visualized among many other data (links, banners, text and so on), and so they are not easy to be focalised. Thus, some ways aimed to pointing out "short" message are suggested.

For examples, typical situations of this kind can occur sending a form, sending an sms message, making a search, and so on. In all these cases a short reply is visualised in order to inform the users about the success of operation, or if some errors occurred. More precisely, the output the research can be a series of results, or a short message like ‘no result found’. Therefore, specific ways to focalise or organise those results should be applied.

1.3.1.b Dynamic messages: Focusing on new messages

A first way to getting attention to a response short message coming out in a Web page is moving the focus over it, when the page is loaded, so that all previous content is automatically skipped.

- *To be applied*

First an anchor 'msg' which marks the message has to be applied:

```
<A NAME="MSG"> </A>
```

Then a JavaScript can be used in the `<BODY>` tag in order to move the focus on the message:

```
<BODY ONLOAD="JAVASCRIPT:WINDOW.LOCATION='#MSG'">
```

- *To be checked*

In order to evaluate if this checkpoint is used, checking if in the page there is an anchor ‘msg’ and a JavaScript fragment that moves focus on it. However, evaluating if developer used this checkpoint is not so easy, because it is mainly related to dynamic Web pages, i.e. the inspection should be carried out on pages generated by server rather then on the source code.

1.3.2.b Dynamic messages: Using a dialogue window

A simple way to manage replies to users is to provide a little message window, called ‘alert box’ (or Popup Message), containing a short response message.

- *To be applied*

In order to visualise the message, an alert box is sufficient, however prompt and confirm boxes can be used.

```
ALERT ("");
CONFIRM ("");
PROMPT ("");
```

- *To be checked*

Checking if Alert, Prompt or Confirm boxes are used is easy; but verifying if they are used correctly is fairly difficult automatically, so participation of evaluator is suggested.

1.3.3.b Dynamic messages: Opening a new window

When the system gives a response message in order to inform the user on a certain task result, developers can display it in a new window, so that it is clearly identifiable.

We suggest using this method especially when the response is not a short message, but a lot of information is provided. More precisely, a new window should be used to show a list of search results, definition of a some word, description in more details of news, and so on.

- *To be applied*

A first solution does not use the JavaScripts, but `_BLANK` or `'FRAME NAME'` as destination target.

```
<A HREF="" TARGET="_BLANK"> </A>
<A HREF="" TARGET="FRAMENAME"> </A>
```

However it is possible to use also a JavaScript in order to open a new window. This second solution gives some additional options, such as setting window properties. For example one specific parameter allows to show or to hide the menu bar. The presence of the menu bar is enough important, because users could not be able to copy the page content. In fact, by using the keyboard selecting the content is possible only through the item 'Select all' that is in the 'Edit' menu. Or, if users want to save the page in an external file, the 'Save as' in the 'File' menu is necessary. Therefore it is important that the JavaScript function does not have the parameter `"MENUBAR=NO"`. Also, it is advisable to set the resizable property to 'yes', so that users are able to magnify the window. This could be useful for visual impaired people.

The function below is an example of one possible use in the pages to open new windows. That JavaScript function can be used with links and buttons to open a new window with the related url.

```
WINDOW.OPEN (URL, WINDOWNAME, "MENUBAR=YES, DIRECTORIES=NO,
LOCATION=NO, RESIZABLE=YES")
```

- *To be checked*

In order to evaluate if developer used new window opening, checking either the use of `TARGET=_BLANK` or the `WINDOW.OPEN()` function is associated with special statements. We suggested that a new window should be opened to show dialogue messages or in special way to list the results found by a search function. So the use of a new window is important especially in these cases.

1.4.a Proper style sheets

One of the most important features of style sheets is that they allow specifying how a document has to be presented on different media, such as screen, paper, with a speech synthesizer, with a braille device, etc. If the user agent is able to recognize the various devices existing in the system, using different styles is advisable.

Thus, it is useful to express that a style sheet, or a section of a style sheet, applies to certain special media types. In order to improve navigation for blind or visual impaired people, voice synthesizer, display and printer braille devices are taken into account. Currently, there are two ways to specify media dependencies for style sheets: specify the target medium within html documents, or specify the target medium from a style sheet (with the @media or @import at-rules). So, for each media type considered, we suggest using both methods.

1.4.1.a Proper style sheets: Voice synthesizer

When a voice synthesizer is installed, it is very likely that the system is used by blind or visual impaired people; so some specific features could be applied in order to improve their navigation.

- *To be applied*

The code relating to style sheet loading applies In the `<HEAD>` section of page.

Specification within html document:

```
<STYLE TYPE="TEXT/CSS" MEDIA="AURAL"> ... </STYLE>
```

or

```
<LINK REL="STYLESHEET" MEDIA="AURAL" HREF="AURAL.CSS"
TYPE="TEXT/CSS">
```

Specification from a style sheet:

```
@IMPORT URL("LOUDVOICE.CSS") AURAL;
```

or

```
@MEDIA AURAL {
/* STYLE SHEET FOR VOICE SYNTHESIZER GOES HERE */
}
```

- *To be checked*

In the <HEAD> section of the page, the presence of the style sheet code for media aural device can be checked. Alternatively the presence of "aural" directly in the style sheet can be verified.

1.4.2.a Proper style sheets: Braille display

Existing braille displays have several features and configuration options. Usually the display is composed of 40, 60 or 80 braille cells, so if the paging up takes into account this aspect, user can read the content easier.

- *To be applied*

Specification within html document:

```
<STYLE TYPE="TEXT/CSS" MEDIA="BRAILLE"> ... </STYLE>
```

or

```
<LINK REL="STYLESHEET" MEDIA="BRAILLE" HREF="BRAILLE.CSS"
TYPE="TEXT/CSS">
```

Specification from a style sheet:

```
@IMPORT URL("BRAILLEDISPLAY.CSS") BRAILLE;
```

or

```
@MEDIA BRAILLE {
/* STYLE SHEET FOR BRAILLE DISPLAY GOES HERE */
}
```

- *To be checked*

In the <HEAD> section of the page, the presence of a type style sheet loading for media braille device can be checked. Alternately the presence of "braille" directly in the style sheet can be done.

1.4.3.a Proper style sheets: Braille printer

A braille printer has different features from a normal printer. For example it can print 40 lines per page, and 38 characters per line. Furthermore other special codes are requested for a braille printing. Therefore the use of specific style sheets is advisable.

- *To be applied*

Specification within html document:

```
<STYLE TYPE="TEXT/CSS" MEDIA="EMBOSSED"> ... </STYLE>
```

or

```
<LINK REL="STYLESHEET" MEDIA="EMBOSSED" HREF="EMBOSSED.CSS"
TYPE="TEXT/CSS">
```

Specification from a style sheet:

```
@IMPORT URL("BRAILLEPRINTER.CSS") BRAILLE;
@MEDIA EMBOSSED {
/* STYLE SHEET FOR BRAILLE EMBOSSED GOES HERE */
}
```

- *To be checked*

In the <HEAD> section of the page, the presence of the style sheet code for media embossed device can be checked. Alternatively the presence of "embossed" directly in the style sheet can be done.

1.5.b Layout and Terminological Consistency

Terminological consistency is a usability feature which allow users to understand better the context and available operation. So, label for particular buttons, links, etc., are quite important, especially for users who can have a restricted overview of content, in order to limit possible unclear situations.

1.5.1.b Terminological consistency: Button labels

Button features have a very strong impact on the user: it is important that all the pages of the whole Web site do not use different labels for buttons performing the same function (e.g. OK/Yes, quit/exit, next/forward).

It is also important that screen readers are able to read the value or alt (if a graphical button is used) content.

- *To be applied*

Even if the value attribute in some case it is not necessary, we suggest using it, because screen readers could read its content, so that users get button information from it.

```
<INPUT TYPE="SUBMIT" VALUE="LABEL">
<INPUT TYPE="RESET" VALUE="LABEL">
<INPUT TYPE="BUTTON" VALUE="LABEL">
<BUTTON VALUE="LABEL"> BUTTONTEXT </BUTTON>
```

For 'label' terms, developer should apply the same values corresponding to elements having the same functionality. In addition to the last case, which is `<BUTTON>` tag, it is important that value and ButtonText are the same.

- *To be checked*

With regard to labels of buttons we have to check whether only one 'label' corresponds to the same function. By using an automatic control, possible synonymous words can be stored in a file of synonyms. Then in order to check the button labels, if a button has a certain value (e.g., 'next'), other synonyms should not exist (e.g., 'forward').

1.5.2.b Layout consistency: Button dimension

For visually impaired users who mainly rely upon dimension/colour references, dimension of buttons may be a significant aspect: buttons have to be clearly identifiable, and they should always have the same dimension.

- *To be applied*

The same values of WIDTH and HEIGHT should be applied to all button tags.

```
<INPUT TYPE="SUBMIT" WIDTH="" HEIGHT="">
<INPUT TYPE="RESET" WIDTH="" HEIGHT="">
<INPUT TYPE="BUTTON" WIDTH="" HEIGHT="">
<BUTTON WIDTH="" HEIGHT=""> </BUTTON>
```

- *To be checked*

All the button tags have to be checked in order to evaluate if they contain the WIDTH and HEIGHT attributes, and if those attributes have the same values.

1.5.3.b Terminological consistency: Link to home page

The link to the home page should have a clear notation like 'home page', 'home', and other expressions should be avoided, because they could cause uncertainties.

- *To be applied*

```
<A HREF=""> HOME PAGE </A>
```

or, if a graphical link is used

```
<A HREF=""> <IMG SRC="" ALT="HOME PAGE"> </A>
```

- *To be checked*

Checking if a link containing the notation 'home page' exists. Then, in order to evaluate all <A HREF=""> tags, the text (or alternative image description) contained between <A> and </A> has to be analyzed.

2. Efficiency checkpoints

2.1.b Number of links and frames

The number of frames and links present in a Web page can have a certain effect on navigation with special devices. The page should not have either many or too few links: many links take long time in content reading, too few links may imply too levels in the Web structure. In order to suggest some proposed ranges, we considered a possible Web site and page taxonomy.

2.1.1.b Number of frames

In order to suggest some number of frames which can be used, we refer to possible Web site types. In fact, usually frame structure act on whole Web site, or a certain portion, not only on a single page.

Type of site	Suggested number of frames
News and information	3 - 5
Reading and documentation	2 –3
Search	2 -3
Services	3 - 5
Generic	2 - 3

Table 5.2 - Suggested number of frames according to Web site typology

- *To be applied*

The tag related to this checkpoint is

```
<FRAMESET ROWS="" COLS="">
<FRAME SRC="" NAME="" TITLE="">
…
<FRAME SRC="" NAME="" TITLE="">
</FRAMESET>
```

The number of frame elements to be applied according to Table 5.2. In addition, developers should pay attention to possible other frame structure of pages loaded in the frames.

- *To be checked*

To evaluate the number of frames used, we have to check how many `<FRAME>` are in the main page, where is the tag `<FRAMESET>`, and we have to refer to Table 5.2.

2.1.2.b Number of links

In order to indicate an advisable range of number of links, we consider possible Web page typologies.

Type of page	Suggested Number of Links
Contend reading	< 20
Data input	< 15
Index	< 50
Downloading and Saving	< 30
Search and data retrieval	< 20

Table 5.3 - Suggested number of links according to Web page type

- *To be applied*

The tag involved by this checkpoint is `<A>`, but not those used to insert some anchors.

```
<A HREF="" > </A>
```

The number of this tag for a certain page should belong to a range according to Table 5.3.

- *To be checked*

We have to count how many links are in a page, in order to check if the number is in the range according to Table 5.3. Namely, we consider the tag

```
<A HREF="" > </A>
```

2.2.b Proper name for frames, tables and images

When the content is organized by using frames or tables, it is important that the corresponding names and summary texts are appropriate. Most screen readers have special commands to show a frame list and to skim through of tables. For this reason, in order to enable the users to reach their goal, using appropriate text is important. For instance, if the name used for frame associated to navigation bar is like ‘top frame’ or ‘left frame’, this might not result really useful to the user. A similar case can occur if the navigation bar is rendered by a table and its summary is like ‘this table is used for layout purpose’.

2.2.1.b Frames: Proper name and title

The tag <FRAME> have two attributes relating to names, that is NAME and TITLE. Both attributes are captured by screen reader, so it is important that they have the same values, otherwise probably it might confuse the user.

- *To be applied*

With regard to <FRAME> both NAME and TITLE ATTRIBUTES have to be used with the same value.

```
<FRAME SRC=”” NAME="TEXT" TITLE="TEXT"…>
```

- *To be checked*

We have to check that frames have the same value corresponding to NAME and TITLE attributes, and then we have to estimate/assess the value.

```
<FRAME NAME="" TITLE="">
```

Analyzing automatically if a value is appropriate or not, is not so easy. Suggested and non-suggested name lists can be stored in external files so that it is possible to have different files for different languages.

2.2.2.b Frames: Proper page title

Usually developers do not care about the <TITLE> related to pages visualized inside frames, because titles are not shown on the screen. As page title could be read by screen reader as the beginning of the page, it is important that its value is appropriate. Also when, for example, a decorative image or a background are shown in a specific frame, the page title should not be empty, but should contain a brief description of the image and its purpose, in order to make user understand its content (e.g., ‘decorative image’).

- *To be applied*

Relating to Web page called by SRC attribute, the tag <TITLE> should not be empty.

```
<TITLE> TEXT </TITLE>
```

- *To be checked*

It is not easy to evaluate if a title is appropriate for a Web page. However, if developer used a specific design program to create the page, the value of <TITLE> tag is not empty, it is like 'Untitled Document'. So in this case we can say that it is not appropriate.

```
<TITLE> TEXT </TITLE>
```

So we can analyze the 'TEXT' value, and pointing out when that value is empty or like 'untitled document'.

2.2.3.b Tables: Proper summary value

The presence of SUMMARY attribute is useful because the vast majority of screen readers is able to read it so that user can understand at once the table content. In addition, many screen readers have special commands to skip table by table, reading the SUMMARY attribute value if it exists, or otherwise the first cell content. Although we do not suggest using tables as content formatting, usually when the table is used as layout, and not to show data content, developer does not care of its summary text, and he writes something like 'this table is used for layout purposes'. We believe that even in this case an appropriate text should be used. For example, if the table is used to contain the navigation bar links, the summary text could be 'navigation bar'. If a layout table is used to format the page content, we suggest applying a text like 'page content'.

- *To be applied*

<table summary="text"> … </table>

Where text should be appropriate to table content or usage.

- *To be checked*

```
<TABLE SUMMARY="TEXT"> … </TABLE>
```

Analyzing <TABLE> tag in order to understand if table has been used as layout or data rendering. We can consider tables with only one row or column as layout tables. Then we have to take into account summary text in order to evaluate if it is used appropriately. For example, we can evaluate the summary text for layout tables used for navigation bar, page content, search result visualization, and so on.

2.2.4.b Images: Proper description

With regard to images, a short or long description can be used in order to allow blind users to get information on graphics. A description can be associated to images through ALT or LONGDESC attributes. Usually we suggest associating a description which is what the images render. We suggest that in specific cases it is better to provide a description which informs user about the functionality of that graphic rather than the real description of it (e.g., graphical link to go to a paragraph). Then, we suggest using LONGDESC attribute for cases in which a more complete description is important. Furthermore ALT attribute should not be empty and should not contain simply the name of files corresponding to the picture (e.g., 'image.jpg').

- *To be applied*

For short description

```
<IMG SRC="FILENAME.EXT" ALT="DESCRIPTION">
```

For long description

```
<IMG SRC="IMAGEFILE.EXT" ALT="SHORT DESCRIPTION">
<A HREF="LONGDESCFILE.HTM" TARGET="_BLANK">[D]</A>
```

Where 'LongDesckFile.htm' is an external file containing a more precise and accurate description of the image.

- *To be checked*

In the <IMG> tag checking that ALT attribute is not empty and not containing simply the file name, i.e., searching in the string if there is substring like 'gif', 'jpg', etc..

2.3.a Location of the navigation bar

Pointing out the navigation bar, if any at the top and/or the bottom of the page, can be useful in order to enable users who are not able to see its visual features (e.g., horizontal or vertical position, colour or font types, etc.) to understand it . Thus other features can be used in order to localize the navigation bar with a screen reader. So we could mark the beginning of the static links, insert a particular link to skip the bar, and so on. Specific labels and strings can be used in order to identify the navigation bar, such as 'navigation bar', 'navbar', 'menu bar', 'navigation sub_bar'

and so on. All these possible alternatives, should be stored in a dictionary file in order to make easier the checking phase. In addition, different dictionary files can be created for different languages.

2.3.1.a Navigation bar: Using hidden label

A first way to mark the navigation bar beginning is by using a specific hidden label that can be interpreted by a screen reader (or textual browser), but which does not appear on the screen. As we discussed for the criterion 1.1.b (precisely checkpoint 1.1.2.b) a possible solution is by marking with significant hidden label; so for this purpose, the text label should be like 'Navigation bar'.

- *To be applied*

```
<IMG SRC="" ALT="NAVIGATION BAR:">
```

For label text some possible alternatives can be used: 'Navigation bar', 'nav bar' or 'menu bar'.

- *To be checked*

We should look for the presence of `<IMG>` tag as follows

```
<IMG SRC="" ALT="NAVIGATION BAR">
```

Then we have to analyze the alt text by comparing it to possible labels, sutch as 'navigation bar:', 'nav bar:', and 'menu bar:'

2.3.2.a Navigation bar: Using Iframe

A possible solution to make different the navigation bar within the page is by using `<IFRAME>` tag. This solution can be useful when the Web site does not use frames, but we would like to put navigation bar links in a specific area, with background and text colours different from those used in the page.

- *To be applied*

We put the links composing the navigation bar in the file 'navbar.htm'; then we insert that content by means of `<IFRAME>` tag in every page we like to show the navigation bar.

```
<IFRAME TITLE="NAVIGATION BAR" NAME="NAVBAR" SRC="NAVBAR.HTM"
WIDTH="XXX" HEIGHT="XXX" SCROLLING="NO" FRAMEBORDER="1">
[<A HREF="NAVBAR.HTM"> NAVIGATION BAR </A>]
</IFRAME>
```

Noting that the content wrapped by square brackets is necessary for browsers which does not support iframes.

- *To be checked*

We look for the presence of <IFRAME> tag, and we check the text of attributes TITLE and NAME by comparing them with possible labels (e.g., 'navigation bar', 'navbar', etc.).

2.3.3.a Navigation bar : Using frames

If the Web site is developed by using frames, one of them should be assigned to navigation bar. In this way, the user is able to identify it more quickly from the frame list. In order to enable the screen reader to read the names of frames, it is necessary that each <FRAME> has an appropriate text corresponding to TITLE and NAME attributes.

- *To be applied*

```
<FRAME SRC="" TITLE="NAVIGATION BAR" NAME="NAVBAR">
```

As the previous cases, the text used for the two attributes, can be chosen among some possible ones (e.g., 'navigation bar', 'nav bar' or 'menu bar').

- *To be checked*

For each <FRAME> tag, we have to check the presence and the value of two attributes TITLE and NAME. Furthermore, those values have to be compared with list of strings referring to navigation bar.

```
<FRAME SRC="" TITLE="NAVIGATION BAR" NAME="NAVBAR">
```

2.3.4.a Menus and submenus: Using lists

When the navigation bar or navigation sub-bar include many links (i.e., items and sub-items), lists can be used. When we build a menu composed of sub-items, it is important to understand what main items and sub-items are. So, unordered list (i.e., <UL>) can solve this issue.

- *To be applied*

As <UL> tag can be used for any kind of unordered list, it is necessary to insert a hidden label marking the beginning of the list (see 2.3.1.a). Then more nested <UL> can be associated to each level.

```
<UL>
<LI> <A HREF=""> FIRST ITEM: </A> </LI>
<UL>
   <LI> <A HREF="">FIRST SUB-ITEM </LI>
   …
</UL>
<LI> <A HREF=""> SECOND ITEM </A> </LI>
</UL>
```

- *To be checked*

First, we have to check the presence of hidden label associated with navigation bar (See the 'code to be checked' in 2.3.1.a); then we have to check the possible sequence of <UL>.

2.3.5.a Popup menu: Using disappearing blocks

If the developer decides to apply a popup menu for grouping the main navigation links, it is important to ensure its accessibility, even if some typical features of popup menu might be lost with some browsers. Furthermore, the hierarchical structure of the menu has to be kept.

For this reason it is important:

- Presenting the menu titles and all submenu items by means of ;
- Enclosing each menu title and every submenu item group in distinct blocks;

- Placing in logical order menu titles and submenu groups;
- Using lists for creating the submenus to maintain the hierarchical structure;
- Using JavaScript functions rather than submenus only to make visible or to change rendering features.

- *To be applied*

The code we suggest applying consists of <DIV> blocks with absolute positions and visibility properties, CSS styles to control the rendering features, and JavaScript functions to change colours and (dis)appearing functions.

The suggested CSS styles are:

```
<STYLE TYPE="TEXT/CSS">
#MENU1 {
   POSITION: ABSOLUTE;  VISIBILITY: VISIBLE;
   TOP: …;   LEFT: …;
   WIDTH: …;   HEIGHT: …;
   }
... //OTHER MENU AND SUBMENU BLOCK INFORMATION
.MENU {...} //RENDERING FEATURES FOR MENU ITEMS
.SUBMENU {...} //RENDERING FEATURES FOR SUBMENU ITEMS
</STYLE>
```

The html code for the ordered blocks could be:

```
<DIV ID="MENU1" CLASS="MENU">
<A HREF="" ONMOUSEOVER="OVERMENU('MENU1', 1)"
ONMOUSEOUT="OUTMENU('MENU2', 1)" ONFOCUS="OVERMENU('MENU1',
0)" ONBLUR="OUTMENU('MENU1', 1)"> MENUTITLETEXT </A>
</DIV>
<DIV ID="SUBMENU1" CLASS="SUBMENU"
ONMOUSEOVER="OVERMENU('MENU1', 1)"
ONMOUSEOUT="OUTMENU('MENU1', 1)">
   <UL CLASS="SUBMENU"> //SUBMENU ITEMS BY A LIST
   ...
```

```
   </UL>
</DIV>
```

The JavaScript functions are:

```
FUNCTION OVERMENU (IDMENU, SM)
{
 DOCUMENT.GETELEMENTBYID(IDMENU).STYLE.BACKGROUND = "#…";
   IF (SM==1)
   DOCUMENT.GETELEMENTBYID('SUB'+IDMENU).STYLE.VISIBILITY =
"VISIBLE";
}
FUNCTION OUTMENU (IDMENU, SM)
{
   DOCUMENT.GETELEMENTBYID(IDMENU).STYLE.BACKGROUND = "#…";
   IF (SM==1)
   DOCUMENT.GETELEMENTBYID('SUB'+IDMENU).STYLE.VISIBILITY =
"HIDDEN";
}
```

- *To be checked*

In order to check if a popup menu is built correctly, we could look for a possible sequence of `<DIV>` blocks having `IDS` attributes equal to “menu” and “submenu” values. Then the submenu item links should be grouped in a `<UL>` tag.

2.4.b Importance levels of elements

In order to facilitate the navigation, especially when the keyboard is considered, it is possible to assign different importance values to interaction elements (i.e., links, buttons, fields). In this way, when the users move through the 'Tab' key (element by element), they first visit those with a higher level and then those with a lower one. For instance, links belonging to navigation bar could have a lower level, whereas those belonging to the current page might have a higher level. Consequently, the users might find more quickly the new elements.

2.4.1.b Importance levels: Tabbing order position

The attribute to use in order to assign a level to one element is TABINDEX. Navigation proceeds from the element with the lowest tabindex value to the element with the highest value (i.e., tabindex values and importance levels are inversely proportional). Values need not be sequential nor they must begin with any particular value. Elements that have identical tabindex values should be visited in the order they appear.

- *To be applied*

```
<A HREF="" TABINDEX=""> </A>
<BUTTON TYPE="BUTTON" TABINDEX=""> </BUTTON>
<INPUT TYPE="" TABINDEX="">
<SELECT TABINDEX=""> </SELECT>
```

- *To be checked*

In order to evaluate if different levels have been applied, a search of TABINDEX presence can be executed. In addition, a possible sequence of progressive values can be verified.

2.5.b Assignment of shortcuts

Assigning hot keys to main links or buttons can be useful in order to allow users to move more quickly through the main parts of the Web site. This feature may be helpful especially when users visit frequently the Web site, and they learn by heart the key combinations. In addition, a specific link (possibly hidden) to a page showing the list of correspondences should be added to the navigation bar.

2.5.1.b Keyboard shortcuts: Main links, buttons and fields.

Assigning mnemonical shortcuts to main links, buttons and fields can improve the navigability chiefly through keyboard. Using shortcuts with the first letter of the link button and field text is suggested.

- *To be applied*

In order to assign shortcuts the ACCESSKEY attribute can be used.

```
<A HREF="" ACCESSKEY=""> </A>
<BUTTON TYPE="BUTTON" ACCESSKEY=""> </BUTTON>
<INPUT TYPE="" ACCESSKEY="">
<SELECT ACCESSKEY=""> </SELECT>
```

- *To be checked*

Checking if the tags related to interaction elements (links, buttons, etc.) have ACCESSKEY attribute. Moreover, a comparison between used letter and the first character of the corresponding element can be performed.

2.5.2.b Keyboard shortcuts: New interaction elements

Usually in a Web page there are the navigation bar links, then, in case, some advertisement banners, and then new text and elements related to the current page. In order to identify more quickly those new elements of the current page, a possible strategy consist of applying to all new elements the same shortcut. In this way users, by pressing that shortcut, can skip across the various elements. So, by using the same shortcut for all the new links (or other elements) it is possible to visit only the new objects, by pressing repeatedly that key combination, so avoiding all the others.

- *To be applied*

The ACCESSKEY attribute with the same key value can be applied to all tags related to current page.

```
<A HREF="" ACCESSKEY="LETTER"> </A>
<BUTTON TYPE="BUTTON" ACCESSKEY="LETTER"> </BUTTON>
<INPUT TYPE="" ACCESSKEY="LETTER">
<SELECT ACCESSKEY="LETTER"> </SELECT>
```

- *To be checked*

Verifying if there is a possible set of elements which have a same ACCESSKEY value.

2.6.a Proper form layout

To improve clarity of forms dealing with several groups of data, an appropriate layout should be used for grouping titles and fields. In fact, the specific arrangement of elements within the page might be confusing when screen readers are used. For example, in some cases the voice synthesizer or braille display could read before the 'checkbox', 'combobox' or 'field' item, and after its value, or vice versa. Often blank characters not appropriately used can be cause of difficulties in reading. Thus, a correct application of layout elements (e.g., simply by using the return tag in the proper place) is recommended.

2.6.1.a Proper form layout: Button labels

It is important that all the buttons but those rendered through an image have VALUE attribute, since screen readers are able to get that value. So, we suggest using VALUE also for ‘submit’ and ‘reset’ buttons. With regard to graphical buttons, the VALUE attribute is not necessary, because in this case speech synthesizer (or braille display) reads the ALT text. If VALUE attribute is not used, screen reader gets out only ‘button’ word.

- *To be applied*

```
<INPUT TYPE="SUBMIT" VALUE="">
<INPUT TYPE="RESET" VALUE="">
<INPUT TYPE="BUTTON" VALUE="">
<INPUT TYPE=”IMAGE” SRC=”” ALT=””>
<BUTTON VALUE=""> BUTTONTEXT </BUTTON>
```

- *To be checked*

Considering all tags by which buttons can be built, and verify the presence of value attributes. The check is not necessary for image buttons, however the alt text is considered.

2.6.2.a Proper form layout: Groups of control elements

In order to group control elements the <FIELDSET> tag can be used. In this case we recommend that a set of control elements related to the same subject should be grouped together in a <LEGEND>. The text related to <legend> is shown on top of the panel containing the set of logically grouped controls. When the screen reader read it, if the text is not appropriately formatted, the screen reader puts together the legend text with the first text belonging to <FIELDSET>. For this reason it is important that </LEGEND> is followed by
 or, better, <P>. In addition, according to criterion that suggests grouping information by headings (1.1.1.b), <HI> tags can be applied to text appearing between opening and closing legend tag.

- *To be applied*

```
<FIELDSET>
   <LEGEND> … </LEGEND> <P> …
</FIELDSET>
```

Or by using a blank line

```
<FIELDSET>
   <LEGEND> .. </LEGEND> <BR>
…
</FIELDSET>
```

- *To be checked*

First of all, checking that <FIELDSET> tags have <LEGEND> tag. Then, verifying the presence of
 or <P> after </legend>.

2.6.3.a Proper form layout: Onchange event

The use of onchange element can create difficulties to users who navigate especially tusing a keyboard; so, using other elements (e.g. onclick event) to obtain a similar effect. Infact, ONCHANGE event causes an immediate effect (e.g., based on a certain JavaScript), therefore this could be an important problem especially interacting through keyboard. ONCHANGE event is used particularly with combobox, which allows users to choose an option among several available in a list; as soon as an item

of combobox receives focus, the action associated to ONCHANGE is activated. This does not let users pressing another key to move in the list. So, a possible solution is to use a button which allows to activate the action associated to combobox.

- *To be applied*

```
<FORM>
…
<SELECT ONDBCLICK=”” > … </SELECT>
<INPUT TYPE=”BUTTON” VALUE=”GO” ONCLICK=”” >
</FORM>
```

- *To be checked*

As problems depending on ONCHANGE event are related to <SELECT> elements, it is necessary to verify that no checkboxes have onchange event.

2.6.4.a Proper form layout: Matching labels and input elements

In a form when focus moves from input element to input element, it is important that it is clear which is the text associated to input object. In checkpoint 2.6.1a we have shown how to give a name to a button, so that screen readers are able to read it. A similar case should be obtained for other input elements. This can matching <LABEL> tags to text related to a input object. Consequently, when users move on an input object, first of all the screen reader should say the label, then the type of input element. Unfortunately, if <LABEL> is not used by developers, the screen reader could say only the type of input object, without informing users about its label. This issue is particularly important when a table is used to render the form on the screen, especially when label and object are placed in different cells. In fact, in this case the screen reader could not be able to understand the pairs of label-object, and in some cases it could associate them wrongly.

- *To be applied*

```
<LABEL FOR="OBJECT-ID”> TEXTLABEL </LABEL>
<INPUT ID="OBJECT-ID" TYPE="">
```

- *To be checked*

Checking if each form interaction element has a label associated by <LABEL FOR>.

2.7.b Specific section

Providing a specific Web page containing the list of updates, ordered by date, can significantly improve information retrievals. In fact, identifying new information, data and most recent versions (placed in several pages), is not so easy when the reading is in sequential way. Therefore an available quick list of updates (equipped with links to relating pages) can be a useful tool to save time. In addition to Web page containing updates, a specific link to that page should be inserted to home page or to navigation bar.

2.7.1.b Specific sections: Last updates

- *To be applied*

In the home page or navigation bar

```
<A HREF="UPDATES.HTM" TITLE="LAST UPDATES"> LAST UPDATES </A>
```

In the head section of Web page 'updates.htm'

```
<TITLE> LAST UPDATES </TITLE>
```

In the body section, organizing information by date, and add to each update the link to Web page having the corresponding update content.

- *To be checked*

Verifying the existence of a link in the Web page (it is not important checking if it is also in other pages)

```
<A HREF="UPDATES.HTM" TITLE="LAST UPDATES"> LAST UPDATES </A>
```

Then, finding a page named 'updates.htm' and having 'Last updates' title

```
<TITLE> LAST UPDATES </TITLE>
```

2.7.2.b Specific sections: List of assigned keys

In order that shortcuts are used for navigating with the keyboard, a specific section where the key combinations are available is suggested. In fact not all the screen readers are able to inform user that for a certain object a shortcut is available. To

allow users to get information about hot keys available, a specific link (possibly hidden) to a page contains the corresponding list can be added to the navigation bar.

- *To be applied*

The following code produces a hidden link with 'shortcut list' text.

```
<A HREF="KEY-LIST.HTM" TITLE="SHORTCUT LIST"> </A>
```

However, by introducing the description between `<A>` and `</A>` tags, the link will be visible.

- *To be checked*

In order to evaluate if a specific session concerning a list of usable shortcuts is available, we can search a link (i.e., <a> tag) with title attribute or text link similar to 'key list' string.

2.8.b Indexing of contents

When the information can be structured in some way, it is useful putting indexes at users disposal, so that they can navigate among information more quickly. Indexes can be created for information which can be divided in chapters, paragraphs, for texts which can be logically split into more parts (single news, groups per ltters, and so on). For this reason two different index types can be built: index to more pages (e.g., chapters and paragraphs) and index internal to Web page (e.g., index to each letter group, or each single piece of information).

2.8.1.b Indexing of content: Highlighting each part

This checkpoint is applicable especially to index information contained in a single Web page. More precisely, we refer to information which can be logically divided in single blocks or individual information (e.g., a page of news), where each one has a title. This method does not build a real index, but it aims to localise each piece of information so that users are able to find it more quickly. The idea is to insert a bookmark at the beginning of each part (real text), and then to build a local link with the title, which points to that anchor. It is also possible defining those links in hidden way, so that nothing changes on the screen. Thus, when users move link by link

through ‘Tab’ key, the focus moves among the titles of blocks, and by activing a link, the screen reader begins to read the relating text, that is from the bookmark.

- *To be applied*

```
<A HREF="#BOOKMARK"> TITLE OF BLOCK </A> <BR>
<A NAME="BOOKMARK"> </A>
```

In order to define non-visible links, the ‘title of block’ has to be referred to TITLE attribute and not enclosed between <A> and </A> tags.

```
<A HREF="#BOOKMARK" TITLE="TITLE OF BLOCK"> </A> <BR>
<A NAME="BOOKMARK"> </A>
```

- *To be checked*

In order to understand if this checkpoint has been applied, independently of type of links (visible or not), the presence of <A HREF=""> and <A NAME=""> consecutive tags can be checked. Furthermore, it is necessary to verify that the HREF and NAME attributes, corresponding to two consecutive tags, have the same content.

2.8.2.b Indexing of content: Index at the top of page

This method can be applied both to information blocks belonging to an individual Web page (e.g., a list of items grouped by letter), and to information spread in more pages (e.g., chapters and sections). This index (or menu) should be put at the head of the page, so that it is immediately available. In this case we suggest using a compacted index that it takes not much space; e.g., using links referred to letters(if information are grouped by letter) would take 26 links. Whereas if a combobox is used, it needs less space. Furthermore, the screen reader saves time reading a combobox rather then several links, especially when users skim the page line by line through arrow keys. Besides an index combobox allows to do a choice by first letter of the wanted item. So, we suggest using <SELECT> element and to relate it by <LABEL FOR> tag to a short description (written before it) of that menu. In order to execute an action as soon as a choice is done, a ‘Go’ button which runs a JavaScript is necessary. Note that also in this case (as 2.6.3.a) ONCHANGE event should not be used.

- *To be applied*

```
<LABEL FOR=”INDEX”> SHORTDESCRIPTION </LABEL>
<SELECT ID="INDEX" >
<OPTION VALUE=”#BOOKMARK”> OPT1DESCRIPTION </OPTION>
<OPTION VALUE=”PAGEADDRESS”> OPT2DESCRIPTION </OPTION>
…
</SELECT>
<BUTTON VALUE="GO"
ONCLICK="JAVASCRIPT:WINDOW.LOCATION=INDEX.VALUE"> </BUTTON>
```

- *To be checked*

In order to evaluate if the developer used a menu-index, we can look for a combobox having the ID setted to ‘index’ or ‘menu’ value, and an associated ‘go’ button. To exclude <SELECT> ELEMENTS belonging to objects of a form, a precondition should be that the <SELECT> and <BUTTON> are both not enclosed between <FORM> and </FORM> tags. Moreover, the button action should be ONCLICK="JAVASCRIPT:WINDOW.LOCATION=INDEX.VALUE" and its VALUE ATTRIBUTE equal to ‘go’.

2.8.3.b Indexing of content: Index frame

As discussed in1.1.3.b, frames can be useful to understand the partition of the page content, provided that each frame has a significant name. Since an index (or menu) frame has a particular importance, we mention about it at this point.

- *To be applied*

```
<FRAME SRC="" NAME="INDEX" TITLE="INDEX">
```

- *To be checked*

Looking for a frame having NAME and TITLE attribute values equal to ‘index’.

2.9.b Navigation links

In order to reach more easily some location of the page (or of the site) we can insert local navigation links, referring to bookmarks in the ambit of the page (e.g., 'skip to content', 'go to top', 'go to navigation bar', etc.). Links of this kind are useful to navigation by screen reader, but, especially, by browsers which have not specific movement commands. So the suggestions are to add to Web page such some links and, if developer preferes, make them hidden (i.e., only readable by screen readers and non-visible). Furthermore, all advisable and useful contents of navigation links can be stored in a specific dictionary file to use during the evaluation phase. Since few terms can be defined directly in the program code, they can be stored in an exsternal file; that can be useful also to differentiate them by different languages.

2.9.1.b Navigation links: Skip to content

When a page has many elements before the content, such as links, advertisement frames etc., using a screen reader it is not easy to reach the current content of the page. So, adding a specific link (at the top of the page) that allows to skip directly to page content, is a possible means to navigate more quickly. Therefore a simple local link pointing to an page anchor can be used.

- *To be applied*

As first link (in case non-visible) of the page

```
<A HREF="#CONTENT" TITLE="SKIP TO CONTENT"> </A>
…
<A NAME="CONTENT"> </A>
```

- *To be checked*

In order to verify if this checkpoint has been applied, the presence of ‘skip to content’ link can be checked: searching a link having as title or text like ‘skip to content’ and pointing to an ‘content’ anchor. Then, also an anchor ‘content’ has to be searched. Both conditions are necessary in order to checkpoint is verified.

2.9.2.b Navigation links: Go to top of page

A specific link to top of the page is particularly useful in those pages having a certain size. So, a link ‘go to the top of the page’ (or similar) should be added to the end of pages. However, more such links can be added in various page places, especially at the end of distinct sections contained in the page. Possible descriptions to use for these links can be stored in a specific file.

- *To be applied*

Using a link like

```
<A HREF="#..."> TOP OF PAGE </A>
```

and defining an anchor at the top of the page

```
<A NAME="TOP"> </A>
```

- *To be checked*

Checking if there is at least one page local link having a text like ‘Top of page’ or similar.

2.9.3.b Navigation links: Go to specific section

Apart from ‘Skip to content’ and ‘Top of page’ links, other page local links can be useful to navigate easier and more quickly among page sections. Furthermore also when a new page is loaded, moving the focus on a specific area is a valid way to reach quickly a precise area. Therefore, the use of links to local (and non) anchors is suggested.

- *To be applied*

Local sections

```
<A HREF="#ANCHORTEXT"> </A>
```

External sections

```
<A HREF="EXTERNALPAGE#ANCHORTEXT"> </A>
```

- *To be checked*

In order to verify if links to precise page sections have been used, the use of '#AnchorText' in <A> tags can be checked (checking if they are local or external anchors is not important).

2.10.b Identifying the main page content

One of the most important problems encountered by blind users, owing to lack of page overview, is to be able to identify the new content of the page. In fact, often Web pages have many information, objects, elements and so on, which stay static, and only a part of the page changes. This changed part can be called as 'new page content' or 'main page content'. So, blind users could not be able to understand the 'current page content' because the screen reader deals with the page in sequential way, and hence too many data (sometimes not well organized) could create confusion and make not clear the content. For this reason, we suggest some possible strategies to use in order to improve that issue. Note that in addition to suggestions described later on, a specific link 'skip to content' can be used (See 2.9.1.b).

2.10.1.b Identifying the current page content: Positioning at page loading

A useful way to allow users to locate the new content of a Web page could be moving the focus at the beginning of the main page content when the page is loaded. Thus, in this way users are able to identify clearly the beginning of main content of the current page and they can start from there to read the page.

- *To be applied*

First of all, it needs to add the anchor 'content' at the beginning of current page content:

```
<A NAME="CONTENT"> </A>
```

Then, there are two possibilities to moving focus at new content when a page is loaded.

By adding the call to '#content' to Web page addresses as follows:

```
<A HREF="PAGE.HTM#CONTENT"> … </A>
```

Ora JavaScript can be used in the <body> tag:

```
<BODY ONLOAD="JAVASCRIPT:WINDOW.LOCATION='#CONTENT'">
```

- *To be checked*

First of all it needs to check the presence of an anchor named ‘content’ or ‘page content’, since it is the presupposition for this checkpoint. Then it can be verified if each page has in the <BODY> tag a JavaScript pointing to ‘#content’ anchor. On the contrary, verifying if the page content is “focalised” by using a page address containing ‘#content’ (e.g., page.htm#content) is a little more complicated process. A possible solution could be checking if for each page there exists almost a link pointing to that one, having at the end ‘#content’ code.

2.10.2.b Identifying the current page content: Using heading level

This checkpoint suggests applying a some heading level <HI> to first line of current page content (that usually is a title or subtitle). In this way, users, during the reading of the page, guess that when they hear ‘heading level i’ (with i=1..6) followed by a certain information, that means it is the new page content.

- *To be applied*

First of all, an anchor ‘content’ should be added to the beginning of main content. Then the following text (corresponding about to line length) should be enclosed in a <HI> tag.

```
<A NAME="CONTENT"> </A> <HI> … </HI>
```

Note that heading level should be applied to only one line, and not to a larger block, otherwise it could become a bothering element rather than a useful one.

- *To be checked*

In order to evaluate if developer has applied this checkpoint to get attention about the beginning of main content, the use of a ‘content’ anchor immediately followed by a some heading level can be checked. Furthermore verifying that heading level is applied to a non-big block (e.g., not more 15/20 words) is suggested.

2.10.3.b Identifying the current page content: Using frames

This checkpoint suggests to use a specific frame named ‘content’ when the Web site is structured by frameset. Thus users, reading in the frame list (produced by special

command of the screen reader) a frame named ‘content’, are able to identify immediately where the main content is visualised. However, we suggest applying (if necessary) also other checkpoints in order to highlight the main content being in the Web page loaded in the frame ‘content’.

- *To be applied*

The code segment to apply is simply

```
<FRAME SRC="" NAME="CONTENT" TITLE="CONTENT">
```

- *To be checked*

In order to evaluate the application of this checkpoint, it needs to verify, in case the Web site use <frameset> tag, if there is a frame having both name and title attributes set to ‘content’ value.

3. Satisfaction checkpoints

3.1.b Addition of short sounds

Associating a short sound to different elements and in different multimedia environments can make the user more "satisfied", e.g., associating each page with a short sound indicating when the loading of the page is completed, so sparing him the need to repeatedly check the state bar. Associating different sounds to different links makes it easier to identify the link type during the skimming. A similar case occurs for different format types. These checkpoints suggest associating different sounds to different cases, but they do not indicate which audio kinds should be used.

3.1.1.b Addition of short sounds: Page loading

Adding a short sound to a Web page can be useful to understand when that page is loaded. In this way blind users do not need to read the status bar. It is not indispensable that the sound is short, but it is advisable because otherwise it should be annoying. We suggest to use longer sound just in the home page.

- *To be applied*

```
<HEAD>
<BGSOUND SRC=" " LOOP="0">
…
</HEAD>
```

where SRC content should be a short audio file.

- *To be checked*

Checking if in <HEAD> section there is the tag <BGSOUND>. To say that this criterion has been applied, it is necessary that all page (or almost) have this tag.

3.1.2.b Addition of short sounds: Different link types

In order to inform users about the type of a link, it could be useful to use a different sound for different types. Following table shows link types we consider.

Type	Description	Example
Internal	A link which points to an anchor of the current page	<a href=”#AnchorText”>
Local	A link which points to a page which is local to Web site	<a href=”LocalPath/PageName.htm”>
Global		
Mail		
Script	A link which actives a JavaScript	<a href=”JavaScript:script()”>

Table 5.4 - Types of links

Therefore, a different sound should be used for each of these types. Unfortunately in order to assign a sound to a link, a JavaScript is necessary. This means that for those browsers which do not support scripts, are not able to perform the sound.

- *To be applied*

First of all a specific JavaScript is added to <HEAD> section. Then the JavaScript function call is associated to each <A> tag by ONFOCUS attribute. The parameter *n* (0, 1, 2..) indicates which audio file should be played according to <EMBED> tags.

```
<HEAD>
<SCRIPT LANGUAGE="JAVASCRIPT">
FUNCTION PLAYSOUND(SOUND)
   {
   DOCUMENT.EMBEDS[SOUND].PLAY();
   }
</SCRIPT>
</HEAD>
<BODY>
<A HREF="" ONFOCUS"PLAYSOUND(N)"> </A>
…
<EMBED SRC="AUDIOFILENAME" HIDDEN=TRUE AUTOSTART=FALSE>
</EMBED>
</BODY>
```

Note: it is important that `<EMBED>` tags are added at the end of `<BODY>`, because in some cases screen readers could interpret them like links.

- *To be checked*

In order to evaluate if this issue has been used by developers, following things can be checked.

- The presence of a JavaScript function `PLAYSOUND();`
- Command use like `DOCUMENT.EMBEDS[SOUND].PLAY();`
- For <a> tags, the OnFocus attributes call '`PLAYSOUND()`' function;
- The addition of `<EMBED>` tags at the end of the `<BODY>`.

3.1.3.b Addition of short sounds: Different file formats

Usually, when a document is available in different formats (pdf, doc, rtf, zip, exe, etc.), different icons (visual features) are used to distinguish those different types. In order to provide a similar feature to blind users, earcons (aural features) could be associated to each format. Pratically an earcon is an audio file which gives information similar to those provided by icons. By this method users are able to

understand immediately the format when they move document by document through Tab key.

- *To be applied*

The code to use is similar to the previous checkpoint (3.1.2.b), where `PLAYSOUND()` function is applied to link to file with pdf|doc|rtf|txt|zip|exe|gz extensions.

- *To be checked*

In order to verify if this checkpoint has been applied, the `PLAYSOUND()` function in the `<HEAD>` section, and associated to `ONFOCUS ATTRIBUTE` of `<A>` tag related to pdf | doc | rtf | txt | zip | exe | gz file extensions.

3.2.a Colour of text and background

This aspect can make easier the navigation of visually impaired people who, with a particular type of contrast, may feel less tired by navigation. It is therefore advisable to avoid colour combinations giving a poor contrast. Furthermore, changing colours in correspondence to some events, or particular areas, can be a way to get attention. For example, when users with low vision move mouse pointer over the object, should be useful the colour of those object changes, or specific areas, such as navigation bar, have different colour combinations.

3.2.1.a Colour of text and background: By passing mouse

This checkpoint suggests changing colour when mouse is over the object, such as links and buttons, that is a similar effect got by focus event. The goal is to point out current object, getting user attention.

We suggest using CSS styles, using external CSS files rather then inserting CSS code in the `<HEAD>` section of page, so that they affect all pages which refer to them.

- *To be applied*

In the CSS file (or `<STYLE>` section)

```
A:HOVER {COLOUR : XX; BACKGROUND-COLOUR : YY;}
```

Where xx and yy are the picked colours.

- *To be checked*

In order to verify if this checkpoint has been applied, the HOVER construct, with at least one of COLOUR or BACKGROUND-COLOUR attributes, is checked. The evaluation is applied to CSS files, if the site uses external CSS files, or else inside of <STYLE TEXT/CSS> section.

3.2.2.a Colour of text and background: Getting focus

This checkpoint suggests changing colour when a link (or button) gets focus. This helps to low vision users to localize the focused elements more quickly, when they navigate through Tab key. Although there exists the possibility to insert in the <BODY> tag the necessary code to give different colours to link states, or using specific JavaScripts, we suggest using CSS style sheets.

- *To be applied*

In the <BODY> tag:

```
<BODY ALINK="XX">
```

Where xx denotes the picked colour.

In the CSS file (or <STYLE> section):

```
A:ACTIVE {COLOUR : XX; BACKGROUND-COLOUR : YY;}
```

Where xx and yy are the picked colours.

- *To be checked*

To evaluate if this feature has been applied, the use of ALINK or ACTIVE (with BACKGROUND-COLOUR and/or COLOUR attributes) can be verified. So, the use of alink is verified in the <body>, while a:active construct is checked in the <style type="text/CSS"> section, or, if the CSS features are separate from page code (better), in the CSS files.

3.2.3.a Colour of text and background: Specific areas

Using different colours for different logical areas can facilitate the localization of precise page parts. For instance, using different background and text colours for navigation bar links, for index links, search bar, and so on, are some examples of

such an issue. We think that this feature can be useful especially when users navigate often on the same Web site. In fact, they can learn different colour and individuate more quickly a specific area simply by its colour. However, we suggest not to use too many colours, because that could create confusion. Although there exist several ways to assign colours and to change them, we refer only to colour links and text blocks.

- *To be applied*

To colourate a specific area, we suggest to use <DIV> blocks:

```
<STYLE TYPE="TEXT/CSS">
#BLOCKID {BACKGROUND: XX; COLOUR: YY }
</STYLE>
<BODY>
…
<DIV ID="BLOCKID">
…
</DIV>
</BODY>
```

Defining several <DIV> blocks, different areas with different colours are created. We consider more important the background colour change, because it is more perceptible then text colour by low-sighted people. However, changing background colour, also text colour change could be requested in order to create an appropriate contrast.

To colour a group of consecutive links, it is possible using simply background and colour attributes:

```
<STYLE TYPE="TEXT/CSS">
A.CLASSNAME {COLOUR: XX; BACKGROUND-COLOUR: YY }
</STYLE>
<BODY>
<A HREF="" CLASS="CLASSNAME"> </A>
</BODY>
```

Assigning different colours to sequences of links, link bars having different colour are visible.

- *To be checked*

A way to verify if different colours for different groups of links or areas have been applied, is to check if there exist classes or `<DIV>` id having different couples of background and/or text colours.

3.3.a Magnifying at passing by mouse

The use of this feature can help people with a good visual residue to better focus to the pointed object. The idea is to enlarge particular elements such as images, navigation links and buttons, not all text. We suggest magnifying the elements only when the mouse pointer goes over them, because this feature can be useful especially for users who have a visual residue and therefore who can use the mouse. This functionality could be applied to images and links having a certain importance or meaning, such as navigation bar links, or images of specific products.

3.3.1.a Magnifying at passing by mouse: Links

A useful tool to aid low-sight people in the exploring of available links of a Web page is enlarging their text. In this way the mouse pointer works as magnifying glass, which focuses the text of pointed link. To do this effect, simply style sheet properties can be used.

- *To be applied*

```
<HEAD>
<STYLE TYPE="TEXT/CSS">
A:HOVER {FONT-SIZE: XX%}
</STYLE>
</HEAD>
```

Where xx indicates the absolute or relative increment of size enlargement.

- *To be checked*

In the CSS file, or in the <STYLE> section of page heading, checking if there is a line like A: HOVER containing, among others, the property FONT-SIZE. Then, comparison with other FONT-SIZE values associated to A: style property can be necessary in order to understand if the change consists of a magnifying.

3.3.2.a Magnifying at passing by mouse: Images

In some cases, enlarging a picture when the mouse is over can be a valid tool to focus on a certain object. For instance, this functionality could be used in a e-commerce Web site, or in a site where a set of particular objects are collected. We believe that not all images of a Web site should be magnified, but only those have a certain meaning.

- *To be applied*

Unfortunately, to magnify a picture two JavaScript functions are necessary.

```
<HEAD>
<SCRIPT LANGUAGE="JAVASCRIPT">
FUNCTION EXPAND()
   {
   DOCUMENT.IMG1.WIDTH=DOCUMENT.IMG1.WIDTH*X
   DOCUMENT.IMG1.HEIGHT=DOCUMENT.IMG1.HEIGHT*X
   }
FUNCTION REDUCE()
   {
   DOCUMENT.IMG1.WIDTH=DOCUMENT.IMG1.WIDTH/X
   DOCUMENT.IMG1.HEIGHT=DOCUMENT.IMG1.HEIGHT/X
   }
</SCRIPT>
</HEAD>
<BODY>
<IMG SRC="" ONMOUSEOVER="EXPAND()" ONMOUSEOUT="REDUCE()">
</BODY>
```

- *To be checked*

Checking if an image is enlarged when the mouse passes over it, is not so easy, because a JavaScript function is necessary and its name can differ according to developer's choices. However, a possible method is checking if the <IMG> tags contain ONMOUSEOVER and ONMOUSEOUT events, if they call two functions, and finally checking if those functions deal with WIDTH and HEIGHT properties.

3.4.b Page information

Some information such as the page pathway, last update of the page, and so on, can give additional useful data to users. The first and last line of the page could be used for this purpose. For example, if the first line contains the pathway of the same page, users know it as soon as the page is loaded. Furthermore, also the last line is almost important when navigating through screen reader. In fact, if that line has the same content in all the pages, users can recognise it easier.

3.4.1.b Page information: Page pathway

Providing the pathway of a page can help users to know where that page is located in the Web site. This can be useful especially when the page is called from a link outside to the Web site. This information could be added to the title of the page, or inserted in the page content (e.g., by using a hidden label).

- *To be applied*

Adding to <TITLE> tag a pathway like

```
<TITLE>
   HOME::SECTIONNAME::SUBSECTIONNAME::PAGE TITLE </TITLE>
```

Or, by using an hidden label

```
<IMG SRC=" "
ALT="HOME::SECTIONNAME::SUBSECTIONNAME::PAGETITLE">
```

- *To be checked*

Checking if there exists a sort of patway in the <TITLE>, in the page content, or in an hidden label. Looking for a string like “Home::…” or “Home>>…” could be a way to evaluate this checkpoint.

3.4.2.b Page information: Repeated end page line

When a blind user reads in sequential way a Web page by the screen reader, having the same text for last line of every page is advisable. In fact this allows users to identify easier the end of the page.

- *To be applied*

The content to be used could be the last update of the page; the address of the current page, e.g. “the address of this page is…”; or simply a string (in case hidden) like “Page end”.

- *To be checked*

In order to evaluate this checkpoint, verifying that all the page have the same content is sufficient.

5.3 Application Conditions

In the previous paragraph a checkpoint set for implementing the criteria has been proposed. For the most criteria more than one checkpoint exists. So now it is necessary to state when a criterion can be considered applied or not; i.e. we have to define a relationship among the checkpoints necessary to implement the same criterion.

For instance, the criterion 1.1.b has three checkpoints (i.e. grouping by headings, marking blocks, and grouping by frames). Well, this criterion is applied if almost all its checkpoints are used. However, the criterion 3.1.b has three checkpoints (i.e. page loading, different type links, and different file formats), all of which should be applied in order to satisfy the criterion.

Next table summarises for each criterion the Boolean relation existing among its checkpoints.

Criteria	Description of Criteria	Checkpoints for criterion
1.1.b	Logical partition of interface elements	OR
1.2.a	Proper link content	AND
1.3.b	Messages and dynamic data management	OR
1.4.a	Proper style sheets	AND
1.5.b	Layout and Terminological Consistency	AND
2.1.b	Number of links and frames	AND
2.2.b	Proper name for frames, tables and images	AND
2.3.a	Location of the navigation bar	OR
2.4.b	Importance levels of elements	AND
2.5.b	Keyboard shortcuts	OR
2.6.a	Proper form layout	AND
2.7.b	Specific sections	AND
2.8.b	Indexing of contents	OR
2.9.b	Navigation links	AND
2.10.b	Identifying the main page content	OR
3.1.b	Addition of short sounds	AND
3.2.a	Colour of text and background	AND
3.3.a	Magnifying at passing by mouse	AND
3.4.a	Page information	AND

Table 5.5 - Boolean relation among checkpoints for each criterion

6
Criteria Formalization

6.1 Introduction

In this chapter we introduce a brief formalization to better define some concepts about the criteria application status. As seen in chapter 3, criteria are general principles expressing the usability of accessibility issues. Then, in chapter 5 a list of checkpoints has been proposed in order to show how such criteria can be implemented. As already explained, a checkpoint is a specific code construct that can be used in order to check or apply the relating criterion: for a given criterion, one or more checkpoints are needed. The application status of criteria depends on the application status of necessary checkpoints. Hence, it is important to define whether a criterion can be considered “Applied” “Not applied”, or “To be reviewed”. For this reason, in this chapter we introduce a quality model based on two functions and some definitions of criteria application status. In order to obtain the quality model, we considered the Goal, Question, Metrics (GQM) approach outlined first in Basili and Weiss [1984] and then often adopted in software engineering investigations. The GQM approach can be followed on any analysis that requires data collection. The GQM approach applied to Web sites is described in [Brajnik 2002] and the steps that should be followed are also indicated: (1) establish the goals of the analysis (e.g. detecting and removing usability obstacles, comparing two designs of a site to determine which one is more usable); (2) develop questions of interest whose answers would be sufficient to decide on the appropriate course of actions; (3) establish measurement methods (i.e. metrics); (4) provide a description of how data are identified; (5) give a description of how data are going to be categorized. Thus, the quality model describes which properties are important for given goals, and how these properties can be traced back to simpler attributes that can be measured. The model also prescribes how measurements have to be taken. Furthermore, automatic

Web testing tools can play an important role in the usage of quality models, because they necessarily adopt objective metrics only, are systematic and error free, and are much more cost-effective than any other manual method. As pointed out in [Brajnik 2001] the issue of the validity of metrics adopted in a quality model arises when metrics are computed by automated tools and when metrics start dealing with more interesting properties, like assessing accessibility or usability. In general, only relatively simple quality models are based entirely on automatic tools. In the vast majority of the cases, quality assessment is based also on human inspection and judgment. The quality model proposed in this chapter basically consists of a polynomial expression of criteria with weights base on a Boolean expression among checkpoints.

6.2 The Criteria and Checkpoints Sets

Let define now the sets which the evaluation and repairing functions work on. As mentioned in previous chapters, a criterion expresses a general usability principle of accessibility to be practically implemented by using the technical checkpoints. So, we refer to given criteria to express the principle we would like to apply or to check, but the evaluation and repairing functions work on the checkpoint set. Furthermore, the proposed functions handle the Web pages, in order to verify or fix a specific criterion, according to the corresponding checkpoints.
The needed sets are defined below.

Criteria set

Let CR be the set of effectiveness, efficiency, and satisfaction criteria. Given N_1=5, N_2=10, and N_3=4, we define CR as the set of all the criteria:

$$CR = \cup_{i=1..3} CR_i \text{ where}$$

$$CR_i = \{C_{i.j}\} \text{ with } i=1..3, \text{ and } j=1..N_i$$

are the effectiveness, efficiency, and satisfaction criteria subsets.

So, according to the above definition we simply have the three following subsets:

Effectiveness, efficiency, and satisfaction criteria subsets

Let $CR_1 = \{C_{1.j}\}$ with $j=1..N_1$ be the effectiveness criteria subset,
let $CR_2 = \{C_{2.j}\}$ with $j=1..N_2$ be the efficiency criteria subset, and
let $CR_3 = \{C_{3.j}\}$ with $j=1..N_3$ be the satisfaction criteria subset,
such as $CR = CR_1 \cup CR_2 \cup CR_3$

Now, On the basis of effectiveness, efficiency, and satisfaction criteria we give the definition of checkpoint set.

Checkpoint set

Let CP be the set of checkpoints necessary to implement the criteria belonging to CR. Given $N_1=5$, $N_2=10$, and $N_3=4$; and for each <i,j> let N(i,j) be the number of the checkpoints associated to $C_{i.j} \in CR$; we define:

$CP = \{C_{i.j.k}\}$ with $i=1..3$, $j=1..N_i$, $k=1..N(i,j)$,
where N is the $\#\{C_{i.j.k}\}$

So, generally speaking, we denote with $C_{i.j} \in CR$ (with $i=1..3$ and $j=1..N_i$) a generic criterion, and with $\{C_{i.j.k}\} \subset CP$ (with $k=1..N(i.j)$ the set of checkpoints necessary to implement the same criterion.

6.3 Evaluation and Repairing Functions

Now we wish to introduce two functions that operate on pages and on criteria. As already said, our purpose is to use the proposed criteria both for evaluating a Web site, and for aiding developers into repairing the pages. Each criterion can be complied with applying one or more checkpoints (i.e. it depends on kind of criterion). Therefore, our functions work on the basis of those checkpoints.
We define the functions named *Eval* [ ] and *Repair* [ ] that express the evaluation and repair process. Let's define them formally.

6.3.1 Evaluation Function

Eval is a function that takes one page, and returns the check results computed by using our proposed checkpoints.

We say that *Eval* function verifies a criterion $C_{i.j}$ to indicate that the function checks whether the page meets that criterion or not. In order to do that, *Eval* works on the basis of the $\{C_{i.j.k}\}$ checkpoints, with k=1..N(I,j). We use {true} when the $C_{i.j.k}$ checkpoint is "applied", and {false} when it is "not applied".

***Eval* function definition**

Let is P a Web page, $C_{i.j} \in$ CR one of effectiveness, efficiency or satisfaction criteria, and $\{C_{i.j.k}\} \subset$ CP (with k=1..N(i,j)) the checkpoint set necessary to implement the criterion. We define the *Eval* function as follows:

$$Eval\ [P, C_{i.j}] = BC_{\{Ci.j.k\}}$$

$$= \{\text{"Applied"} \mid \text{"Not applied"} \mid \text{"To be reviewed"}\}$$

where $BC_{\{Ci.j.k\}}$ is a logical proposition defined on the basis of the checkpoint set.

So, the function result is the application status of criterion $C_{i.j}$ to page P, that can be "Applied", "Not applied" and "To be reviewed", according $\{C_{i.j.k}\}$ (with k=1..N(i,j)) set of checkpoints associated to $C_{i.j}$ criterion.

In practice, the *Eval* function checks each $C_{i.j.k}$ checkpoint, and then it combines the results on the basis of a logical conditions relating the checkpoints, in order to satisfy the criterion. In fact, as we mentioned in chapter 5, some criteria are satisfied if there exists at least one applied checkpoint (OR condition), and others if more then one simultaneously applied checkpoints (AND condition). If no condition is satisfied, or only a part of -the conditions are, the criterion is considered "Not applied" or "To be reviewed". We can formalize these criteria application status through the next three definitions.

"Applied" criterion

Let is P a Web page, $C_{i.j} \in$ CR one of effectiveness, efficiency or satisfaction criteria, and $\{C_{i.j.k}\} \subset$ CP (with k=1..N(i,j)) the checkpoint set necessary to implement the criterion.

We say that P satisfies $C_{i.j}$ (i.e. the criterion is applied), if there exist the necessary $C_{i.j.k}$ checkpoints such as the Boolean condition $BC_{\{Ci.j.k\}}$ is {true}:

$$\exists\ k : BC_{\{Ci.j.k\}} = \{true\}$$

$$Eval\ [P, C_{i.j}] = \text{“Applied”}$$

“Not applied” criterion

Let is P a Web page, $C_{i.j} \in$ CR one of effectiveness, efficiency or satisfaction criteria, and $\{C_{i.j.k}\} \subset$ CP (with k=1..N) the checkpoint set necessary to implement the criterion.

We say that P does not satisfy $C_{i.j}$ criterion (i.e., the criterion is not applied), if all necessary $C_{i.j.k}$ checkpoints are non-applied (false). That means

$$\forall\ k=1..N: C_{i.j.k} = \{false\} \rightarrow BC_{\{Ci.j.k\}} = \{false\}$$

$$Eval\ [P, C_{i.j}] = \text{“Not applied”}$$

“To be reviewed” criterion

Let is P a Web page, $C_{i.j} \in$ CR one of effectiveness, efficiency or satisfaction criteria, and $\{C_{i.j.k}\} \subset$ CP (with k=1..N) the checkpoint set necessary to implement the criterion.

We say that P does not completely satisfy $C_{i.j}$ criterion (i.e., the criterion is to be reviewed), if not all necessary $C_{i.j.k}$ checkpoints are non-applied (false), but there exists at least an applied one.

That means

$$\exists\ h \mid C_{i.j.h} = \{true\},\ BC_{\{Ci.j.k\}} = \{false\}$$

$$Eval\ [P, C_{i.j}] = \text{“To be reviewed”}$$

6.3.2 Repairing Function

Let consider now the other function *Repair* [ ]. As already mentioned, a repairing function should assist the developer / evaluator in changing the page code more automatically possible. This phase is more difficult then the other, but it is important to fix the problems in order to improve the usability of accessibility. In fact, this activity can require much effort and a lot of time.

So, the considered function takes a Web page and the criterion to apply, and returns the changed page. Since a criterion can be applied according to one or more checkpoints, and some equivalent choices can be available, in some cases the developer is involved in the checkpoint selection. For instance, in order to apply the criterion 'logical grouping of content', the developer could decide whether to mark blocks with hidden labels, or to use heading levels. Therefore the *Repair* function has some pieces of input data as parameters. The result of this function is another Web page complying with criterion that has just been fixed or applied.

The Repair function can be expressed as follows:

Repair function definition

Let is $C_{i.j} \in CR$ one of effectiveness, efficiency or satisfaction criterion; let is P a Web page such as *Eval* [P, $C_{i.j}$] = {"Not applied" | "To be reviewed"}, i.e. P does not partially or completely met the criterion. Let are {$C_{i.j.k}$, $C_{i.j.n}$,… $C_{i.j.q}$} (for some k, n, q) the set of checkpoints by which *Eval* [P, $C_{i.j}$] = "Applied".

The *Repair* function is such as

$$P \xrightarrow{C_{i.j.k}} P_0 \xrightarrow{C_{i.j.n}} P_1 \xrightarrow{C_{i.j.q}} P'$$

$$Repair\,[\,P, C_{i.j}\,] \xrightarrow{*} P' \mid Eval\,[\,P', C_{i.j}\,] = \text{"Applied"}$$

7

User Testing: Evaluation of Usability Criteria Based on a Web Site Prototype

7.1 Introduction

As already mentioned in chapter 2, usability testing with real participants is a fundamental method for evaluating usability [Nielsen 1993; Shneiderman 1998]. It provides an evaluator with direct information about how people use computers and their problems with the tested interface being. During the usability testing, participants use the system or a prototype to complete a pre-determined set of tasks while the tester records (e.g. by video types or by log files) the results of participants' work.

So, a usability testing is a black-box testing, i.e. the behaviour of Web site is directly used and observed by users or evaluators, without considering the code of Web pages.

The best results come from testing no more than five users and running as many small tests as it can afford, as mentioned by [Nielsen 2000B]. In testing multiple groups of disparate users, it does not need to include as many members of each group as in a single test of a single group of users. Three-four users from each category are recommended if testing two groups of users [Nielsen 2000B]. In our testing two groups of people were involved: blind and visual impaired persons.

In this chapter we report results obtained from the user testing we conducted with blind and visual impaired people. Our studies are a preliminary step in usability testing, according to our proposed criteria. More refinements of Web site prototypes, test organisation (e.g., choice of tasks), and used tools are certainly required. However, these empirical results provide a first feedback on the impact of the criteria application on end users. Non-parametric statistic tests were used on gathered data,

principally due to the small size of the samples considered [Seagle 1956]. Our tests revealed that, when our criteria are applied, the navigability on the Web is improved, since the time spent searching for wanted information or performing a task is significantly reduced.

In the following paragraph the Web site prototype used for the test is described; next, testing method, data collection and analysis are discussed.

7.2 Web Site Prototype

For our testing, we built a Web site prototype based on the proposed usability criteria for interacting through screen reader. The Web site contains specific information about the Tuscany Blind association (Unione Italiana Ciechi – Regione Toscana). This site was chosen with the intent of putting blind people in a comfortable situation, by providing them with familiar information, thus reducing navigation difficulties. In this way, users could concentrate on the test accomplishment.

We developed two identical Web site prototypes, except one contained many of our proposed criteria and one did not. The "naked" prototype (without any criterion) was used as negative control in our testing protocol. The time required for performing the same kind of tasks in both cases was recorded.

The two Web sites have three main sections, i.e. "News", "Documents&Download", and "Organization"; all of them are reachable from each page, each section has subsections, and so on.

The general layout of the page includes:

- a navigation bar at the top;
- a submenu or local index at the left;
- the current page content at the right (it is the greatest area);
- navigation links ("Go back", "Go to navigation bar", etc.) at the bottom.

Criteria applied to one of the two prototypes are listed below

- *Logical partition*

  Heading levels are mainly used to logical partitioning the information; however, in some cases hidden labels or tables with proper summary values are used. For instance, the navigation bar and submenus are marked with

appropriate labels (e.g. “navigation bar:” and “submenu:“). Then, in the pages containing various file information and links related to downloadable manuals or programs, each data group is placed in a specific table with group names as summary attributes.

- *Main page content identification*

  Three ways are used to recognise the main content of the current page. First of all the “skip to content” link is added to each page; then, the first line of the main page content is enclosed between `<H1>` and `</H1>` tags; lastly, when a page is loaded, the focus moves directly to the current page content (i.e. the calling link contains the “#content” bookmark).

- *Different visit order of links*

  In every page, except the home page, the navigation bar links have the lowest visit order values (i.e. the greatest tabindex values),whereas the submenu and other links have a greater priority. In this way users visit links associated to the submenu or to more recent information first, and the navigation bar links at the end. In addition, shortcuts are associated to navigation links.

- *Additional information*

  Each page title, in addition to the real title of the page, contains the page pathway. For example, the title of the page which notes and manuals can be downloaded from, is “Home :: Documents&Download :: Program downloading”. Furthermore, each page has the “last updates information” as last page line.

- *Sound addition*

  For different links (i.e. local, internal, and external), we applied different sounds (see 4.5.2 for more details).

- *Visual features*

  We used different colours for distinguishing the navigation bar, submenus, and the current link pointed by mouse. The navigation bar is located horizontally at the top of the page in a blue area; the submenu links are placed vertically at the left of the page and are separated by the page content by a vertical black line. When the mouse hovers the links, their dimensions and colours change.

An example of a Web page of the described Web site prototype is reported in the Figure 7.1. The navigation bar, the submenu, and the pointed link magnification are shown.

Figure 7.1 - A Web page of the Web site prototype used for the user testing

7.3 Testing

7.3.1 Method

Fifteen blind users were recruited to participate to the testing. All of them used a screen reader on a regular basis and were familiar with Windows and the Internet Explorer browser, which means that they had been using Windows 98/ME and Jaws, as screen reading application, for at least one year prior to the testing. Nine of them are totally blind, while six have a vision deficit (they cannot spot elements on the screen without an auxiliary support). The experience with the screen reader was

extremely different among the participants, (their level ranged from beginner to expert), so they were provided with a summarising list of the most important Jaws commands. Furthermore, before the beginning of the test, the users were allowed to explore both Web site versions (with and without our criteria). This avoided:

- bias due to excessive discrepancy in navigation abilities among testing users, since all the participants started the test with the same basic skills;
- bias due to the navigation experience gained by each user when doing the first test: such increased ability might affect the results of the second test. By allowing a preliminary navigation on both Web sites, the starting point was the same for each test performed by the same user.

The testing aims at collecting two kinds of data:

- *Objective ones* – Time spent by users carrying out the assigned tasks;
- *Subjective ones* – Users' preferences, opinions, and suggestions.

Our testing procedure is based on two remote evaluation techniques: task-based testing complemented with a remote questionnaire. Remote evaluation aims at giving such a support when evaluators and users are widely separated in time and/or space.
The task-based testing is subdivided in two sessions. The first one, named "*test0*", is conducted on the "naked" prototype.; the second one, named "*test1*", is carried out on the prototype built according to our criteria. The two tests were located online, so that users were able to connect to them from their own computers. User's interactions were automatically collected during the test running.
The complemented remote questionnaire aims at evaluating qualitative aspects of Web site usability, such as user satisfaction or amusement.

7.3.2 Logging Tool

In order to collect the times spent by users while carrying out the assigned tasks an automatic logging tool was used. Our tool is an adaptation of the logging tool used also for WebRemUsine [Paternò and Paganelli 2001]. Main interaction activities performed by the user during testing phases were captured and logged by that tool.

The tool included JavaScript functions, java applet, and java servlet. The tool component in JavaScript were able to detect (client side) all user interactions with the Web browser. Then, all the captured events were passed to a Java applet (client side). The applet allowed to gather all the interaction activities provided by the JavaScript component; at the end of the testing procedure, the applet sent all data to a servlet that (server side) created a log files with them.
Next figure shows a fragment of a log file.

	Test logging:
	time: Mon May 19 13:09:16 GMT+01:00 2003
	ID 947752C26201F44139152060A449F015
	ADDR 146.48.82.179
	HOST 146.48.82.179
	test name: Cesare
	Microsoft Internet Explorer Mozilla/4.0 (compatible; MSIE 5.5; Windows 98; Win 9x 4.90)
1	*time:1053342638540*
2	***SKIP TO Task1***
3	*time:1053342651780*
4	*LOAD http://giove.cnuce.cnr.it/Webtest/index.html*
5	*time:1053342662600*
6	*CLICK TARGET=link*
7	*HREF=http://giove.cnuce.cnr.it/Webtest/notizie.htm#submenu*
8	*time:1053342662650*
9	*UNLOAD*
10	*time:1053342662820*
11	*LOAD http://giove.cnuce.cnr.it/Webtest/notizie.htm#submenu*
12	*time:1053342666770*
13	*CLICK TARGET=link*

14	*HREF=http://giove.cnuce.cnr.it/Webtest/notizie/2003/maggio.htm#conte*
15	*nt*
16	*time:1053342666880*
17	*UNLOAD*
18	*time:1053342667050*
19	*LOAD*
20	*http://giove.cnuce.cnr.it/Webtest/notizie/2003/maggio.htm#content*
21	
22	*time:1053342685830*
	SKIP TO Task2
198	*time:1053342699340*
199	*UNLOAD*
200	*...*
201	*time:1053343029660*
202	***SKIP TO Task8***
203	*time:1053343029720*
204	*UNLOAD*
	time:1053343033560
	UNLOAD
	STOP

Figure 7.2 - Fragment of a log file generated by the logging tool

As shown in the figure, such log file contains a wide variety of user actions (such as mouse clicks, text typing, link selections…), as well as browsing behaviour such as start and finish of page loading. In particular, the tool logs the time when a specific interaction is captured. Time is expressed in milliseconds since midnight (GTM - Greenwich Mean Time) on 1 January 1970 and this is due to automatic data analysis. Users were then asked to explicitly indicate when they skip to next task by clicking on “Next task” link during the testing procedure. When they changed tasks, the logging tool recorded the task change with “skip to task” together with its instant time in the log file (see Figure 7.2). Therefore, when the analysis was performed, it was possible to individuate the beginning and the ending of each task. In order to

compute the time spent performing each task, the beginning and the ending times were needed. As mentioned above, when the user finished a task, he/she must click on “skip to next task” link for proceeding with the testing. That instant was considered as the ending of the previous task, while the time following “skip to task” was taken as the beginning of the current task.
In the Figure 7.2, the task changes are in bold; in addition, each line is numbered for convenience.
For example, time requested for task1 is got from subtracting line 21 to line 3. This operation was made for each task in order to get a set of task times for each user. More details are reported in next paragraphs.

7.3.3 The Questionnaire

The questionnaire, composed by 18 questions, allowed to collect some information, such as the operating system and assistive technologies used by the participants, and other pieces of information not obtainable by the logging tool. Then, subjective information was considered. For example, users were asked for some opinions about the usefulness of sounds, shortcuts, etc. Through a specific section designed for low vision users, opinions on colour contrast and link magnification were collected. Indications on the difficulty of each task were also obtained. Finally, suggestions and comments were invited.
Participants filled in the questionnaire only after the end of the tests, so they were not influenced in any way.

7.3.4 The Wizard Test

As above mentioned, the testing procedure was conducted in two sessions: “*test0*” (on the “naked” prototype) and “*test1*” (on the prototype provided with usability criteria). Both sessions were automatic and guided the user during the testing procedure.
Participants were asked to carry out a set of seven tasks per test. The tasks included common navigation operations, such as opening a page, reading its content, and searching a specific information. The participants were also required to download files, fill in a form, and so forth. *Test0* and *test1* obviously included the same

typology of tasks, they only differed for the specific file to download, the piece of information to find and so on. Thus, when we say *"test"* we actually refer both to *test0* and *test1*.

Below there is the list of the tasks assigned to the users.

1) Visit the Toscana's bulletin page
(Home::Notizie::Toscana oggi).
This is a reading content page where the information is organised by using 'heading levels'. The user was asked to look for a specific news item.

2) Visit the Web page "Dipartimenti e commissioni"
(Home::Organizzazione::Dipartimenti e commissioni).
This page is a content reading page where 'heading levels' are used to logically group the different pieces of information. A page index of content is composed of a small set of local links.
The user was asked to find the name of the coordinator of a certain association commission.

3) Visit the alphabetical order list of "Rivista suoni"
(Home::Documenti&Download::Rivista Suoni::Elenco alfabetico).
This page is a content reading where both 'heading levels' and an index combobox are used to organize the page information.
The user had to search a specific song title.

4) Downloading a document
(Home:: Irifor::Corso di formazione "Homerus").
This page is a content reading page where a specific link to download the application form is placed.
The user had to search and to download a specific application form.

5) Download a zip file
(Home::Documenti&Download::Download di manuali)
This is a 'downloading page' where the file groups are placed in several tables: each table summary contains the name of file group.
The users had to download a specific file.

6) Open an external URL

(Home:: Organizzazione::Sezioni UIC Toscana)

This is an 'index page' where little information about local divisions of Tuscany's blind association is provided and related Web site links are listed. The users had to search for the telephone number and to open the Web site of a given local blind association of Tuscany.

7) Fill in a form

(Home::Invia un messaggio).

This is a Web page with a form that can be filled in to send comments, suggestions, or questions.

The users had to compile and to send it.

While user carried out the *test*, "Next task" button was available to skip to following task. Each task appeared in a popup menu, where there was the "Continue the test" button to proceed with the test. At the end, by clicking on "Stop" button all user interactions gathered by the logging tool (see 7.3.2) were recorded in a log file.
An example of a popup window associated to a test *task* is shown in Figure 7.3.

Figure 7.3 - Example of popup window containing the description of *task5*

7.4 Results

Once all data had been gathered through two tests carried out from the fifteen users, the analysis of them was considered. In this paragraph some results are reported.
The difference between the time spent performing given tasks in *test0* (without any criteria applied) and in *test1* (with criteria applied) was used to determine the application of our criteria actually caused navigability improvement. In short, the time saved was taken as an indicator of such improvement. To obtain this information, the automatic logging tool recorded each user interaction time, marking the start of every task. For each user, the time required to perform each task in *test0* and *test1 was* calculated in milliseconds and then converted in seconds. Thus, several time sets were obtained and compared.

In order to determine if there are significant differences between time groups associated to *test0* and *test1* and considering the size and the nature of our samples non parametric statistic tests were applied, according to Seagle [1956]; α was fixed at 0.05.We found a significant difference between the total time spent by all users performing each task in *test0* and *test1* (Wilcoxon matched pair test; n=7; z=2.366; p<0,05).
For each task, the total time was calculated by adding up the time spent by each user (from user1 to user15).
We also found a significant difference between the total time spent by each user performing all the given tasks in *test0* and *test1* (Wilcoxon matched pair test n=15; z=3.237; p<0.05). For each user, the total time required in *test0* and *test1* was computed by adding up the time required for each task, from task1 to task7.
From our results, thus, we can reasonably assess that there is a real benefit achieved by the application of our usability criteria.
The difference between *test0* and *test1* in terms of time spent for each task (considering all the users) and of time spent by each user performing all tasks is shown in Figure 7.4 and Figure 7.5.

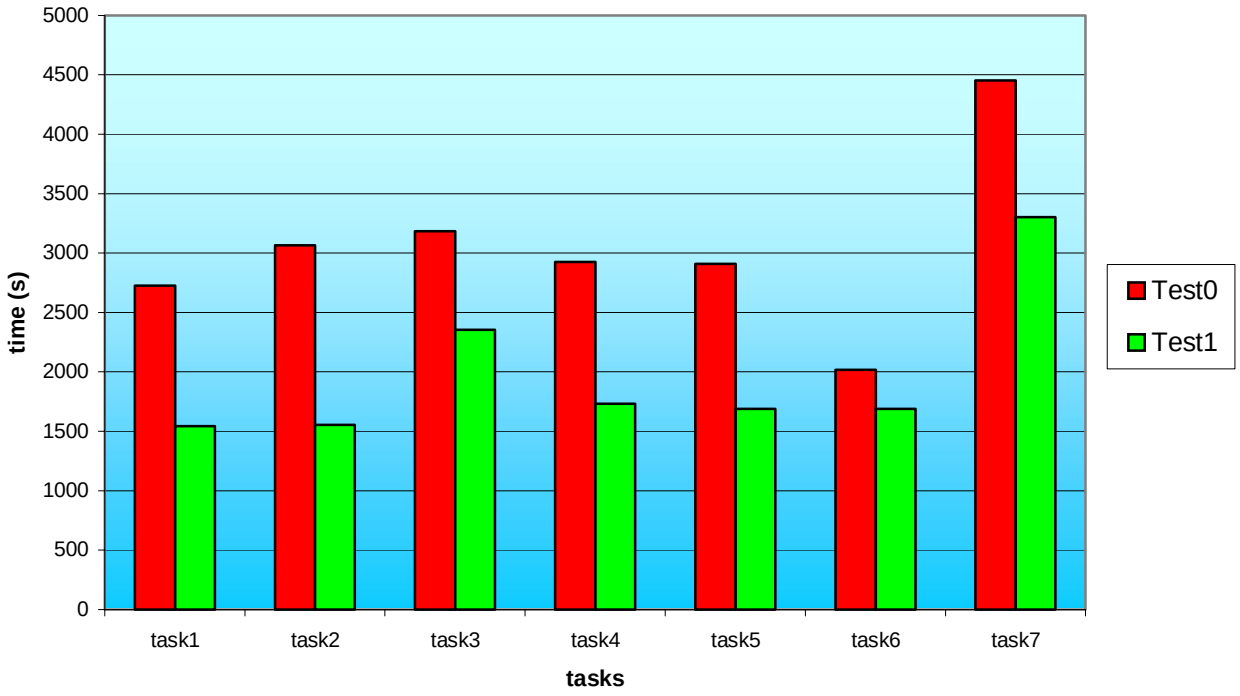

Figure 7.4 – Comparison between test0 and test1 in terms of total time spent performing each task, considering all users (see the text for more details).

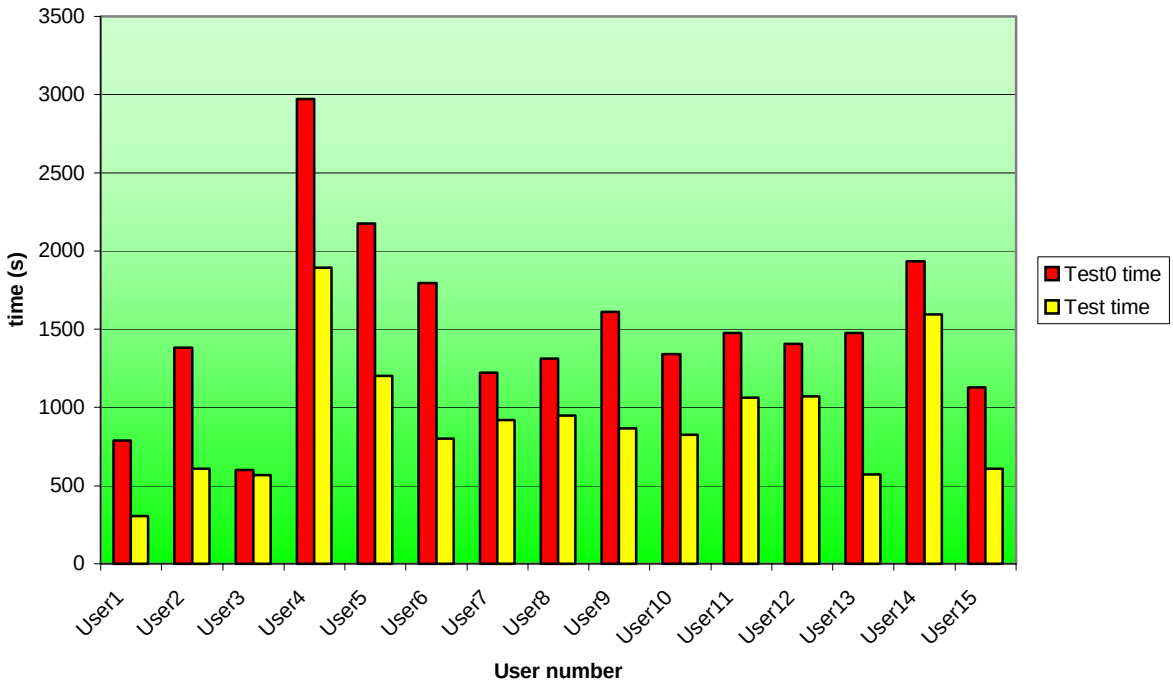

Figure 7.5 – Comparison between test0 and test1 in terms of total time spent by each user performing all the given tasks (see the text for more details).

In Figure 7.4 we can note that the time spent performing task6 (i.e. searching a specific link) is almost the same for *test0* and *test1*. This is expected, because the two prototypes were developed in a way that the differences between them would not affect the time required for this type of task. I.e. in the criteria-based Web page the "grouping by headings" was used, but in that case was not particularly useful, because the link was searched by tab key and not by the Jaws heading navigation command. Thus, task6 is the less influenced by the application of our criteria. Task2 (i.e. looking for information in a large page), instead, turned out to be the most influenced by our criteria. On the whole, information searching in a long page is the task which shows the major benefits from the proposed criteria. However, also task5

(i.e. a file downloading) and 7 (i.e. filling in a form) present are positively influenced by our criteria.

Next figure shows the time saved both by blind and low vision users in consequence of the application of our criteria. The time saved is expressed as percentage difference between the time spent by each category performing *test0* and *test1*. . Even though both categories definitely benefit from the application of our criteria (saving around 40% of the navigation time), totally blind people experience a major time saving (54% vs 46% of low vision users). This result is consistent with our expectations, since usability criteria are thought both for blind and low vision users, but they are mainly aimed at improving the accessibility through screen readers rather than through magnifying programs.

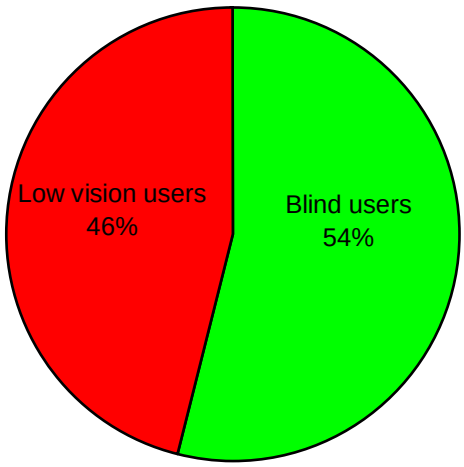

Figure 7.6 - Time saving relating to blind-low vision users' activities

In conclusion, although results presented in this chapter are empirical and preliminary, they certainly reveal that our usability criteria have a positive effect on Web site navigability and bring valuable benefit to blind and low vision users.

8
A Proposed model for Heuristic Evaluation

8.1 Introduction

As already said in chapter 2, usability inspection is the generic name for a set of methods that are all based on allowing evaluators to examine a user interface. Through inspection evaluation, a set of criteria or heuristics is used to identify potential usability problems in an interface. Unlike other UE methods, inspections rely solely on the evaluator's judgment. The heuristic evaluation is the most popular among the usability inspection methods: it is consists in a systematic inspection of a user interface design, for usability.

In this chapter, we propose a heuristic-based model, according to our criteria for the usability of Web sites when screen readers are employed. The model here in proposed basis on the checkpoints set defined in chapter 5. Besides, as described in paragraph 5.3, several logical conditions among the checkpoints were fixed in order to define the application status of criteria. This issue is also conceptualised in chapter 6.

The approach proposed in this chapter represents a pilot heuristic-based evaluation method of the Web site usability defined according to our criteria. In next paragraphs, we are going describe the proposed model, and the results deriving from 16 Web site rating are reported. When we refer to Web site usability, obviously we consider our usability definition of Web site accessibility when interacting through screen readers, which differs from the usability concept used in other contexts.

8.2 The Proposed Model

According to our definition of criteria application status (see 5.3 and 6.3), a criterion is applied if a given Boolean condition on the requested checkpoints is verified. Our

criteria fall into one of these two categories: the “OR” type or the “AND” type, depending on the logical relation (AND/OR) among its checkpoints. An "OR" type criterion is applied if at least one checkpoint is satisfied; an "AND" type criterion is applied if all its checkpoints are used. On the basis of these conditions, we define a measurement model in Web site assessing. In practice, we assign scores to criteria and checkpoints, so that a quantitative measurement of the Web site usability can be achieved. Then, according to such measurement model, various usability levels are determined.

8.2.1 Content-Dependent/Independent Checkpoints

For the implementation of the 19 criteria defined, 54 checkpoints were determined. Some checkpoints should be applied in every page (e.g. “tabbing order position” or “shortcuts”), while others not always are requested (e.g. button labels). We define the first type as *independent* from the page content, and the second as *dependent* from it.
This distinction is fundamental for the evaluation; a “*content-dependent*” criterion, in effect, can be absent simply because it is unnecessary and not because is not applied. For example, let’s consider a page without any kind of form control elements (e.g. buttons, fields, comboboxes, etc.). When we evaluate the criteria application for this page, all checkpoints relating to form control elements (e.g. 1.5.1.b, 1.5.2.b, 2.6.1.a, etc.) are unnecessary, so their application status is “unnecessary” rather than “not applied”.

8.2.2 Scores

In order to define various usability levels, we assign the score to each criterion and to each checkpoint. First of all we fix the total score to 100. According to task accomplishment-based importance levels [see 3.4) the criteria scores are like so shared: effectiveness criteria 50, efficiency 40, and satisfaction 10. Then, for each criteria set, the score is further divided among the number of criteria.
The score of every checkpoint is computed according to the type of its own criterion. Each checkpoint of an "or" criterion is worth the criterion score, considering that to satisfy the criterion at least one checkpoint is sufficient. The score of a checkpoint

associated to an "and" criterion is obtained by dividing the criterion score per the number of its checkpoints. Next table shows whole criteria-checkpoint structure with associated scores.

Criteria number	Description of Criteria	Criteria score	Checkpoint number	Checkpoint Description	Criteria type	Checkpoint Score
1.1.b	Logical partition of interface elements	10	1.1.1.b	Grouping by headings	OR	10
			1.1.2.b	Marking page parts	OR	
			1.1.3.b	Grouping by frames	OR	
1.2.a	Proper link content	10	1.2.1.a	Textual links	AND	3,3
			1.2.2.a	Graphical links	AND	3,3
			1.2.3.a	Graphical and textual links	AND	3,4
1.3.b	Messages and dynamic data management	10	1.3.1.b	Focusing on new messages	OR	10
			1.3.2.b	Using a dialogue window	OR	
			1.3.3.b	Opening a new window	OR	
1.4.a	Proper style sheets	10	1.4.1.a	Voice synthesizer	AND	3,4
			1.4.2.a	Braille display	AND	3,3
			1.4.3.a	Braille printer	AND	3,3
1.5.b	Layout and Terminological Consistency	10	1.5.1.b	Button labels	AND	3,4
			1.5.2.b	Button dimension	AND	3,3
			1.5.3.b	Link to home page	AND	3,3
2.1.b	Number of links and frames	4	2.1.1.b	Number of frames	AND	2
			2.1.2.b	Number of links	AND	2
2.2.b	Proper name for frames, tables and images	4	2.2.1.b	Frames: proper name and title	AND	1
			2.2.2.b	Frames: proper page title	AND	1
			2.2.3.b	Tables: proper summary value	AND	1
			2.2.4.b	Images: proper description	AND	1
2.3.a	Location of the navigation bar	4	2.3.1.a	Using hidden label	OR	4
			2.3.2.a	Using Iframe	OR	
			2.3.3.a	Using frames	OR	
			2.3.4.a	Menus and submenus: using lists	OR	
			2.3.5.a	Popup menu: using disappearing blocks	OR	
2.4.b	Importance levels of elements	4	2.4.1.b	Tabbing order position	OR	4
2.5.b	Keyboard shortcuts	4	2.5.1.b	Main links, buttons and fields	OR	4
			2.5.2.b	New interaction elements	OR	
2.6.a	Proper form layout	4	2.6.1.a	Button labels	AND	1
			2.6.2.a	Groups of control elements	AND	1
			2.6.3.a	Onchange event	AND	1
			2.6.4.a	Matching labels and input elements	AND	1
2.7.b	Specific sections	4	2.7.1.b	Last updates	AND	2
			2.7.2.b	List of assigned keys	AND	2
2.8.b	Indexing of contents	4	2.8.1.b	Highlighting each part	OR	4
			2.8.2.b	Index at the top of page	OR	
			2.8.3.b	Index frame	OR	
2.9.b	Navigation links	4	2.9.1.b	Skip to content	AND	1,4

			2.9.2.b	Go to top of page	AND	1,3
			2.9.3.b	Go to specific section	AND	1,3
2.10.b	Identifying the main page content	4	2.10.1.b	Positioning at page loading	OR	4
			2.10.2.b	Using heading level	OR	
			2.10.3.b	Using frames	OR	
3.1.b	Addition of short sounds	4	3.1.1.b	Page loading	AND	1,4
			3.1.2.b	Different link types	AND	1,3
			3.1.3.b	Different file formats	AND	1,3
3.2.a	Colour of text and background	2	3.2.1.a	By passing mouse	AND	0,8
			3.2.2.a	Getting focus	AND	0,6
			3.2.3.a	Specific areas	AND	0,6
3.3.a	Magnifying at passing by mouse	2	3.3.1.a	Links	AND	1
			3.3.2.a	Images	AND	1
3.4.a	Page information	2	3.4.1.a	Page pathway	AND	1
			3.4.2.a	Repeated end page line	AND	1
		100				100

Table 8.1 - Criteria and checkpoint scores according to our proposed model

When a page is evaluated, its score is the outcome of the addition of the scores of all the single checkpoints applied. The value of a non-applied checkpoint is 0, but if it is a "content-dependent" checkpoint and is unnecessary for the current page, it is treated as an applied one. The goal of this choice is not to mark down a site because of unnecessary checkpoints.

If for an "OR" criterion more than one checkpoint is applied, the value obtained by summing up the checkpoint scores is "normalised" to the criterion score.

For example, if the checkpoint 1.1.1.b (i.e. "grouping by headings") and 1.1.2.b (i.e. "grouping by hidden labels") are both applied, the addition of checkpoint score is 20; in the usability measurement computation we consider 20 "normalised" to 10 (i.e. to criterion score). Concluding, according to this proposed model, for each page there is a measurement that expresses its qualitative level of usability.

8.2.3 Usability Levels

According to criteria and checkpoints scores, we defined 9 usability levels to express a quality measurement of a Web site.

Next table shows the 9 quality levels.

Usability level	Score
Unusable	20
Scarce	30
Not sufficient	40
Minimum	50
Sufficient	60
Fairly good	70
Good	80
Very good	90
Maximum	100

Table 8.2 - Usability levels defined according to the proposed model

The “Minimum” usability level is fixed to 50, taking into account the case which all and only effectiveness criteria are applied. In this situation almost a “Minimum” level should be assigned.

8.3 Model Application Results

The model presented above has been applied for evaluation of 16 Web sites, of which 11 were Italian, and 5 British. In particular, we evaluated 6 public administration’s Web sites, and 5 Web sites chosen among providers of services of general interest. The five British Web sites chosen were the counter parts of five Italian sites, in fact they provided the same kind of public service or information (National government site; national railway site; national airway site; national whitepage service site; country capital site).

For each Web site 5 pages were considered. So, for every Web site, five distinct scores, so five usability levels, were determined and used in for computing the final usability level of the Web site. The Web site evaluation score is the average of the 5 single page score. Then, according to values reported in Table 8.2, the corresponding usability level was defined.

In the next table, all evaluation scores of the 16 chosen Web sites are reported.

Web site	Effectiveness	Efficiency	Satisfaction	Total	Quality level
regione.lombardia.it	6.6	9	1.6	17.2	Unusable
regione.toscana.it	26.04	25.58	0.84	52.46	Sufficient
regione.campania.it	31.13	19.8	0.6	51.53	Sufficient
euronews.it	26.7	20	0.2	46.9	Minimum
repubblica.it	22.02	18.8	0.6	41.42	Minimum
paginebianche.it	28.96	19.14	1.5	49.6	Minimum
trenitalia.com	24.72	14.66	0.6	39.98	Not sufficient
alitalia.it	22.13	14.67	0	36.8	Not sufficient
governo.it	28.68	34.2	1.28	64.16	Fairly good
comune.pisa.it	28.9	13.77	1.93	44.6	Minimum
comune.roma.it	30	16	2.27	48.27	Minimum
whitepages.lycos.com	16.7	9.33	1.27	27.3	Scarce
uk railways.com	32.23	16.33	0.4	48.96	Minimum
britishairways.com	13.3	14.07	0.6	27.97	Scarce
uk government.com	40	24	1.6	65.6	Fairly good
Uk londontwon.uk	13.34	11	0.6	24.94	Scarce

Table 8.3 - Evaluation scores resulting from the proposed model application to 16 Web sites

The results of the evaluation of the 16 Web sites considered is summarised in Table 8.3. Only two Web sites turned out to be “Fairly good” (i.e. Italian and British Government Web sites), and two sufficient (i.e. other two public administration Web sites).

Next graphic shows the percentage of the Web sites examined for each usability level

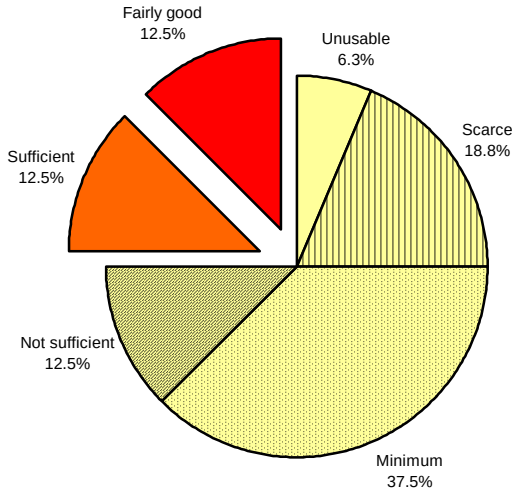

Figure 8.1 – Percentage of Web sites assigned to different usability levels

In Figure 8.2 a histogram graph is reported: it separately shows, for each site, the score assigned by the three groups of criteria effectiveness, efficiency, and satisfaction); in Figure 8.3 the total score obtained by every Italian site is shown.

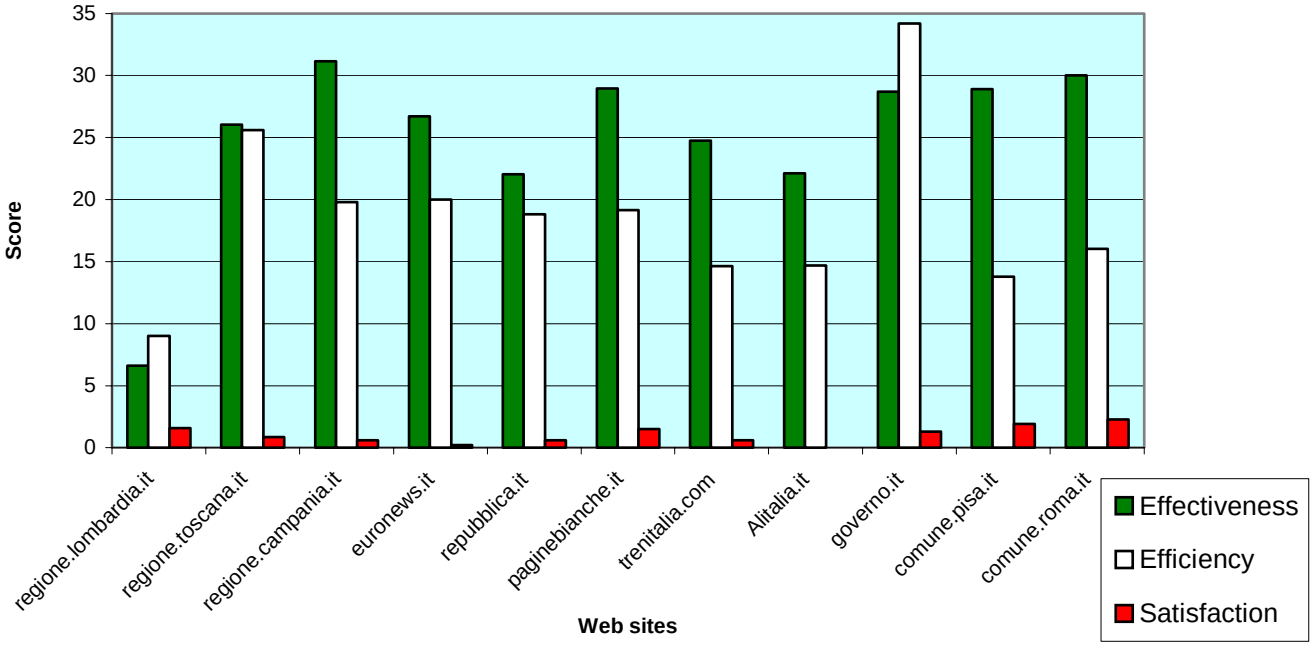

Figure 8.2. Evaluation scores for each site, considering separately the 3 groups of criteria

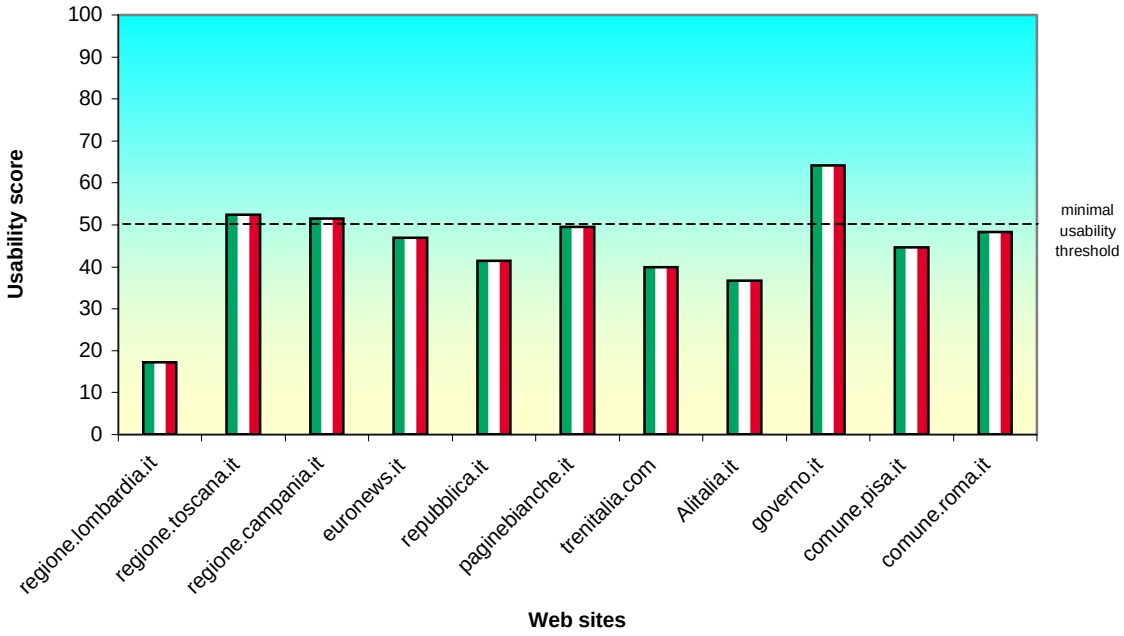

Figure 8.3 - Usability levels resulting from the Italian Web sites' evaluation

Lastly, the next figure compares the usability levels of the Italian Web sites with those of British. As we can note, there are not particular differences; however a slight difference can be observed: the Italian Web sites result slightly better of those British.

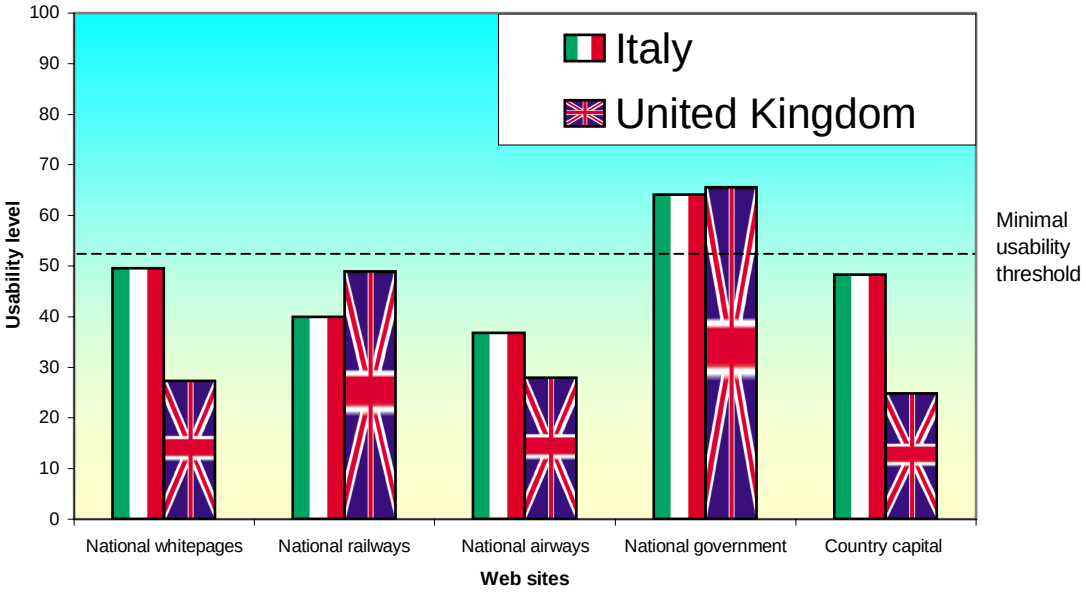

Figure 8.4 - Comparison between the usability level of the Italian and the British Web sites examined.

In conclusion, in this chapter we proposed a criteria-based method that we applied to some selected Web sites. The results coming out from this evaluation indicate that few Web sites have a sufficient usability level and underline the necessity of formalizing standardized criteria that can guide developers to create really usable site.

8.4 Final Considerations

The method explained in this chapter simply represents one of the possible ways of evaluation of Web page usability and, even if a single experiment of application was performed, some indications on advantages and disadvantages of such method have already emerged.

Our evaluation method is certainly simple and easily applicable, and it gives synthetic, but rather informative results. Moreover, from such results it is possible to detect which of the three groups of criteria are followed the most.

On the other hand, the method cannot be completely implemented by an automatic tool (since it requires the intervention of the evaluator). Besides, it is not always easy to establish whether a criterion is applied or not, when more than one Web page element is involved. Finally, criteria belonging to the "OR" type are considered satisfied if at least one of their checkpoints is applied. This means that if more than one checkpoint is applied, no extra-score is given for the final evaluation. Thus, the method does not distinguish between a satisfied criterion (one checkpoint applied) and an over-satisfied criterion (more checkpoints applied).

For example, in order to consider criterion 1.1.b ("logical partition of interface elements") as satisfied, it is sufficient that one of the following checkpoints is applied: hidden labels, heading levels, or frames. However, if all these three checkpoints are applied, users have more possibilities for moving through the page, understanding content organization etc., so they have a higher number of degrees of freedom.

Our method includes both static and dynamic evaluation. The first one is directly performed on the Web page code, by comparing it with the checkpoints related to the criteria. The dynamic analysis is performed by directly testing the interaction

between the Web page interface and the screen reader. This kind of analysis can be carried out only by an expert evaluator, which has to be familiar with internet navigation, guidelines/criteria, and also with screen reader usage.

In fact, in order to determine whether the Web interface properly interacts with the screen reader, it is necessary to know both browser and screen reader keyboard commands for navigation.

In other words, parts changed or added to the Web pages by our criteria and checkpoints aim at conferring characteristics interpretable by the screen reader. Often, more advanced configurations of the screen reader are required. Consequently, if the dynamic evaluation on one side allows to test directly the interaction UI-screen reader, on the other side it provides a rather subjective evaluation, depending on tester personal knowledge.

On the contrary, the static evaluation is more objective, both when it is manually performed by an expert evaluator and when it is executed by an automatic tool. However, even in the case of a static evaluation of the code, subjective decisions have to be made, for example when the evaluator is asked to decide if a certain link, button, or table summary have the appropriate text. With this kind of evaluation the tester does not need to know exactly all the commands of the screen reader.

Mariage and Vanderdonckt [2001] analyzed various ways to mechanize the evaluation of some design guidelines towards automated evaluation, concluding that the static analysis of HTML code can certainly provide some support for automated evaluation of very-large Web sites, but is clearly a hard to reach target. In general, we believe that both static and dynamic evaluation are necessary, since they provide different kinds of information and detect different failures: this aspect is really very important for a correct and reliable evaluation of the usability of Web sites.

9
Towards an Automatic Evaluation and Application

9.1 Introduction

In this chapter we introduce a brief discussion about a possible automatic evaluation and application of our proposed criteria. Since the usability of accessibility criteria for improving the navigation through screen reader we proposed, our algorithm should implement the evaluation of those criteria. That means we would like the algorithm is able to check the Web pages in order to evaluate if developers applied the usability criteria for accessibility and, when possible, it aids them to repair the pages. Although an automatic support is useful, the developer's (or evaluator's) participation is advisable.

Each criterion can be applied, as seen in [5.3] and [6.3], more then one checkpoints could be necessary, with 'OR' or 'AND' logical relation conditions. So, an automatic tool should assist evaluator/developer in the checking and applying of those conditions. Practically, the automatic tool should implement the Eval and Repair functions described in chapter 6.

Furthermore, in chapter 8 we proposed an evaluation model that can be used for an inspection criteria-based evaluation. In addition, we reported some examples regarding the methodology application, which were conducted by hand. Therefore, an automatic support to apply that heuristic-model would be advisable.

In this chapter we introduce a proposed tool working on our recommendations. A preliminary version of the tool is ongoing; herein we would like just to discuss about its functionalities.

9.2 Description of the Algorithm

Our proposed tool is aimed at implementing two functionalities: first, an evaluation routine which checks if the developer applied the criterion; second, an repair routine which suggests and assists developer to fix or to apply the non-complied criterion. The idea is developing an interactive tool, with which developer (or evaluator) can interact during the evaluation and application process. For example, the developer could be involved in fixing a "problem", or in customising some "parametric criterion" (e.g., number of links or frames).

9.2.1 Evaluation and Repair Phases

The evaluation routine takes the Web page and verifies, as far as possible, if a specific criterion is applied. To this end, it checks HTML or JavaScript code, and compare them to necessary checkpoints in order to detect the application status of the criterion.

As already shown in chapter 6, as result of this routine three possible responses can be come out:

- "Applied" means that the evaluated criterion is complied by the necessary checkpoints application;
- "Not applied" means that no necessary checkpoints for that criterion is applied;
- "To be reviewed" means that the criterion is used, but it is not completely applied; not all necessary checkpoints are applied, or they are not correctly used.

For example, the evaluation routine identifies that all frames have both "NAME"and "TITLE" attributes, but it detects that their values are different between them. In this case, it could be useful getting attention to developers about meaning of them. Therefore, in this example the criterion is applied, but it is to be reviewed (i.e., the evaluation result will be "To be reviewed").

In order to assist developer (or evaluator) to fix the problem, an interactive process is suggested: a repairing routine.

The repair routine takes the Web page and, by involving the developer, tries to modify the code in order to improve the usability aspects. I.e., after evaluation routine detected the problem, developer is warned, and he/she can decide to “repair” it. Thus, the repairing routine assists developer in this process.

For instance, let consider the “repairing” of the navigation bar. The procedure adds the appropriate hidden label (or frame name), but it needs of developer’s indication about the links forming the navigation bar.

Or, let consider the case in which buttons used in all Web pages have not VALUE attribute; the routine can add automatically those attributes, but it needs of the necessary strings to associate to them.

However, even if the developer's participation is required, modifying automatically most of code, especially when the Web site consists of a great number of pages, can be a useful tool in order to reduce usability problems.

We think of the repairing phase is the most important, because after that problems come out through evaluation routine, the effort to fix them could be considerable. Furthermore, since one problem can be often related to the whole Web site, developers should fix too many pages; that could discourage them from doing it. Therefore if a semi-automatic or fully automatic assistant exists, it is more likely that some changes are made.

Unfortunately, fixing phase is more difficult then that one evaluation, because there exist many types of Web pages, and introducing changes without causing non-desired effects is unlikely. In addition, there are many changes that can not be quite automated, such as assign suitable names to frames, or appropriate texts to links: the repair routine can insert the correct code, but the text has to be given by users. In order to do this automatically, a natural language processing phase could be useful, even if it would be difficult in just the same. Then, another instance is the navigation bar design: the developer marks the links forming the bar, and then the repair procedure modifies all the pages.

If the developer wants to add one sound to each page loading, or different sounds to different link types, the algorithm can operate more automatically. Practically, it

creates the JavaScript file with the ‘PlaySound()’ function, and then it inserts into all pages the necessary code to call that JavaScript file. Next, the algorithm tries to detect various types of links and adds the correct code in order to associate them different sounds. This could be done automatically by a repairing routine.

9.2.2 Input and Output Data

The input of the algorithm can be a single page, or the entire site. The user can choose if applying the evaluation or fixing criterion to one single Web page, or to whole site. This depends on type of criterion, and / or on user's choice.

Evaluator can choose what to do, whether evaluation or repairing, which one criterion to apply, and so on. The user introduces the page address from which starting. Then developer can decide if checking only that page, or also the pages linked in the starting page.

Therefore the first input data is page code. In addition to this, we can have some dictionary files, in which various terms can be memorized in order to use them to evaluate link text or frame name. That means dictionaries can be contain additional information to use during the evaluation of checkpoints.

File name	Description	Used by module
Non-frames.dic	Contains a list of non-advisable names for frames	Evaluation
Frames.dic	Contains a list of Advisable names for frames	Repair
Non-links.dic	Contains a list of non-advisable texts for links	Evaluation
Navlink.dic	Contains possible texts of navigation links	Eval/repair
Buttonsyn.dic	Contains a list of synonimous terms useful for consistency issues	Evaluation
Non-table.dic	Contains non-advisable contents for summary table	Evaluation
Non-images.dic	Contains possible non-advisable definitions to use with alternative text (e.g., filename.gif, .jpg, .pcx, etc.)	Evaluation

Table 9.1 - Dictionary files used during the evaluation process

The possible dictionary files are listed in the Table 9.1.
In addition, the evaluation or repairing procedures are based on our proposed criteria. Them substantially consist of one or more checkpoints, i.e. HTML, CSS, or JavaScript constructs to verify or apply to Web pages.

The outputs of the algorithm are the results coming out from evaluation and repair routines. The result of evaluation process is a report that summarizes the analysed criteria, their application status (“Applied”, “Not applied”, or “To be reviewed”), and some indications relating to the checkpoints used. Moreover, the “non-correct” code is visualised so that developer can quickly identify the portion to be reviewed.
The repair process gives out the page/s changed according to checkpoints and developer's choices.

9.3 An Overview of the Tool

As the development of our proposed automatic tool is at the beginning, in this paragraph we give just an idea about it works. In addition, it is a preliminary version of the tool supporting our evaluation method. Hovever, we would like to summarise some considerations about its functionalities.
The proposed tool is aimed at implementing the evaluation criteria and helping the developer in fixing the detected problems.
Next picture shows the user interface of the tool at its starting.

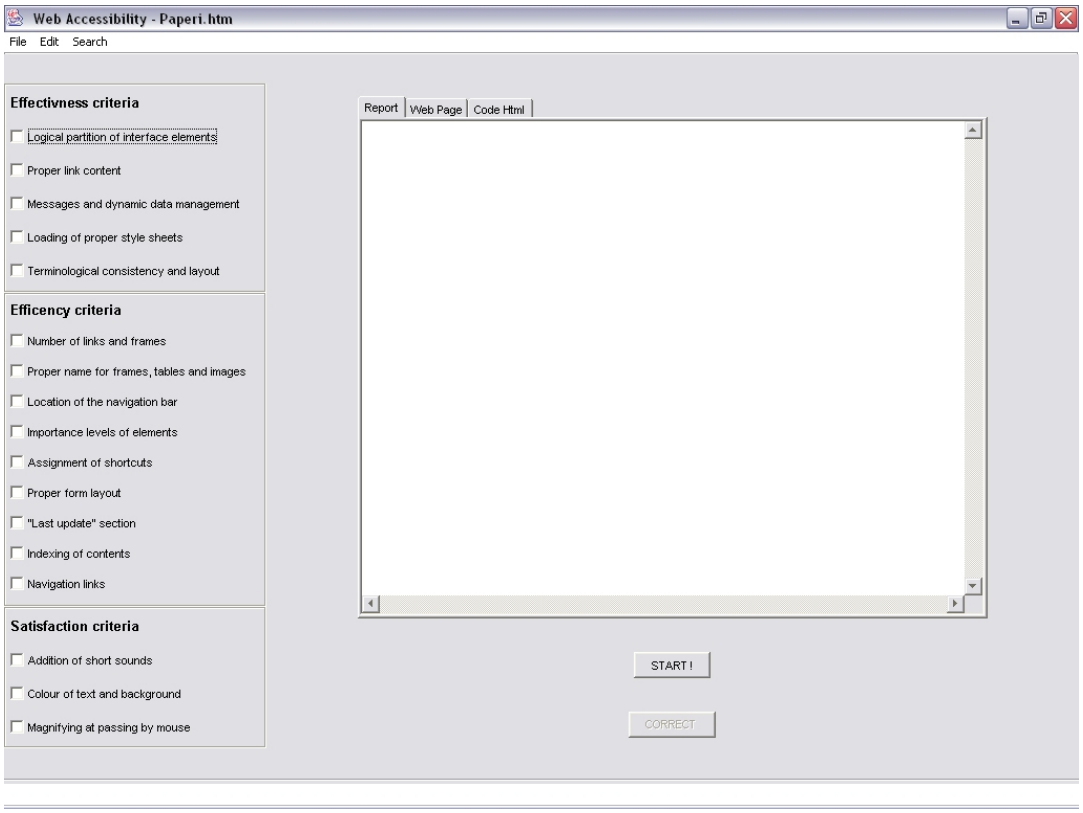

Figure 9.1 - User interface of the tool when it starts.

As the title shows, a Web page to be checked is open. By clicking on “Start” button, the evaluation phase begins. The list of all the criteria grouped by type (effectiveness, efficiency, and satisfaction) is available. The developer can chose which criteria to be verified, by selecting the corresponding checkboxes.

Next picture shows the output generated by the tool as result of a Web page evaluation.

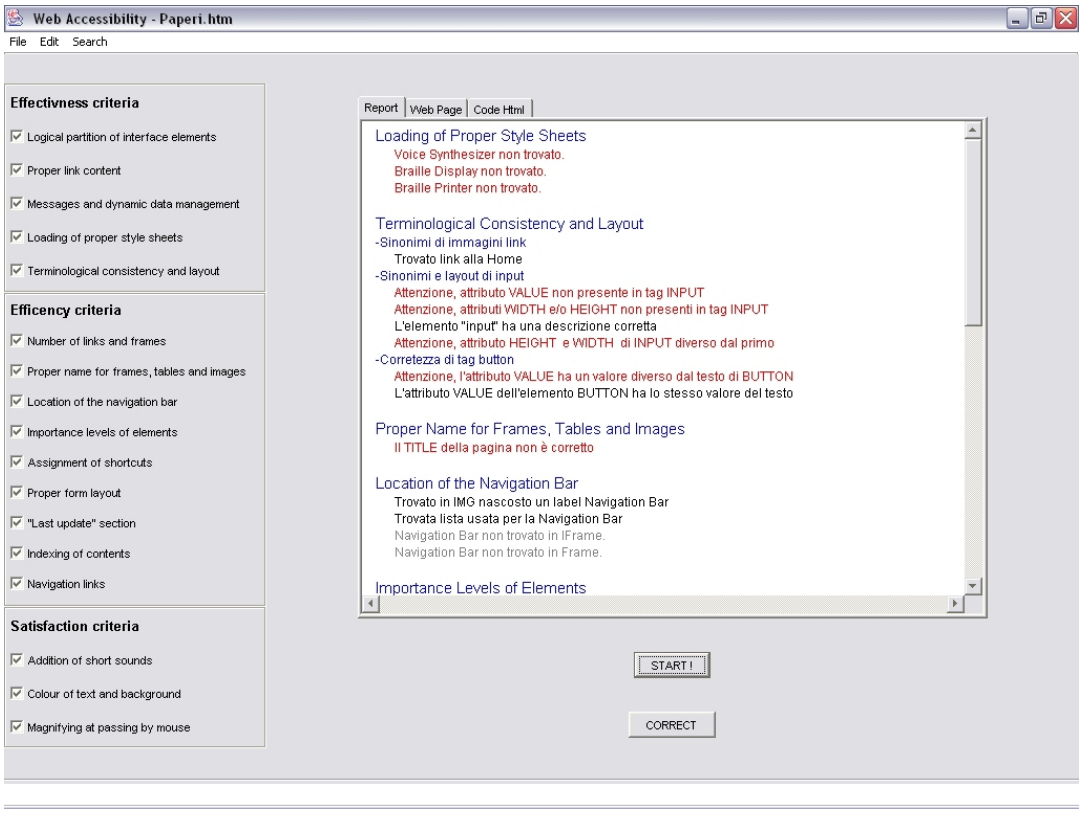

Figure 9.2 – Screenshot of a tool output relating to a Web page criteria-based evaluation

The red writings are the parts relating to non applied checkpoints; those in grey, warn the evaluator in paying attention to some application elements. For example, the tool can detect that more then one `TABINDEX` attributes with same values were applied. This could not be an "error", because assigning a same `TABINDEX` attribute to more elements could be a developer's decision. So, those parts are pointed out in grey just to get attention. The developer "knows" if it is an error or not.

The parts written in black are related to fixes already made, and those in blue indicates the name of criteria.

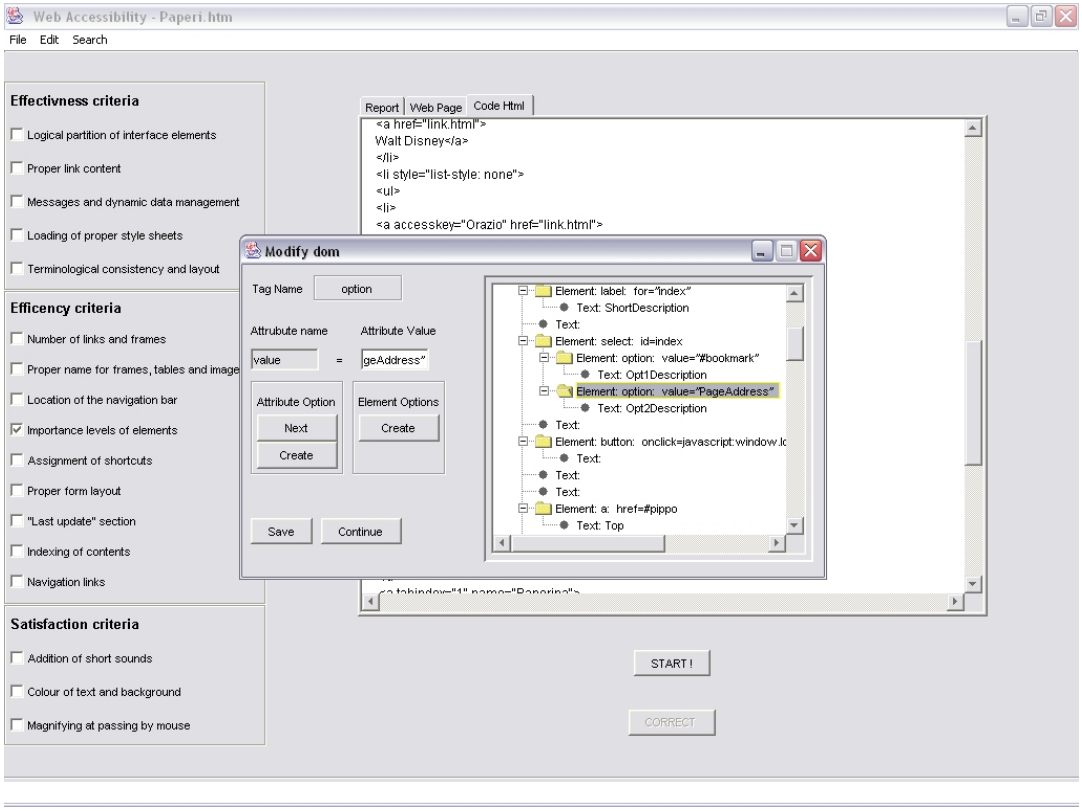

Figure 9.3 – Interactive window where the developer can change the page code

As shown in Figure 9.3 the developer can directly make changes in the Web page by interacting with the DOM model. In fact, the tool provides a DOM three structure where the elements to be changed can be selected. The made modifies affect the Web page HTML code, so that a “repaired” page is generated.
The tool window is split in two areas: one contains buttons whereby the DOM three can be navigate; in the other at the right, the DOM three of the Web page is visualised. However, for less expert users, it is also possible changing the problem directly in the HTML code.

Lastly, another example of interaction with the tool is shown. Next picture visualises how a dictionary file can be modified or customised from the developer. A dictionary file contains all the terms referred to those criteria considering the use of proper content and text (e.g. frame names, content link, etc.). The tool allows to the developer to interact with the dictionary file by adding, changing and customising terms by different language.

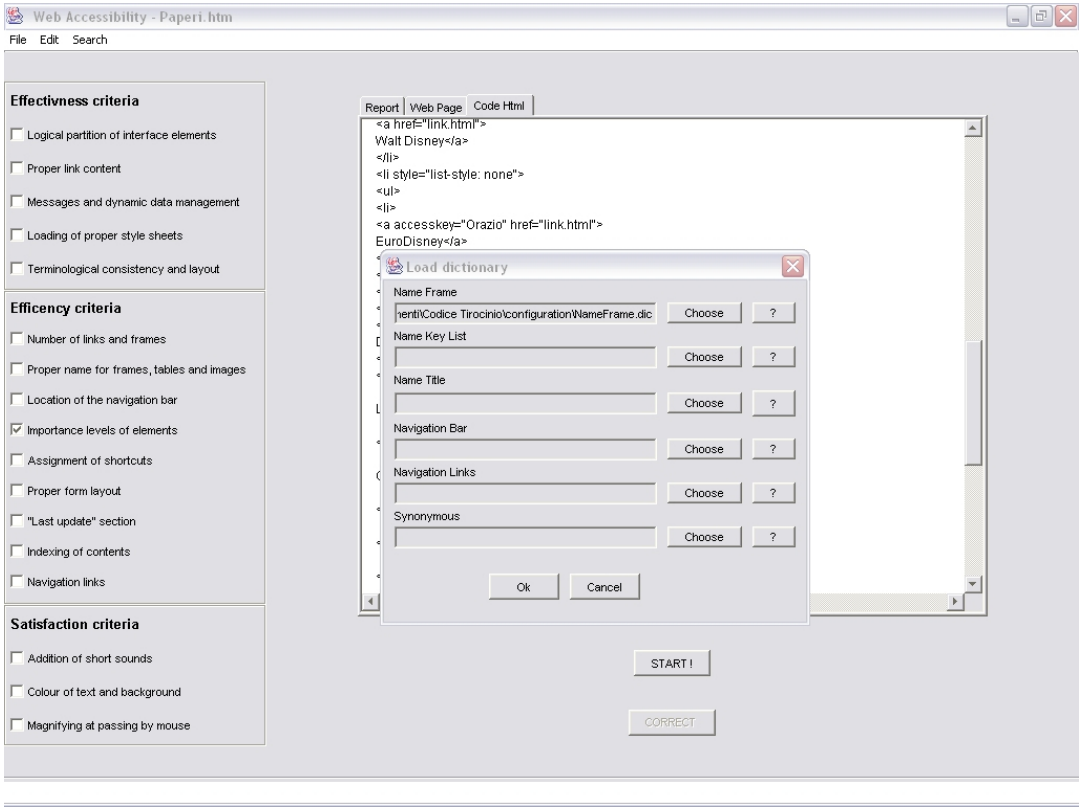

Figure 9.4 – Window in which a dictionary file can be open and modified

9.4 Remarks

In this chapter we gave a brief introduction of a semi-automatic tool working on our criteria. As also shown through the figures, the philosophy of this tool basis on a dynamic interaction between the tool and the developer / evaluator. In fact, providing a support that allows to made customisation and interacting activities can improve the evaluation and repairing phases. In addition, it could reduce the bias due to “non-desirable” effect deriving from an automatic process.

10
Conclusions

During the last years there has been an increasing interest in accessibility and usability issues, since it is more and more important that the information on the internet can be easily reachable for all categories of users, including disabled people.
The Web offers one of the best opportunities to get information of any kind and it is fundamental to have World Wide Web resources accessible to people with disabilities.
The Usability of a Web site depends on how easily its organization and functions can be learnt and remembered and on other three variables: effectiveness, efficiency, and satisfaction of the navigation.
In this work, we consider usability and accessibility as two distinct, even if correlated, concepts. Our purpose herein was to identify the relationship between those concepts: in particular we aimed at identifying the meaning of usability of Web sites when they are accessed by disabled users through screen readers.
In order to improve the navigability through a screen reader, we proposed 19 usability criteria that we partitioned in three sub-sets related to different aspects of usability: effectiveness, efficiency and satisfaction. Those criteria should be taken into account during the Web site development phase. To facilitate the application of the proposed criteria by developers of Web sites, we suggested 54 technical checkpoints. Each checkpoint was indicated as “to be applied” and “to be checked”, in order to provide specific indications both to developers and evaluators.
We provided some examples of possible applications on the basis of existing Web sites, thus showing how the criteria can be used and in which way they can affect the Web user interface when navigating through screen reader.
Next, after defining the relationship between criteria and checkpoints, a pilot heuristic-based evaluation method was presented and used to define nine usability

quality levels. In addition, it was applied to 16 Web sites in order to “test” its functionality and to assess the usability levels of such Web sites. This proposed quality model basically consists of a polynomial expression of criteria with weights base on a Boolean expression among checkpoints. The research is in progress to improve this methodology.

In order to evaluate the real impact of our proposed criteria on the Web interface, a user testing was conducted. In our testing procedure, two groups of people were involved: blind and visual impaired subjects. Although our results are empirical and preliminary, they certainly reveal that our usability criteria have a positive effect on Web site navigability and bring valuable benefit to blind and low vision users.

Lastly, a brief overview of an automatic tool, whose implementation is in progress, was reported.

The research is in progress to improve the heuristic-based evaluation methodology and the definition of criteria and their checkpoints, in the optic of integrating the automatic tool implementation. The usability testing has certainly to be better organised, by improving the test phases, the gathering of data and the analysis of the results. In conclusion, our work represents a first step into usability and accessibility issues for special users.

Bibliography

[**Abascal et al. 2002**] Abascal, J., Arrue, M., Garay, N., Tomás J. *USERfit Tool. A tool to facilitate Design for All.* In proceedings of the 7th ERCIM Workshop "User Interfaces for All", Paris (Chantilly), France, 23 - 25 October 2002, in "Universal Access: Theoretical Perspectives, Practice, and Experience", published by Springer-Verlag, Lecture Notes in Computer Science n.2615 (2003), pp. 141-152.

[**Abascal et al. 2003**] Abascal J., Arrue M., Fajardo I., Garay N., Tomás J., *Use of Guidelines to automatically verify Web accessibility*. Universal Access in the Information Society. Special Issue on "Guidelines, standards, methods and processes for software accessibility" (Guest editors: Gulliksen J., Harker S., Vanderheiden G.). Accepted for publication (Expected publication date: Autumn 2003).

[**Allen 1996**] Allen, B., Information tasks: *Toward a User-Centered Approach to Information Systems*. Academic Press, 1996

[**Balbo 1995**] Balbo, S., 1995. *Automatic evaluation of user interface usability: Dream or reality*. In S. Balbo Ed., Proceedings of the Queensland Computer-Human Interaction Symposium (Queensland, Australia, August 1995). Bond University.

[**Barnicle 2000**] Barnicle, K. *Usability Testing with Screen Reading Technology in a Windows Environment*. Proceedings of the 2000 Conference on Universal Usability (CUU-00), pp. 102-109, ACM Press, November 16-17 2000.

[**Basili and Weiss 1984**] Basili, V.R., and Weiss D., *A methodology for collecting valid software engineering data*, IEEE Trans. On Software Engineering, SE-10(6), pp. 728-738, 1984.

[**Bastien et al. 1999**] Bastien, J.M.C., Scapin, D.L., Leulier, C., *The Ergonomic Criteria and the ISO 9241-10 Dialogue Principles: A comparison in an evaluation task*. Interacting with Computers, 11(3), 1999, pp. 299-322.

[**Beirekdar et al. 2002A**] Beirekdar, A., Vanderdonckt, J., Noirhomme-Fraiture, M., *A Framework and a Language for Usability Automatic Evaluation of Web* Sites by Static Analysis of HTML Source Code, Chapter 29, Proc. of 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2002 (Valenciennes, 15-17 May 2002), Kluwer Academics Pub., Dordrecht, 2002, pp. 337-348).

[**Beirekdar et al. 2002B**] Beirekdar, A., Vanderdonckt, J., Noirhomme-Fraiture, M., *KWARESMI - Knowledge-based Web Automated Evaluation with REconfigurable guidelineS optiMIzation*, in PreProceedings of 9th International Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2002 (Rostock, 12-14 June 2002), University of Rostock-Université catholique de Louvain,

[**Brajnik 2000**] Brajnik, G., 2000. *Automatic Web usability evaluation: what needs to be done?* In Proceedings of the 6th Conference on Human Factors & the Web (Austin, TX, June 2000). Available at http://www.tri.sbc.com/hfWeb/brajnik/hfWeb-brajnik.html.

[**Brajnik 2001**] Brajnik, G., *Towards valid quality models for Websites*, in Proc. Human Factors and the Web, 7th Conference, Madison, WI, June 2001. www.dimi.uniud.it/~giorgio/papers/hfWeb01.html

[**Brajnik 2002**] Brajnik, G. *Quality Models based on Automatic Webtesting*. Presented at the CHI2002 workshop Automatically evaluating usability of Web Sites", Minneapolis (MN), April 2002, www.acm.org/sigchi/chi2002.

[**Brewster 1994**] Brewster, S.A. (1994). *Providing a structured method for integrating non-speech audio into human-computer interfaces*. PhD Thesis, University of York, UK

[**Clarck and Dardailler 1999**] Clarck D., Dardailler D. (1999) *Accessibility on the web: Evaluation and repair tools to make it possible*. In proceedings of the CSUN Technology and Persons with disabilities Conferences, Los Angeles, CA. Available at http://www.cast.org/bobby

[**Cooper et al. 1999**] Cooper, M., Limbourg Q., Mariage, C., Vanderdonckt J., *Integrating Universal Design into a Global Approach for Managing Very Large Web Sites*, in Proc. of the 5th ERCIM Workshop on User Interfaces for All (Dagstuhl, 28 November-1 December 1999), A. Kobsa & C. Stephanidis (eds.), GMD, Sankt Augustin, 1999, pp. 131-150.

[**Coutaz 1995**] Coutaz, J., 1995. *Evaluation techniques: Exploring the intersection of HCI and software engineering.* In R. N. Taylor and J. Coutaz Eds., Software Engineering and Human- Computer Interaction, Lecture Notes in Computer Science, pp. 35-48. Heidelberg, Germany: Springer-Verlag.

[**Dix et Al. 1998**] Dix, A., Finlay, J., Abowd, G., and Beale, R., 1998. *Human-Computer Interaction* (Second ed.).

[**Etgen and Cantor 1999**] Etgen, M. and Cantor, J., 1999. *What does getting WET (Web Event-logging Tool) mean for Web usability*. In Proceedings of the 5th Conference on Human Factors & the Web (Gaithersburg, Mary- land, June 1999). Available at http://www.nist.gov/itl/div894/vvrg/hfWeb/ proceedings/etgen-cantor/index.html.

[**Farenc et al. 1996**] Farenc, Ch., Liberati, V., Barthet, M.-F., *Automatic Ergonomic Evaluation : What are the Limits?*, in Proc. of 2nd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96 (Namur, 5-7 June 1996), J. Vanderdonckt (ed.), Presses Universitaires de Namur, Namur, 1996, pp. 159–170.

[**Flanagan 2001**] Flanagan, D., JavaScript: The Definitive Guide, 4th Edition. O'Reilly, 2001.

[**Fleming 1998**] Fleming, J., *WEB navigation: designing the user experience*. O'Reilly, 1998

[**Grammenos et al. 2000**] Grammenos, D., Akoumianakis, D., & Stephanidis, C. (2000). *Integrated Support for Working with Guidelines: The Sherlock Guideline Management System*. International Journal of Interacting with Computers, special issue on "Tools for Working with Guidelines", 12 (3), 281-311

[**Hilbert and Redmiles 2000**] Hilbert, David M., Redmiles, David F., 2000. *Extracting Usability Information from User Interface Events*. - University of California at Irvine - ACM Computing Surveys, Vol. 32, No. 4, December 2000, pp. 384- 421.

[**ISO**] ISO - International Organization for Standardization. http://www.iso.ch/iso

[**Ivory and Hearst 1999**] Ivory, M. Y. and Hearst, M. A.., 1999. *Comparing performance and usability evaluation: New methods for automated usability assessment*. Unpublished manuscript. Available at http://www.cs.berkeley.edu/_ivory/research/Web/papers/pe-ue.pdf.

[**Ivory and Hearst 2001**] Ivory, M. Y. and Hearst, M. A. (2001). *"The state of the art in automating usability evaluation of user interfaces"*. ACM Computing Surveys, 33(4), pp. 470-516, December 2001.

[**Jain 1991**] Jain, R., *The Art of Computer Systems Performance Analysis*, John Wiley and Sons, Inc. 1991.

[**JFW**] Henter-Joyce, *Jaws For Windows*. http://www.hj.com

[**Lecerof and Paternò 1998**] Lecerof, A., Paternò, F., 1998. *Automatic Support for Usability Evaluation* - IEEE Transactions on Software Engineering, Vol. 24, N° 10, October 1998, pp. 863 - 888.

[**Leporini and Paternò 2002**] Leporini, B., Paternò, F., 2002. *Criteria for Usability of Accessible Web Sites*. In proceedings of the 7th ERCIM Workshop "User Interfaces for All", Paris (Chantilly), France, 23 - 25 October 2002, “Universal Access: Theoretical Perspectives, Practice, and Experience” published by Springer-Verlag, Lecture Notes in Computer Science n.2615 (2003), pp. 43-55.

[**Leporini and Paternò 2003**] Leporini, B., Paternò, F., 2003. *Increasing Usability when Interacting through Screen Readers.* Accepted by Springer International Journal Universal Access in the Information Society (UAIS), Special Issue on "Guidelines, Standards, Methods and Processes for Software Accessibility" (Expected publication date: Autumn 2003).

[**Levental and Earl** 2000] Levental, J., Earl, C., *Windows-based SCREEN readers: tools for beginners and power users*. Proceedings of the 2000 Conference on Universal Usability (CUU-00), ACM Press, November 16-17 2000.

[**Lindgaard 1989**] Lindgaard, G. *Testing the Usability of Interactive Computer Systems*, In: Lindgaard, G. and Millar, J. (1989) Testing the Usability of Interactive Computer Systems. Proceedings of Workshop at HCI Australia’89. Ergonomics Society of Australia, Computer-Human Interaction Special Interest Group, 1-13.

[**Löwgren and Laurén 1993**] Löwgren, J., Laurén, U., *Supporting the use of guidelines and style guides in professional user interface design*. Interacting With Computers, Vol. 5, 1993, pp. 385 - 396.

[**Mariage and Vanderdonckt 2001**] Mariage, C., Vanderdonckt, J., *Mechanization of Web Design Guidelines Evaluation*, Proc. of 1st Int. Conf. on Universal Access in Human-Computer Interaction UAHCI’2001 (New Orleans, 5-10 August 2001), Stephanidis, C. (Ed.), Lawrence Erlbaum Associates, Vol. 3, Mahwah, 2001, pp. 190-194.

[**Meyer 2001**] Meyer, E., *Cascading Style Sheets 2.0 Programmer's Reference*. McGraw-Hill (eds), 2001.

[**Nicolle and Abascal 2001**] Nicolle, C., Abascal J. *Inclusive design guidelines for HCI*, p. 285, Taylor & Francis, 2001.

[**Nielsen 1993**] Nielsen, J., *Usability Engineering*. Boston: Academic Press, 1993.

[**Nielsen and Mack 1994**] Nielsen, J., and Mack R., (eds), *Usability Inspection Methods*, Wiley, 1994.

[**Nielsen 1999**] Nielsen, J., *Designing Web Usability: the practice of semplicity*. New Riders Publishing, 1999.

[**Nielsen 1999**] Nielsen, J., http://www.useit.com/alertbox, January 10, 1999

[**Nielsen 2000A**] Nielsen J., *Designing Web usability*. Macmillan computer Publishing (eds), 2000.

[**Nielsen 2000B**] Nielsen, J., http://www.useit.com/alertbox, March 2000

[**Nielsen and Mack 1994**] Nielsen, J., and Mack, R.,(eds), *Usability Inspection Methods*. Wiley, 1994.

[**Nist**] Nist Web Metrics: http://zing.ncsl.nist.gov/WebTools/tech.html

[**OSW**] ALVA B.V., OutSpoken. http://www.alva-bv.nl

[**Paganelli and Paternò 2002**] Paganelli, L., Paternò, F. *Intelligent Analysis of User Interactions with Web Applications*. Proceedings ACM Intelligent User Interfaces, pp.111-118, ACM Press, San Francisco, January 2002.

[**Paternò 1999**] Paternò, F., *Model-Based Design and Evaluation of Interactive Application.* Springer Verlag, ISBN 1-85233-155-0, 1999.

[**Paternò and Ballardin 1999**] Paternò, F., Ballardin, G., 1999. *Modelaided Remote Usability Evaluation*. Human Computer Interaction INTERACT'99

(Edinburgh, August 1999), A. Sasse & Ch. Johnson (eds.), IOS Press, 1999, pp. 434- 442.

[**Paternò and Ballardin 2000**] Paternò, F., Ballardin, G., *RemUsine: a bridge between empirical and model-based evaluation when evaluators and users are distant.* Interacting with Computers, Vol.13, N.2, 2000, pp. 151-167.

[**Paternò and Paganelli 2001**] Paternò, F., Paganelli, L., 2001. *Remote evaluation of Web sites based on task models and browser Monitoring*. Proceedings of CHI'01 (Seattle, WA, USA, April 2001), pp 283-284.

[**Paternò et al. 1997**] Paternò, F., Mancini, C., Meniconi, S., *ConcurTaskTrees: Diagrammatic notation for specifying task models.* Proceeding of the IFIP TC13 6th Interational Conference on Human-Computer Interaction (Sydney, Australia, July 1997), pp. 362-369.

[**Patton 2001**] Patton, R., 2001. *Software Testing*. Sams Publishing.

[**Rowan et al. 2000**] Rowan, M., Gregor, P., Sloan, D. and Booth, P. (2000). *"Evaluating Web Resources for Disability Access"*. Fourth Annual ACM Conference on Assistive Technologies (ASSETS00), pp. 80-84, ACM, 2000.

[**Scapin et al. 2000A**] Scapin, D., Leulier, C., Vanderdonckt, J., Mariage, C., Bastien, C., Farenc, C., Palanque, P., Bastide, R., *Towards automated testing of Web usability guidelines.* Human Factors and the WEB, 6th Conference, Austin, June 2000.

[**Scapin et al. 2000B**] Scapin D., Vanderdonckt, J. Farenc, CH., Bastide, R., Bastien, CH., Leulier, C., Mariage, C., Palanque, PH. *Transferring Knowledge of User Interfaces Guidelines to the Web*, in Proc. of Int. Workshop on Tools for Working with Guidelines TFWWG'2000 (Biarritz, 7-8 October 2000), Springer-Verlag, London, 2000, pp. 293-303.

[**Shneiderman 1998**] Shneiderman, B., 1998. *Designing the user interface: strategies for effective human- computer interaction* (Third ed.). Reading, Mass.: Addison-Wesley, 1998.

[**Seagle 1956**] Seagle S., (1956). *Non parametric statistics*: *for the behavioral science.* New York. Mc Graw - Hill Book Company, Inc.

[**Stephanidis et al. 1997**] Stephanidis, C., Paramythis, A., Karagiannidis, C., Savidis, A. *Supporting Interface Adaptation: the AVANTI Web-Browser*. 3rd ERCIM Workshop on "User Interfaces for All", Strasbourg, France, November 3-4, 1997.

[**Stephanidis et al. 1998**] Stephanidis, C., Akoumianakis, D., Sfyrakis, M., & Paramythis, A. (1998). *Universal accessibility in HCI: Process-oriented design guidelines and tool requirements*. In C. Stephanidis & A. Waern (Eds.), Proceedings of the 4th ERCIM Workshop on "User Interfaces for All", Stockholm, Sweden, 19-21 October (15 pages).

[**Stein 1997**] Stein, L. D., 1997. *The rating game.* http://stein.cshl.org/□lstein/rater/.

[**Turk 2001**] Turk, A., *Towards Contingent Usability Evaluation of WWW Sites*, Proceedings of Australian Int. Conf. on Computer-Human Interaction OZCHI'2001.

[**Usable net 2000**] Usable Net., 2000. *LIFT online.* http://www.usablenet.com.

[**Vanderdonckt 1999**] Vanderdonckt, J., *Development Milestones towards a Tool for Working with Guidelines, Interacting with Computers*, Vol. 12, N° 2, 1999, pp. 81- 118. Accessible at http:// belchi. qant. ucl. ac. be/ publi/ 1999/ Milestones. Pdf

[**W3C**] World Wide Web Consortium (W3C), *Web Accessibility Initiative (WAI)*, accessible at http://www.w3.org/WAI/

[**WAI 1999**] *WAI Accessibility Guidelines*. Web Accessibility Initiative, World Wide Web Consortium, 1999. Accessible at http://www.w3.org/TR/WCAG10/

[**Whitefield et al. 1991**] Whitefield, A., Wilson, F., and Dowell, J., 1991. *A framework for human factors evaluation.* Behaviour and Information Technology 10, 1, 65-79.