Usability Tool Support for Model-Based Web Development

Richard Atterer

München 2008

Usability Tool Support for Model-Based Web Development

Richard Atterer

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

vorgelegt von
Richard Atterer
aus Augsburg

München, den 2. Juli 2008

Erstgutachter: Prof. Dr. Heinrich Hußmann
Zweitgutachter: Prof. Dr.-Ing. Martin Gaedke
Tag der mündlichen Prüfung: 9. Oktober 2008

rev. 284

Contents

1 Introduction **1**

1.1 Motivation: Achieving Good Web Usability is Becoming Harder 1

1.2 Proposed Solution: Improved Web Development Tools, Models and Methods . . 3

1.3 How This Thesis is Organized 5

2 Usability on the Web **7**

2.1 Usability and Accessibility 7

2.2 Accessibility and Usability Guidelines for Web Pages 9

2.2.1 W3C WCAG 1.0 9

2.2.2 Legislation related to WCAG 1.0: BITV, US Section 508, Stanca Act . . 12

2.2.3 W3C WCAG 2.0 14

2.2.4 Research-Based Web Design & Usability Guidelines 18

2.2.5 Web Style Guide 20

2.2.6 Further Resources 21

2.3 Existing Usability Tool Support for Web Developers 23

2.3.1 Generation of Web Pages 24

2.3.2 Automatic Capture of User Interaction With Web Pages 26

2.3.3 Automatic Evaluation of User Interaction Logs 28

2.3.4 Automatic Accessibility and Usability Validation 31

2.3.5 Simulation of User Perception or Behaviour 36

3 Usability Aspects in Existing Web Engineering Development Processes **39**

3.1 Web Engineering – Overview 39

3.2 Web Engineering Approaches 41

3.2.1 UML-Based Web Engineering (UWE) 42

3.2.2 WebML 46

3.2.3 OO-H 50

3.2.4 Other Approaches 53

3.2.5 Conclusion 54

4 Method and Tool Concepts for Improved Web Usability **55**

4.1 Requirements Analysis Phase 56

4.1.1 Using Activity Logs for Web-Based Requirements Elicitation 58

4.1.2 Using a Support Website for Web-Based Requirements Elicitation 59
4.2 Design Phase 60
4.2.1 "Virtual Paper Prototyping" of Web User Interfaces 61
4.2.2 Usable Model Transformation and Code Generation Defaults 62
4.2.3 Automatic Usability Analysis of Models 64
4.3 Implementation Phase 67
4.3.1 Automatic, Continual, Model-Based Usability Evaluation 68
4.3.2 User-Friendly "Implicit Modelling Support" for Content Management Systems 69
4.4 Testing Phase 74
4.4.1 Web-Based Interaction Logging For User Tests 75
4.5 Maintenance Phase 78
4.5.1 Model-Based Usability Evaluation of Implementation and Usage Statistics 78
4.5.2 Support for User Feedback 81
4.6 Conclusion: Tools Can Help to Identify Usability Problems More Easily 83

5 A Model for Usability and Accessibility Information 85
5.1 Modelling of User Properties and Context 86
5.2 Model Information for Advanced Usability and Accessibility Tool Support 89
5.2.1 Overview of the Context Model 91
5.2.2 Information About the Users 94
5.2.3 Information About the Technical Platform 99
5.2.4 Information About the Environment 102
5.2.5 Presentational Aspects 104
5.2.6 Navigational and Functional Aspects 107
5.3 Integration With Existing Web Engineering Models 112
5.4 Embedding Model Information in the Implementation 117
5.4.1 Motivation 117
5.4.2 Alternatives for Representing Model Data 118
5.4.3 Proposed Method for Embedding Model Data 119
5.5 Conclusion 123

6 Wusab, a Tool for Automatic Model-Based Usability Analysis 125
6.1 Related Work 126
6.2 Improving Automated Usability Evaluation Using Models 132
6.2.1 Problem: Many Usability Guidelines are Abstract or Vague 132
6.2.2 Potential Benefits of Models as a Source of Meta Information 134
6.2.3 Examples for New and Improved Model-Based Usability Guideline Checks 136
6.3 Prototype Implementation 142
6.4 Discussion and Conclusion 148

7 UsaProxy, a Tool for Website Usage Analysis and Testing 149
7.1 Related Work 150
7.2 Tracking User Actions on Web Pages 154
7.2.1 Requirements 154
7.2.2 Approach 155
7.3 Making Use of the Collected Tracking Information 156
7.4 Implementation 161
7.4.1 Forwarding and Modifying HTTP Requests and Responses 161
7.4.2 Operation Modes of the Proxy 163
7.4.3 Client-Side JavaScript-Based Interaction Tracking 165
7.5 Example Output 167
7.6 Evaluation 168
7.7 Conclusion 172

8 Conclusions and Future Work 175
8.1 General Conclusion 175
8.2 Contributions of the Thesis 177
8.3 Future Work 179

Bibliography 183

Acknowledgements 199

Zusammenfassung

Bei der Verwendung einer Web Engineering-Methode für die Entwicklung von Web-Anwendungen werden während des Entwicklungsprozesses Modelle erstellt, die die Website beschreiben. Wenn eine verbesserte Benutzerfreundlichkeit angestrebt wird, ermöglichen die Informationen in diesen Modellen Werkzeuge, die weiter fortgeschritten sind als bisherige Ansätze, welche sich nicht auf das Vorhandensein von Modellen stützen. Diese Dissertation stellt Ideen für Werkzeug-Unterstützung während der verschiedenen Phasen der Entwicklung vor, z.B. für Implementierung oder Test. Beispielsweise kann ein Tool aus einem Modell über die Information verfügen, dass die Besucher einer Website Teenager sind. Es ist daher in der Lage zu untersuchen, ob die Wörter und Sätze auf der Website von Teenagern verstanden würden. Die Arbeit stellt einen Ansatz vor, wie existierende Web Engineering-Modelle um Zusatz-Informationen (in diesem Fall „Alter“) erweitert werden können, und wie die Informationen Werkzeugen zugänglich gemacht werden können, z.B. durch Einbetten in den HTML-Code. Zwei Prototypen setzen Konzepte für eine Einbeziehung von Usability-Werkzeug-Unterstützung in Web Engineering um.

Abstract

When web engineering methods are used for the development of web applications, models are created during the development process which describe the website. Using the information present in these models, it is possible to create usability tool support that is more advanced than current approaches, which do not rely on the presence of models. This dissertation presents ideas for tool support during different phases of the development, such as the implementation phase or the testing phase. For example, if a tool knows from a model that the audience of a website are teenagers, it can examine whether the words and sentences used on the website are likely to be understood by teenagers. An approach is presented to augment existing web engineering models with the additional information (“age” in this case) and to make it available to tools, e.g. via embedding it in HTML code. Two prototypes demonstrate the concepts for integrating usability tool support into web engineering.

Chapter 1

Introduction

This thesis shows ways for developers to engineer websites and web applications in a more user-friendly way. With a stress on tool support for web engineering environments, it discusses how a systematic approach to development can improve the usability of the result. Together with appropriate changes to the methods and models, the developed tool concepts can improve usability without significant additional work for the web developer.

1.1 Motivation: Achieving Good Web Usability is Becoming Harder

> **Summary:** Good web usability is important because the web is used for more and more aspects of our life. At the same time, different output devices and developments such as AJAX make it hard for the developer to ensure good usability in all cases.

Today, the World Wide Web is probably the most important interactive medium. After e-mail, it is the second most frequently used type of service on the Internet [Eimeren06OnlineStudy]. It is used regularly by a very large number of people for a diverse range of purposes, including searching for information (e.g. search engines, online databases), interacting with information systems (web shops, online banking) and interacting with other people (online auctions, online dating, Wikis, chat).

The importance of website usability has increased steadily over time. In the early years of the World Wide Web, websites were primarily used for information retrieval, i.e. to search for information and to browse pages in a relatively passive way.

This situation changed when the web was beginning to be used in a more interactive way. In particular, the advent of online shops motivated website operators to make their pages as easy to understand and to use as possible – a customer who was willing to buy an item in a web shop but unable to do so because of errors in the user interface would represent a financial loss.

In more recent times, the user interfaces that are built using web technology have become more and more complex. Using AJAX technology (Asynchronous JavaScript and XML, [Garret05AJAX]), web pages mimic the look and feel of normal desktop GUI applications, allowing more complex interaction with web pages than previously. At the same time, these new rich Internet applications (RIAs) are still quite different from desktop applications in some aspects, which can be a cause for confusion among users. Additionally, at a technical level the web developer is faced with flexible and versatile concepts to implement the GUI – this makes it easy to leave established interaction concepts and metaphors behind. Together with the absence of established de-facto GUI standards for AJAX applications, this makes it challenging to develop applications which are easy to use.

Another phenomenon in today's World Wide Web is the growing diversity of devices. For many years, a desktop PC was the standard for web browsing, and the web developer could be fairly confident that main properties of the rendering device (screen resolution, processing power, available input devices etc.) were always similar. However, mobile devices have been gaining importance, to the point that they are now the primary way to access the web in some parts of the world. Nowadays, web browsing facilities are also built into other devices of varying form factors. For example, television sets feature a large screen area, but not necessarily a high-resolution one, and the type of input device (remote control) is very limited compared to the keyboard and mouse of a desktop PC.

Finally, over time not only the rendering technology and devices have diversified, but also the audience. The World Wide Web gradually turned from an exotic, technically oriented medium into a mass medium – today, users of the WWW include virtually all age classes, educational backgrounds and fields of interest.

As shown in a representative study for the population of Germany in the years 1997 to 2006 [Eimeren06OnlineStudy], gender differences are not as pronounced today as they used to be – in 2006, some 67% of all men and 52% of all women used the Internet at least from time to time. Whereas younger users are much more active (97% of teenagers aged 14 to 19 were online), in recent years the number of older Internet users has also increased dramatically: In the year 2000, only 22% of those aged 50 to 59, and only 4% aged 60 or more used the Internet from time to time – in 2006, this increased to 60% and 20%, respectively. In a similar vein, Internet usage is not restricted to social status or regular income from work; 28% of all retired/non-employed Germans were Internet users in 2006. Details of the study suggest that the majority of the users' Internet activities involves WWW usage.

All in all, many factors have an influence on how users perceive a website and thus whether they will be able to use and understand it. For a web developer who wants his site to be easily usable for as many visitors as possible, this poses a problem. In practice, several different approaches are imaginable to achieve acceptable usability:

- The website is designed for and tested on all devices using as many users as possible. In practice, this is not feasible – the cost of large-scale usability tests is prohibitive, important types of users may still be missed and it is almost impossible to test on all past and present combinations of hardware and software.

- Based on an analysis of the expected audience of the site, the context in which it is used etc., usability testing is performed with “typical” test users and devices. This way, a reasonable compromise can be reached. For example, the tests can be designed to cover 90% of all expected users of the site with comparatively little effort. However, there is a certain risk that the *actual* users (or devices used) will differ from the expected ones. In this case, the site’s failure to provide good usability may prevent the site from establishing itself for new user groups, and in a negative feedback loop, the lack of users of a certain type will prevent the webmaster from noticing that his website could be of interest for this type of user.

- With the previous point in mind, it may be advantageous to perform basic usability testing for a somewhat wider range of users, devices etc. than what is shown by an analysis. However, a stage is quickly reached where the effort necessary for wider support appears in no way justified for the small number of prospective users. For example, it may be necessary to purchase devices for testing or to invite people from another continent to one’s usability lab. Furthermore, the the web developer may only have limited expert knowledge about important usability issues in these special areas.

In conclusion, the WWW has become a mass medium with a multitude of different types of users who access web pages in diverse situations using very different devices. In the light of these preconditions, it will be hard for a web developer to ensure good usability of his website, and compromises are inevitable, e.g. in the form that more stress is put on usability testing for some types of visitors than for others.

1.2 Proposed Solution: Improved Web Development Tools, Models and Methods

Summary: The tools introduced in this thesis automate some aspects of the web development process, which makes development cheaper and faster. At the same time, expert knowledge built into the tools ensures that the resulting website is usable. Additional steps like model creation will only be performed if they are easy to achieve. Thus, good usability of the tools for model creation (and similar tasks) is also an issue – all tools should be seamlessly integrated into an IDE.

The insights of two different branches of research are taken into account in this work, with the aim of combining them to their mutual benefit. The term *user-centered web engineering* describes this combination of the two fields of study:

- On one hand, *web usability research* has come up with rules and guidelines to make web applications more usable, but does not provide integrated tool support for this task.

- On the other hand, *web engineering* solutions use powerful MDA (model-driven architecture) tools to quickly and systematically develop a working prototype or implementation, but do not pay particular attention to the usability of the resulting application.

The primary concern of user-centered web engineering is to address the problems outlined in the previous section by improving the usability-related tool support available to the web developer. However, it is not possible to restrict oneself to tool support only: Later chapters of this thesis will elaborate on the fact that improved tools are not enough and that changes and improvements to the employed models and methods are also necessary.

The term “model” has its origin in the software engineering or (when restricting oneself to web-based applications) the web engineering domain. The model of a system is a description or specification of that system and its environment for a certain purpose, it is often specified by combining drawings and text [OMG03MDA]. In web engineering, different models are usually employed to describe the navigational structure of a site, the presentation of a single type of web page, and the concepts the web application works with (section 3.2). The last of these models, the conceptual model (which, for example, might include a `Book` class for an online library application) should not be confused with the term “conceptual model” as it is used in the HCI community. In that community, it describes the model that forms in a user’s mind as a result of how an application’s user interface is organized. Depending on the application, an overlap may or may not exist between it and the concepts handled by the application’s implementation.

The aim of this research is to make the task of creating a website or web-based application cheaper for the web developer, both in terms of the time necessary to perform the development task and in terms of the money that is spent on usability equipment, payment to test users etc. (Note: While the singular form of the term “web developer” is used throughout this work, this is also meant to include teams of developers who work together.)

It is possible to reduce costs by supplying the developer with special programs which automate some of his work. These programs are tailored to the different steps of the product lifecycle. They can often avoid manual, repetitive steps, complicated setup procedures or studying of guideline documents. For example, one of the main contributions of this work is a tool which makes it easy to perform usability tests of a web application in a variety of situations (see chapter 7).

A basic difficulty with automated tools which aim to improve usability is that “improved usability” inherently includes the notion of a human who interacts with the system. It would be unwise to assume that a tool can always predict the reaction of this human algorithmically. As a consequence, only some of the proposed tool concepts are capable of improving a website’s usability in a fully automatic way. For other tools such as the example above, there is no attempt to eliminate essential manual steps. In the case of the example, an analysis of the development task revealed that user testing involves steps which *must* always be performed manually, but that a user test includes other work, and the tool eliminates many non-essential manual steps.

A second goal of this research is to ensure that despite the development cost reduction, the resulting website will have better usability. As described in section 2.1, good usability means that a user can reach their goal efficiently and will be satisfied with the result. Over time, the publication of different sets of guidelines (section 2.2) has made this very general definition more concrete. While the growing amount of literature is a positive trend in general, a normal web developer cannot be expected to follow it closely and to be an expert in all areas of web usability. Thus, some of the concepts presented in this work are based on the idea that the developer should rely on expert knowledge about web usability. This knowledge can be included in the tools that he uses. For example, the tool from chapter 6 is intended to be run continually to watch the website while it is under development, verifying its quality using a set of algorithms. As soon as the developer is about to make a mistake, he can be alerted to the fact that his action would violate a certain usability guideline. This follows the idea from [Brajnik00AutoWebUsab] which proposes to perform automatic usability evaluation very frequently during development. A development environment can also help the developer in the sense that it can support planning of aspects of the development method which are essential for usability, such as user testing, and it can remind him of the necessity to perform these steps.

Another aim of this work follows from the first two: In order to achieve reduced development costs and better usability of the result, it is necessary that the tools and procedures introduced in this thesis are fit for the task. This implies that the tools themselves must be easy to use and that any additional work for the developer, e.g. creating a slightly more elaborate model than previously, must pay off at a later stage of development. In other words, while the usability of the resulting website is the primary focus, the usability of the development environment is also an important factor which must not be disregarded. This is particularly relevant for less proficient web developers, who may have neither experience with model-based development nor great expertise in the area of web usability.

As shown later, the existence of models can be a significant advantage for automated usability validation. On the other hand, model-based development and model-driven architecture (MDA) approaches are not commonplace today when websites and web applications are created, even less so than for the development of desktop GUI software or other, non-web programs. Thus, it is necessary to convince the developer of the benefits of model-based concepts. This will only be possible if he is guided by user-friendly authoring tools which make the creation of models a part of the development process. It is the task of an integrated development environment (IDE) to seamlessly integrate model creation into the workflow, so that it is not seen as an additional burden by the user. A complete implementation of an integrated development environment with usability tool support is not presented in this work, but the different lower-level parts (models, tools and methods) which comprise it are discussed in detail. In addition to this, the discussion in section 2.3 and chapter 4 provides thoughts and insights on what such an IDE would look like.

1.3 How This Thesis is Organized

This thesis does not provide a complete description of the steps that are necessary to plan, create, test and maintain a website or web application, since many aspects of development are addressed in the general web engineering and HCI literature already [Dumke03WE] [Constantine99Software]. Instead, its aim is to extend established practices and also provide ideas for appropriate tools in order to improve the usability of the final web application. This is achieved as follows in the chapters of the thesis:

- Existent work is analysed in the different fields relevant to the thesis. Chapter 2 looks at the area of web usability, in particular attempts to create usability guidelines and existing tools which aim at improving the usability of web pages. In chapter 3, web engineering solutions are examined with regard to the models, method steps and tools they provide, with a stress on models of information related to the users, and tool support for usability.

- Chapter 4 takes a look at each step of the development process, from the requirements analysis phase to the final testing and maintenance phase, and presents ideas on how additional tool concepts could help the developer to achieve significant usability improvements with little additional work for him.
- In chapter 4, it also becomes clear that some improvements (for instance in the area of automatic web page usability evaluation) are only possible if the tools are provided with information about the users, their intentions and the way they access the web application. For this reason, chapter 5 proposes extensions to the existing web engineering models which are capable of accurately representing these facts.
- Finally, chapters 6 and 7 describe two tool concepts in greater detail, including prototype implementations and a comparison with related efforts: The tool Wusab is a model-based usability and accessibility validator for web pages, and UsaProxy is intended for tracking the interaction of users with web pages during a user test.
- Finally, chapter 8 presents a summary and conclusions, and highlights contributions made by the thesis as well as areas of future work.

Chapter 2

Usability on the Web

2.1 Usability and Accessibility

Summary: Usability and accessibility are not the same, but they are related concepts. For example, an AJAX website can be very usable, but inaccessible to blind people. When the aim is to ensure good usability, accessibility cannot be disregarded as it is a precondition for efficient solving of tasks.

Usability

The term usability is often defined as the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use [ISO9241]. Furthermore, the following seven principles are listed in the ISO standard as indicators of a well-designed user interface (see also [Heinsen03UsabPrakt, p. 82f]):

- Suitability for the task: The system provides the required features. Tasks are performed automatically wherever possible, e.g. there is support for any repetitive tasks.
- Self-descriptiveness: The current state and possible ways to proceed are always clear.
- Controllability: The user can influence what happens next and in what order.
- Conformity with user expectations: No surprises – the system's behaviour is consistent.
- Error tolerance: The system alerts the user about irreversible actions, or undo is possible.
- Suitability for individualisation: The user interface can be adapted to the user's language, cultural background and their skill level.
- Suitability for learning: The system does not assume a trained expert user, it aids the user in discovering advanced concepts.

In the context of the World Wide Web, these principles are applied to the interaction between a user and the pages displayed in their web browser. As explained in the previous section, some aspects of the web platform introduce additional challenges. For example, in contrast to desktop GUI applications, the web developer must support a wider variety of different input and output devices, and he must take some special properties of the web platform into account, such as the request-response paradigm of the HTTP protocol.

Accessibility

The World Wide Web Committee (W3C), whose Web Accessibility Initiative (WAI) has been active since the late 1990s, defines web accessibility as follows:

> "Web accessibility means that people with disabilities can perceive, understand, navigate, and interact with the Web, and that they can contribute to the Web." [W3C-WAI, Introduction]

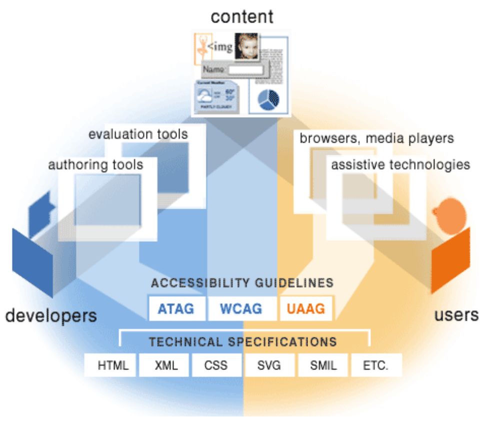
In other words, whereas *usability* concentrates on the task the user has to perform and the way achieval of the task is supported by a user interface, the primary point of good *accessibility* is that web content is offered to the visitor for consumption and interaction in a way which does not impose any barriers to them.

A central aspect of accessibility is the thought that not differences of the devices or other technology prevent the user from accessing the content, but that differences between humans are responsible for problems – in particular, physical disabilities of the users. Interest in accessibility was probably influenced most by legislation (e.g. [BITV02]) in several countries which aimed at providing equal opportunities for disabled persons. However, this does not imply that only a small minority of disabled people are the focus. Instead, accessibility efforts also include relatively common disabilities, such as the different forms of red-green colour blindness which affect about 10% of the male population.

It should be noted that usability and accessibility are not the same: Some concepts, like the use of AJAX technology to achieve drag and drop functionality, will improve usability for most users because with their help, tasks can be completed more efficiently. On the other hand, in the case that the AJAX website does not provide an alternative, non-AJAX version of a web application, the site will be rendered inaccessible to a blind visitor who uses a screen reader.

Still, a dependency exists between the two concepts: Problems with accessibility usually have severe effects on the usability of a site. Thus, trouble-free accessibility can be considered a precondition for good usability, and it does not make sense to exclude accessibility issues from any investigation about usability issues. As a consequence, some sources (e.g. [Ivory03WebEval]) use the term *web quality* to refer to the analysis both of usability and accessibility issues.

For the reasons outlined above, this thesis focuses both on accessibility and usability issues, and the associated models and tools introduced in later chapters attempt to address both types of problems. In the remainder of this work, the term "usability issue" will often be used in a somewhat inexact manner to include not only direct issues which affect a website's usability, but also accessibility problems which would prevent one of the target audience groups from using the website efficiently, or which would cause the user to be dissatisfied with the results.



Source: `http://www.w3.org/WAI/intro/components`, accessed 19 June 2007

Figure 2.1: The Web Content Accessibility Guidelines are the central part of W3C's Web Accessibility Initiative.

2.2 Accessibility and Usability Guidelines for Web Pages

Summary: A number of guideline documents exists. The web developer should follow them to achieve good usability/accessibility. The most influential source are the W3C's WCAG (Web Content Accessibility Guidelines). Types of guidelines range from technical (e.g. nesting of tags) to abstract (e.g. how to write prose on web pages). Automatic usability validation has not been the focus of most guideline authors.

This section gives an overview of the most important sets of usability guidelines. The amount of available literature is quite large and web developers may not be interested in achieving good usability in all cases (see section 1.1). Consequently, the approach taken in later chapters of this work is to automate verification of guidelines as much as possible by embedding expert knowledge into programs. With this in mind, the following analysis of existing guidelines also looks at whether guideline authors had automation in mind.

2.2.1 W3C WCAG 1.0

The Web Content Accessibility Guidelines 1.0 [W3C-WCAG1] reached the status of a W3C recommendation in May 1999. Although work on the next version of the guidelines is progressing, at the time of writing they are still the official, "stable" version of the main document produced by the W3C's Web Accessibility Initiative [W3C-WAI].

The WCAG include guidelines, hints and tips primarily for creators of web pages, but also for makers of web authoring software – the latter is necessary because a growing number of developers no longer creates HTML pages manually, but is assisted during the page creation task by dedicated programs. If the programs make accessibility mistakes in the output the generate, the probability that these mistakes remain in many live websites is high. Further accessibility-related advice for page creation software is also available in the Authoring Tool Accessibility Guidelines, ATAG, which are also part of W3C WAI; see figure 2.1.

The purpose of the guidelines is twofold: On one hand, they aim at making content accessible to persons with disabilities. However, on the other hand, they also want to improve accessibility for "normal users" – in this case, the ability of the visitor to access a website may be restricted due to properties of his output device. For example, the visitor might be using a mobile device to access the site, or he might be using a browser which is standards-compliant, but does not feature some of the extensions of the most popular browsers.

Guidelines

The following 14 rules are the central part of the WCAG 1.0:

1. *Provide equivalent alternatives to auditory and visual content.* For the benefit of visually or otherwise impaired users, use e.g. the `alt` attribute when embedding images in the page.
2. *Don't rely on color alone.* Information should not be encoded exclusively as colour values; the user may be visually impaired, or he may be using a device without colour support.
3. *Use markup and style sheets and do so properly.* Use syntactically correct markup. Furthermore, use markup rather than visual formatting (e.g. use `<em>` rather than `<i>`).
4. *Clarify natural language usage.* Use markup to specify that the text is e.g. English or German.
5. *Create tables that transform gracefully.* Use tables only for data representation, and use markup to identify row/column headers.
6. *Ensure that pages featuring new technologies transform gracefully.* Pages should still work if an old browser is used, or if the browser does not support e.g. JavaScript.
7. *Ensure user control of time-sensitive content changes.* Avoid movement, blinking, automatic refresh of the page and similar techniques.
8. *Ensure direct accessibility of embedded user interfaces.* For example, accessibility must be ensured for a Java applet embedded on the page.
9. *Design for device-independence.* Inputting data on the page should be possible regardless of the input device used, and the information provided by the page should be intelligible regardless of the output device.
10. *Use interim solutions.* As a concession to the poor support for some web standards in 1999, workarounds are allowed for some problems with browsers.
11. *Use W3C technologies and guidelines.* Whereas W3C's standards have always been designed with accessibility in mind, this cannot be said of other formats like PDF or Flash, so they should be avoided.
12. *Provide context and orientation information.* Use headings and nested lists when appropriate. Generally, use markup to specify relationships between parts of the page.

13. *Provide clear navigation mechanisms.* Do not use “click here” as link text, provide a site map or similar, provide a search facility, add metadata to pages, e.g. with RDF.
14. *Ensure that documents are clear and simple.* Page layout should be consistent, the sentences easy to understand and the language simple, e.g. for people with learning disabilities.

Together, these rules try to address different aspects of making web pages accessible. Very roughly, the three types of rules below can be distinguished. However, rules often fit in more than one category:

- Many guidelines (1, 2, 7, 9, 11, 13, 14) concentrate on the content itself, how it should be organized and when to provide alternative versions of the same content for different user groups or devices.
- Other guidelines (3, 4, 5, 12) advocate annotating the text with abstract information rather than using HTML’s more “direct” visual formatting facilities.
- Finally, there are guidelines which address compatibility issues and other, more technical details (6, 8, 10).

Priority Classes and Conformance Levels

The WCAG 1.0 recommendation defines three different priorities. Each of the 14 guidelines above is accompanied by a list of so-called checkpoints, i.e. more a concrete list of examples for the general guideline, and each checkpoint is assigned a priority individually. Priority 1 checkpoints *must* always be satisfied, priority 2 checkpoints *should* be satisfied, and priority 3 checkpoints *may* be satisfied.

Three conformance levels exist, they correspond to the priority classes above. The simplest level, “Conformance Level A”, is reached if all priority 1 checkpoints (i.e. the most important ones) are satisfied. For “Conformance Level Double-A”, priority 1 and 2 must be satisfied. Finally, for “Conformance Level Triple-A” all checkpoints must be satisfied.

Automated Validation of Guidelines

How easy is it to validate with a program whether a website conforms to WCAG 1.0? The document itself contains a short appendix A on validation which hints that automated validation is possible for a portion of the checkpoints, including syntax checks for HTML/CSS, readability measurements etc. Nevertheless, the appendix also makes it clear that many aspects can only be verified by a human, for example the clarity of the content.

A combined approach is recommended to the developer: Automated tools should be used for fast results, then further manual tests with different browsers/devices should be conducted, and finally experts (e.g. a professional editor) and people with disabilities should be invited to evaluate the page. The W3C note “Core Techniques for Web Content Accessibility Guidelines 1.0” which accompanies the WCAG 1.0 document contains further suggestions. Apart from W3C’s own syntax validation services for HTML and CSS, it cites the Bobby automated validation tool

[WatchfireBobby] (no longer available as a free service today) and the WAI Evaluation and Repair Working Group of the W3C. This working group concerns itself with issues related to automatic validation, such as the development of a machine-readable language for reporting of test results, Evaluation and Report Language 1.0 [W3C-EARL]. However, no detailed description on how to implement automated tests of the guidelines can be found on its web pages.

2.2.2 Legislation related to WCAG 1.0

Because the WCAG 1.0 were released as early as 1999 and because they were backed by an organisation which was considered the authority for web-related standards at the time, they had an influence on legislation passed in different countries during the following years. Attempts of governments to adapt their equal opportunities legislation to the WWW required them to look for rules on how to best allow disabled people to access information online. The guidelines issued by the W3C contained helpful expertise, so in many cases the relation between the WCAG 1.0 and the country-specific legislation is quite visible. While the legislation only applies to organisations under government control, commercial enterprises were also soon beginning to consider conformance as a requirement when creating new websites. Tool-supported validation is not a subject of the different laws and regulations.

Germany: BITV

In Germany, the *Barrierefreie Informationstechnik-Verordnung* (regulation for barrier-free information technology) was ratified in the year 2002 [BITV02]. It required all websites run by government organisations, universities and other official bodies to adhere to a number of guidelines by late 2005. The WCAG 1.0 are explicitly cited as the basis of the rules, and like in the W3C document, different classes of priorities are present; priority I guidelines must be implemented for all pages and priority II guidelines must additionally be considered for central navigation pages and start pages. The individual guidelines are very similar to WCAG 1.0, but BITV tries to find wording which is easier to validate in an objective way: In most cases, a human validator will have no problems determining whether a guideline is fulfilled.

USA: Section 508

Equal treatment of humans with physical or mental disabilities is also the goal of the US Rehabilitation Act of 1973 [USSection508]. Section 508 was added as an amendment in 1986 and required federal agencies to use accessible information technology. However, information on web accessibility was only added in 1998 with another amendment, and the final version was published in the year 2000, about 1.5 years after WCAG 1.0. The regulation consists of a number of parts – the one of interest in the context of this work is subpart B, “Technical Standards”, section 1194.22, “Web-based intranet and internet information and applications”.

Part 1194.22 of the regulation contains 16 rules, (a) to (p), about web accessibility, most of them bear close resemblance to the WCAG 1.0 guidelines. A more detailed look reveals small differences: Section 508 is more lax in many aspects; for example, it does not require page authors to identify changes in the natural language of the text (WCAG guideline 4, checkpoint 4.1). On the other hand, it is also more specific in some cases. Rule (p) has no equivalent in WCAG 1.0; it states that when a "timed response" is necessary by the user (i.e. there is an input page which automatically redirects to another page after some time), the user must be able to request more time until he has to perform his response.

All in all, section 508 places fewer requirements on the web pages than WCAG 1.0. On the other hand, its scope is broader, as it governs not only web design, but also operating systems and desktop GUI software, telecommunication and other related technologies. Due to the fact that most major IT companies originate from the US, section 508 has also influenced products available outside the US: These companies are required to adhere to the regulation when they supply equipment, software and services to organisations run by the government.

Italy: Stanca Act

The Stanca Act [StancaAct04] (named after the Italian minister of Innovation and Technologies) is a law which has the aim of supporting access to information for elderly citizens and people with disabilities. It addresses the problems of these parts of the population from several different angles, including training programmes for developers of IT systems and books for school libraries. The law is intended to comply with related international specifications and designed to be easy to update in the future. It was passed in late 2004, the guidelines for public adminstration websites were defined in July 2005 in a ministerial decree. These websites then had one more year to achieve compliance with the guidelines. A logo can be used to show conformance to the law, several variants exist for different conformance levels.

The website guidelines in Annex A of the ministerial decree consist of 22 "technical requirements" which have to be fulfilled. They include most WCAG 1.0 priority 1 guidelines. Each requirement cites the related sections both in WCAG 1.0 and US Section 508, which makes the close relationship to these sets of guidelines obvious. In addition to the technical guidelines, Annex B also specifies 12 less technical assessment criteria and promotes the evaluation of a website by an expert. The 12 criteria add some further aspects of WCAG 1.0 to the Stanca Act rules, such as the idea of WCAG checkpoint 14.1, "Use the clearest and simplest language appropriate for a site's content". The criteria are as follows:

1. *Perception*: Information and possible actions must be perceptible.
2. *Comprehensibility:* Information and commands must be understandable.
3. *Operability:* Selection of a suitable action must be possible.
4. *Coherence:* Meaning of symbols/messages/actions must not change.
5. *Safety:* The well-being of the user must be protected.
6. *Security:* Management of transaction and data must be reliable and secure.
7. *Transparency:* The current state of the system must be clear.
8. *Learnability:* The environment must be easy and quick to learn.
9. *Help and documentation:* Help must be available for the current context.
10. *Error tolerance:* Errors must be explained and appropriate actions offered.

11. *User-friendliness:* The interests of the user must be taken into account.
12. *Flexibility:* Individual user preferences must be followed.

With these qualitative properties, the Stanca Act goes beyond the types of rules that are still referred to as *accessibility* guidelines, and also becomes a regulation for *usability*. The points above are reminiscent of the ISO definition for usability with its "user satisfaction" idea (criterion 11, "user-friendliness" above), but also of Jakob Nielsen's definition of usability [NielsenAlertbox, 25 August 2003], which he bases on the five properties learnability, efficiency, memorability, errors and satisfaction.

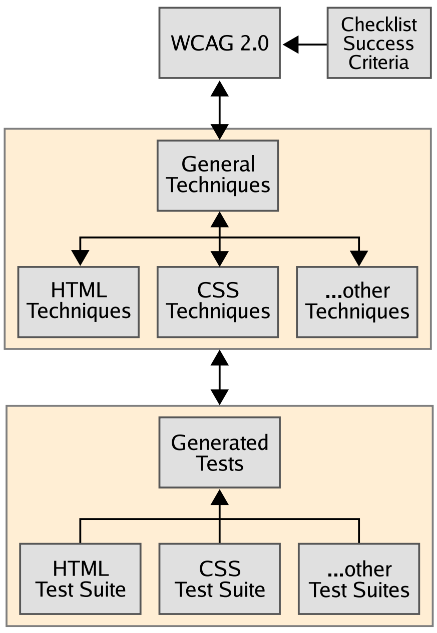
2.2.3 W3C WCAG 2.0

Looking at the details of the WCAG 1.0 document, its age becomes apparent in many places. For instance, user interface issues in browsers have been addressed since WCAG 1.0 was released: Popup blockers allow the user to turn off unwanted opening of windows, and tabbed browsing can further reduce the amount of confusion encountered by users as there is now visual feedback about how many pages are opened in the browser. Furthermore, compatibility issues have been resolved – CSS and client-side imagemaps are now supported by all browsers. Together with advances in many of the technologies used on the web, and practical difficulties when working with WCAG 1.0, it was soon felt that an effort should be started to update the guidelines and to adapt them to current best practices – WCAG 2.0 [W3C-WCAG2].

Requirements

Since the release of WCAG 1.0 in 1999, the WCAG Working Group of the W3C collected feedback about the guidelines and their applicability. Ultimately, the Working Group decided to specify six additional requirements for version 2.0: [W3C-WCAG2req]

1. The rules laid out in the document should be independent of the exact technology used. While in version 1.0 guidelines were mostly applicable to HTML documents, more generic wording should now allow their use with more than one markup language or document format, such as CSS, SML, XML and others.
2. Clearer language was necessary for the conformity requirements, and verification of the guidelines must be possible. While the note does not explicitly mention automatic validation, this goal is nevertheless a positive development in the context of this thesis – clearer criteria can lead to better chances of automating at least some guidelines.
3. The rules should be structured in a clearer and simpler manner.
4. The audience of the WCAG 2.0 should not be restricted to web developers, but should also include designers, authoring solution providers, teachers in the domain of accessibility, and developers of evaluation or repair tools. Again, this last point is a positive development – as outlined in the section on WCAG 1.0 above, no such goal existed for that version of the guidelines.



Source: `http://www.w3.org/2004/09/wai-nav/intro/wcag20.html`, accessed 25 June 2007

Figure 2.2: WCAG 2.0 is built upon a number of accompanying documents. The Techniques documents provide explanations and examples of the main WCAG guidelines, the Test Suites contain sample files e.g. for correct and incorrect HTML code.

5. The advantages of accessible content should be clearly identified, and the new WCAG version should address the different types of disabilities which can be encountered.
6. Backwards compatibility with WCAG 1.0 and forwards compatibility with new technologies should be achieved.

After several years, work on the WCAG 2.0 guidelines is expected to come to a conclusion with the publication of a W3C "Proposed Recommendation" in late 2008.

Figure 2.2 gives an overview of some of the documents that make up the standard. A look at them shows that the Working Group has done its best to address the six requirements mentioned above. This has advantages as well as disadvantages – while the WCAG 2.0 are certainly a significant improvement over version 1.0 as far as clarity and preciseness are concerned, they have also become harder to grasp for the reader.

For the average web developer, the main problem will probably be the sheer size of the different documents. In fact, the working group has felt it necessary to produce a detailed web page titled "Overview of WCAG 2.0 Documents" which gives a description of documents which are themselves overview documents, such as the "WCAG 2.0 Quick Reference" and "Understanding WCAG 2.0". Compared to version 1.0, accompanying documents like the Authoring Tool Accessibility Guidelines [W3C-ATAG2] have become much more voluminous. The combined size of the documents is several times that of WCAG 1.0. This means that a developer may not have the time to read all of them, or that he may lack enough interest to try to understand all the details. Furthermore, because the language is more general than with WCAG 1.0 (see requirement 1 above) and because of the large number of cross-references, the text may be harder to understand for some readers.

The Four Design Principles

Whereas with WCAG 1.0, there were guidelines with checkpoints of different priorities, WCAG 2.0 is centered around only four design principles. The 12 guidelines are assigned to one of these principles, and each guideline comes with so-called success criteria for the different conformance levels, A, AA, or AAA. Here is a brief summary of the principles and the associated guidelines:

Principle 1: Perceivable – Information and user interface components must be perceivable by users. This principle is concerned with alternative representations of non-text content and with multimedia content. WCAG 1.0's guidelines 1, 2 and 3 can be found here in a slightly different form.

1.1 *Provide text alternatives for any non-text content so that it can be changed into other forms people need such as large print, braille, speech, symbols or simpler language.*

1.2 *Provide synchronized alternatives for multimedia.* For example, provide captions, an audio or full text alternative or sign language.

1.3 *Create content that can be presented in different ways (for example spoken aloud, simpler layout, etc.) without losing information or structure.*

1.4 *Make it easier for people with disabilities to see and hear content including separating foreground from background.* This also includes the ability to turn off any audio that is played, and to change the size of any text that is displayed.

Principle 2: Operable – User interface components must be operable by users. The general idea of this principle is that all interface components on the web page should be accessible even if the user has a non-standard input device (e.g. no mouse), problems with providing input (e.g. takes very long to input text) or similar. WCAG 1.0 guidelines 7, 8, 9, 12 and 13 are represented in this principle.

2.1 *Make all functionality available from a keyboard.*

2.2 *Provide users with disabilities enough time to read and use content.* If there is a timeout, the user must be able to extend it. Auto-updating information can be paused, and blinking content can be avoided.

2.3 *Do not create content that is known to cause seizures.* This guideline is intended to avoid problems for conditions like epilepsy.

2.4 *Provide ways to help users with disabilities navigate, find content and determine where they are.* The suggested criteria (descriptive page title and link text, using headings to indicate sections etc.) will also make navigating the site easier for people without disabilities.

Principle 3: Understandable – Information and operation of user interface must be understandable by users. This principle concentrates on good readability of text content and a consistent organisation of interface components. WCAG 1.0 rules 4 and 14 can be found here.

3.1 *Make text content readable and understandable.* The user must be able to understand the content of the text. For their browser, it must be possible to determine the human language used on the page and similar properties.

3.2 *Make Web pages appear and operate in predictable ways.* For example, the order of navigation menu items must not change.

3.3 *Help users avoid and correct mistakes.* Errors must be described to the user, confirmation is required for legal and financial commitments, and reversing an action should be possible.

Principle 4: Robust – Content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies.

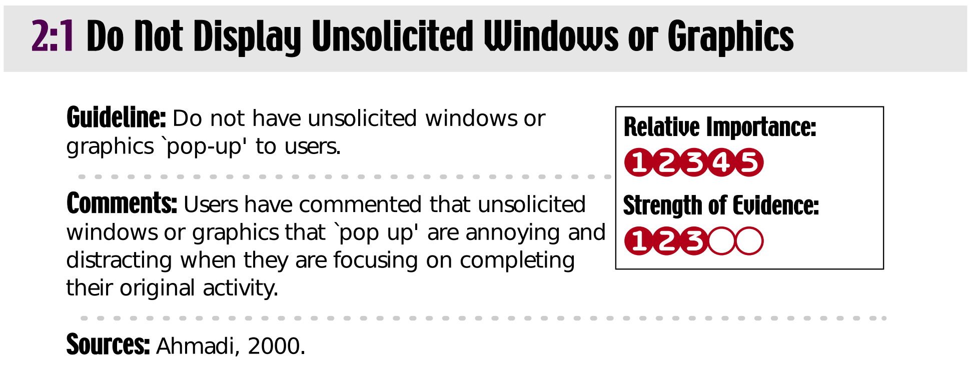
4.1 *Maximize compatibility with current and future user agents, including assistive technologies.* Markup must be syntactically correct, and any custom controls that the developer implements using scripting must work with assistive technologies.

Conformance

As with WCAG 1.0, different conformance levels exist. However, the conformance rules have changed slightly with version 2.0: Level A conformance is reached when all Level A success criteria are satisfied. For Level AA conformance, all Level A and Level AA must be satisfied. For Level AAA, all success criteria (A, AA and AAA) must be satisfied. To achieve conformance, sites may offer alternate, conforming versions of pages which are not accessible. In addition to the success criteria, a number of additional conformance requirements must be fulfilled. For example, if a process (such as buying an item in a shop) requires the user to traverse several pages, then all these pages must fulfil the requirements of WCAG 2.0 in order for any of them to achieve conformance.

The WCAG 2.0 document also specifies details on how to claim conformance with the guidelines. A full conformance claim needs to include details like the date, the exact version of WCAG 2.0 used, the URL(s) and any server-side technologies which were employed to achieve conformance. Moreover, if a site only achieves conformance with a certain level, but fulfills some success criteria of higher levels, it may add this information to the conformance claim. This way, the working group has tried to address criticism with WCAG 1.0 that there was no incentive for sites to fulfil more than what was absolutely necessary for a certain conformance level.

If it is not clear whether later changes to the web pages will cause accessibility problems, a statement of partial conformance is also possible – effectively, it claims conformance if some parts of the page are removed, e.g. snippets of HTML code which are automatically included from external sources.


2:1 Do Not Display Unsolicited Windows or Graphics

Guideline: Do not have unsolicited windows or graphics `pop-up' to users.

Comments: Users have commented that unsolicited windows or graphics that `pop up' are annoying and distracting when they are focusing on completing their original activity.

Sources: Ahmadi, 2000.

Relative Importance: ❶❷❸❹❺

Strength of Evidence: ❶❷❸○○

Figure 2.3: One of the 209 guidelines of the Research-Based Web Design & Usability Guidelines. [HHS06WebGuidelines, page 34]

2.2.4 Research-Based Web Design & Usability Guidelines

Whereas other sets of guidelines were written by smaller teams of practitioners who typically formulated them based on personal experience, the US Department of Health and Human Services (HHS) chose a different approach for its Research-Based Web Design & Usability Guidelines [HHS06WebGuidelines]: The audience of this work includes both website designers/maintainers and the community of researchers whose interests are focused on this branch of human-computer interaction (HCI). Furthermore, the goal was not only to create a catalogue of rules which would result in “good usability” of the website, but to do this in a scientifically sound way. Finally, it was attempted to include in the guidelines the expert knowledge of the most important researchers in the field. The initial impetus for the work was the HHS’s wish to make its own websites usable for its target audience, which includes people with disabilities.

The approach taken by the authors of the guide was to first compile an extensive list of guidelines from existing sources and own experience. Next, duplicates were identified and removed, and conflicting guidelines were looked for and resolved where appropriate. The high quality of the final set of guidelines was achieved by including website and usability professionals at this point in a peer-reviewed process: 36 experts rated the relative importance of each guideline according to their personal opinion, and 13 experts working in a team rated the “strength of evidence” that a particular guideline would have a positive effect if implemented by a website. This strength of evidence rating is based on research results (e.g. formal studies, published papers with objectively measured findings), and the presence or lack of conflicting expert opinions with regard to the effectiveness of the guideline.

In the final 2006 edition of the guidelines, a total of 209 guidelines are presented – see figure 2.3 for an example. Each guideline is comprised of the following information:

- The short description of the guideline
- A “relative importance” rating from 1 (important) to 5 (very important) – guidelines which were not considered at least somewhat important were excluded;

- A “strength of evidence” rating from 1 (weak expert opinion support) to 5 (strong research support);
- A clearly worded description of the guideline;
- Comments for the guideline which explain the idea behind it or how to ensure that it will be met;
- At least one reference for the guideline;
- Where appropriate, an example which illustrates the point made by the guideline.

The guidelines are organized in 18 groups which correspond to the chapters of the book. The number of guidelines for each chapter is given below in parentheses, followed by one example for a guideline from the chapter:

Chapter		Example guideline
1. Design Process and Evaluation	(11)	*Understand and Meet User’s Expectations*
2. Optimizing the User Experience	(16)	*Do Not Display Unsolicited Windows or Graphics*
3. Accessibility	(13)	*Comply with Section 508*
4. Hardware and Software	(5)	*Account for Browser Differences*
5. The Homepage	(9)	*Show All Major Options on the Homepage*
6. Page Layout	(13)	*Place Important Items Consistently*
7. Navigation	(12)	*Provide Feedback on User’s Location*
8. Scrolling and Paging	(5)	*Eliminate Horizontal Scrolling*
9. Headings, Titles, and Labels	(8)	*Provide Descriptive Page Titles*
10. Links	(14)	*Use Meaningful Link Labels*
11. Text Appearance	(11)	*Use Mixed-Case for Prose Text*
12. Lists	(9)	*Order Elements to Maximize User Performance*
13. Screen-Based Controls (Widgets)	(25)	*Minimize User Data Entry*
14. Graphics, Images, and Multimedia	(16)	*Graphics Should Not Look like Banner Ads*
15. Writing Web Content	(11)	*Use Familiar Words*
16. Content Organization	(9)	*Facilitate Scanning*
17. Search	(9)	*Design Search Engines to Search the Entire Site*
18. Usability Testing	(13)	*Use an Iterative Design Approach*

The wide scope of the work becomes apparent from this list: It contains accessibility as well as usability guidelines, and the guidelines cover the entire development process, from the initial design phases to usability testing with the final website. A “practically oriented” approach to formulating rules is also noticeable. For example, unlike with WCAG 2.0, there is no attempt to choose wording for the guidelines which is as general as possible. Instead, a greater number of more detailed rules is listed. In general, the level of detail is much greater than for any of the other sources discussed in this chapter.

The guide does not focus on automation of usability validation. In the introductory section of the book, assistance by programs during guideline verification is not discussed. For some chapters, the presented guidelines are not suitable for automatic validation, for example chapter 1, “Design Process and Evaluation” – the intent of the guidelines is too abstract. For other rules, enough information is presented for an automatic check. For example, guideline 3:13 “Avoid Screen Flicker” contains exact values for the permitted frequency range of blinking content.

All in all, the Research-Based Web Design & Usability Guidelines are an extensive set of guidelines. They are probably a superset of the other guideline documents discussed in this chapter. On the other hand, at 290 pages the document is a rather large book – in practice, this can be a problem for web developers who do not attach great importance to web usability and who have little time as they have to meet a tight deadline. The authors have tried to address this by ordering the guidelines by decreasing importance for each chapter, but as they state in the introductory section, even the guidelines with a lower “relative importance” rating were still considered important by the team of usability researchers.

2.2.5 Web Style Guide

The Web Style Guide [Lynch02WebStyleGuide] is a collection of usability and accessibility guidelines and of hints on how to ensure them with an appropriate development process. The Guide, which is also available from `http://webstyleguide.com`, is written in the style of a textbook, with each of the eight chapters highlighting a different aspect of improving the quality of a website:

1. *Process:* Planning a site’s architecture, preparing the design and construction of the pages, how to market the site, evaluation and maintenance of the final website
2. *Interface Design:* What questions each page should answer, how to present navigation controls, ensuring the accessibility of the design
3. *Site Design:* Splitting up information into individual pages, structuring the website, typical elements of a website (e.g. link to home page, search)
4. *Page Design:* Visual design of pages (layout, headers, footers etc.), advice for cross-platform support, correct markup for accessibility
5. *Typography:* Structuring content e.g. with sections, separating design and content, recommendations for text layout which leads to good legibility, accessibility hints for text
6. *Editorial Style:* Writing for quick scanning by putting important information first and with appropriate markup (bullets, headings), correct use of links
7. *Graphics:* Desirable properties of graphics on web pages (e.g. recommended formats/resolutions), how to present different types of information with images, accessibility issues
8. *Multimedia:* When to use different types of multimedia content, balancing quality against accessibility of the content

At the end of each chapter, references to related work are provided. Furthermore, in most chapters a section is dedicated to accessibility aspects. However, the Web Style Guide also contains many suggestions related to usability.

Compared to related sets of guidelines, the document is not organized in a particularly strict way. Individual rules are harder to distinguish from one another as they are not assigned numbers or made distinguishable in other ways. No formal concept is introduced which helps developers to quickly check whether individual requirements are fulfilled – for example, a list of checkpoints similar to the WCAG is absent. This may make the guide appear to be less useful for usability-related advice than other sources, but that is not the case: Other sets of guidelines aim at providing an objectively verifiable set of rules which can be used e.g. as a part of a web design contract, a basis for legislation or a "proof" that work on usability has been carried out. The Web Style Guide has a different goal: It wants to provide relevant information to its audience in an easy-to-comprehend form, without burdening the reader with formal language. This is achieved by a writing style which can be understood without effort – examples are mixed with explanations of guidelines, which helps to keep the reader focused and interested. For this reason, the guide is also better suited than other standards to be read from first to last page.

As far as automated verification of guidelines is concerned, the Web Style Guide does not offer many insights of how to implement particular usability or accessibility checks. One example for a concrete constraint is the maximum limit of 50 to 75 kB of graphics data that is permissible on a page for modem users. On the other hand, e.g. the discussion of advantages and disadvantages of fixed-width and flexible page layouts in chapter 4 does not give any definitive advice on what should be preferred, so an algorithm cannot be formulated.

2.2.6 Further Resources

Numerous other books and websites are dedicated to issues related to web usability and accessibility, and offer guidelines and other advice for webmasters on how to improve their own sites. In the following, a number of examples for such resources are given.

Don't Make Me Think

Based on the Steve Krug's practical experience as a consultant, the book "Don't Make Me Think" [Krug06Dont] focuses on website design for improved usability and (to a lesser degree) accessibility. It describes the way that users perceive and operate a website. From these observations, it develops guidelines for website authors on how to organize the page layout, write the website content, design the navigation or perform user testing. Due to the entertaining style of the book, its content is easy to grasp for the reader. At the same time, most of the recommendations are not accurate enough to directly permit the formulation of algorithms which validate the guidelines.

useit.com: Alertbox – Current Issues in Web Usability

Jakob Nielsen's website with its bi-weekly alertbox column [NielsenAlertbox] is one of the oldest available sources of information related to web usability: The first issue was released in June 1995, at a time when the web was very different from today, and de-facto standards for website design and interaction were only beginning to emerge.

The Alertbox is organized as a series of short, independent articles which cover a wide range of web-related topics. From one Alertbox issue to another, there are differences with regard to the level of detail, ranging from very definitive rules e.g. concerning screen resolution

[NielsenAlertbox, 31 July 2006] to general observations like usability in science-fiction movies [NielsenAlertbox, 18 December 2006]. The amount of scientific background of any presented guidelines also varies significantly – both personal opinion and results of large-scale user tests can be encountered. Some of the more important Alertbox issues include:

- *Top Ten Mistakes in Web Design* – e.g. bad search, using PDF files, non-scannable text
- *Usability Testing With 5 Users* – even tests with only a few test users are useful
- *Reading on the Web* – users' habit of scanning pages and how to organize content for it
- *Usability of Websites for Teenagers* – teenagers differ from adults in their browsing habits

dev.opera.com/articles/mobile: Designing for Mobile Devices

Opera Software is the supplier of browsers for mobile devices, Opera Mini and Opera Mobile. In the developer section of the company's website, there is a set of articles with guidelines for web developers. Unlike other sources presented above, these guidelines focus on mobile devices rather than desktop PCs. Due to the comparatively large number of different HTML rendering engines, differing screen sizes and low network bandwidth, it is difficult to create web pages which render well on a large number of mobile devices. The guide "Making Small Devices Look Great" [OperaSmallDevices] contains advice on what markup to use and to avoid, on testing procedures during development and on the design of screen layouts for small screen sizes.

As already discussed in section 1.1, the increasing number of mobile devices with their varying properties makes manual testing especially laborious and tiresome. For this reason, any automated aid during development would be particularly useful. However, the Opera guides only contain advice with regard to manual testing of pages.

Web Design for Special Audiences

While the general accessibility guideline documents are founded on the idea that certain visitors of the site will only be able to perceive content in a way that is different from the majority of users, there are also specialized web pages with guidelines for particular disabilities. The following list shows some examples (accessed June 29th 2007):

- M. Okabe, K. Ito: How to make figures and presentations that are friendly to color blind people: `http://jfly.iam.u-tokyo.ac.jp/color/`

- M. Parise: Color Theory for the Color-Blind: `http://www.digital-web.com/articles/color_theory_for_the_colorblind/`

- American Foundation for the Blind: The Elusive Visually Impaired Audience: A Selective Profile: `http://www.afb.org/Section.asp?SectionID=43&DocumentID=1720`

- Mike Paciello: Making the Web Accessible for the Deaf, Hearing and Mobility Impaired: `http://www.samizdat.com/pac2.html`

- M. Bernard: Criteria for optimal web design (designing for usability): `http://psychology.wichita.edu/optimalweb/default.htm` – Advice backed by research results, on improving sites for children, elderly people, international visitors and potential e-commerce customers.

2.3 Existing Usability Tool Support for Web Developers

Summary: Numerous commercial and academic tools are available for improving website usability. They help during the content creation, the capture of usage data (both of "live" websites and during user tests) and its evaluation (e.g. generation of statistics, visualisation). Furthermore, tools exist for automatic critique of the HTML code, and for automatic repair of usability problems.

In this section, an overview is given of existing tool concepts which aid web developers in creating websites that are accessible and usable. This can take different forms – there are tools that only aim at improving a single aspect of a website, tools that support a certain phase of the development process, and tools which are intended to cover the entire development process. The available solutions range from proof-of-concept implementations of interesting research ideas to complete commercial software solutions which are the result of many man-years of development.

This section roughly follows the content lifecycle when giving attention to different types of tools. In section 2.3.1, it first lists different ways in which usability can be improved by tools the moment the web content is created.

Next, the tool-assisted analysis of existing web pages is discussed in sections 2.3.2 to 2.3.4. In an attempt to categorize the different ways in which the usability of websites can be evaluated, Ivory has developed a taxonomy of automated evaluation methods [Ivory03WebEval, section 2.2] [Ivory01Automate, fig. 1], which differentiates between approaches according to their *Method Class, Method Type, Automation Type* and *Effort Level* – see figure 2.4 for an overview. This

Evaluation Method

Method Class	**Method Type**	**Automation Type**	**Effort Level**
– Testing	– Log File Analysis	– None	– Minimal
– Inspection	– Guideline Review	– Capture	– Model
– Inquiry	– Surveys	– Analysis	– Informal
– Analytical Modeling	– GOMS Analysis	– Critique	– Formal
– Simulation	– Genetic Algorithms		
	⋮ *(many more)*		

Figure 2.4: The classification of usability evaluation methods from [Ivory03WebEval, p. 17] provides the basis for a systematic look at different concepts for usability tool support.

taxonomy is useful for distinguishing between different types of tools. Apart from the *Method Class*, the *Automation Type* (based on previous work by Balbo [Balbo95AutoEval]) is an interesting criterion. The following four types of automation are presented:

- Non-Automatic: Analysis by human usability experts without any assistance by programs
- Automatic Capture: Usage data (e.g. mouse and keyboard input) is recorded by software.
- Automatic Analysis: Programs identify possible accessibility and usability problems.
- Automatic Critique: Programs not only identify problems, but also suggest solutions.

The following is intended to convey the different possible ways of approaching the topic of usability tool support, and to contrast them with one another. It does not attempt to cover all of the available tools, as they are far too numerous. Later chapters will provide a more detailed analysis of existing tools for specific areas of usability-related tool support. For an overview of the related general topic of web application testing, see [Kappel04WE, section 7.6].

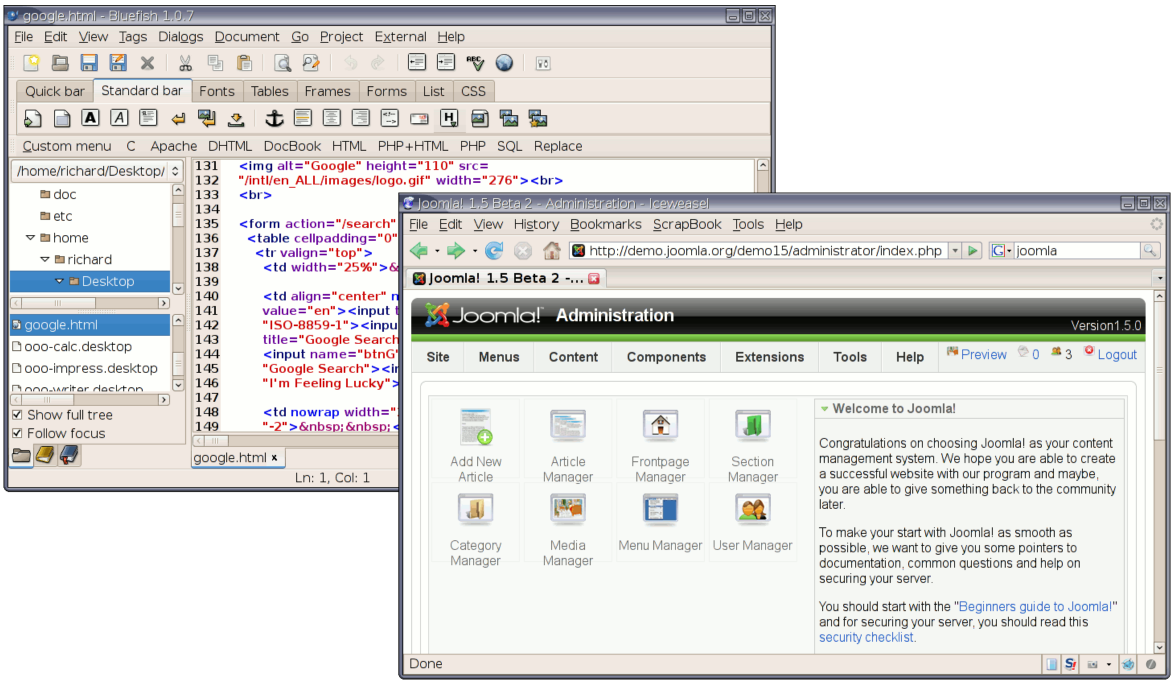
2.3.1 Generation of Web Pages

The functionality that is probably most frequently associated with web developer tool support is the creation and maintenance of the content that will be served by a website. Consequently, countless different tools are available for this purpose. Typically, a tool that is used during content creation will provide the initial HTML code, and the developer is unlikely to change any aspects unless they have a good reason to do so. Thus, there is a unique chance for the tool to "get things right by default", i.e. to create initial HTML code that already satisfies as many usability principles as possible. This way, the chances of a more usable result are higher even if the developer is not overly interested in usability, and not aware of the reasons why the supplied default is output in exactly that way.

Assistance During HTML Creation and Editing

Adherence to the HTML standard is a prerequisite for website accessibility. The reason is that incorrect syntax results in undefined behaviour of different browsers, which might even include a failure to render the page at all. Even though most browsers are relatively forgiving with regard to syntax errors, every effort should still be made to provide a syntactically correct HTML document.

Even in the case that the developer inputs much of the HTML code manually using a text editor with HTML support (e.g. Bluefish, `http://bluefish.openoffice.nl`, see figure 2.5), the editor can provide different template files with correct `<!DOCTYPE>` declarations and a basic structure of elements which the developer can base his work on. Furthermore, it can ensure correct nesting of HTML tags, warn about basic mistakes like missing `alt` attributes in `<img>` tags, or highlight other violations of the W3C HTML specification. More advanced editors allow the user to edit the page visually rather than at the HTML code level, so the editor can take further steps to improve the quality, e.g. by using CSS rather than HTML tables for the page layout.


google.html - Bluefish 1.0.7
File Edit View Tags Dialogs Document Go Project External Help
Quick bar Standard bar Fonts Tables Frames Forms List CSS
Custom menu C Apache DHTML DocBook HTML PHP+HTML PHP SQL Replace
/home/richard/Desktop/
doc
etc
home
richard
Desktop
google.html
ooo-calc.desktop
ooo-impress.desktop
Show full tree
Follow focus
google.html
Ln: 1, Col: 1
Joomla! 1.5 Beta 2 - Administration - Iceweasel
File Edit View History Bookmarks ScrapBook Tools Help
http://demo.joomla.org/demo15/administrator/index.php
joomla
Joomla! 1.5 Beta 2 -...
Joomla! Administration
Version1.5.0
Site Menus Content Components Extensions Tools Help
Preview 0 3 Logout
Add New Article
Article Manager
Frontpage Manager
Section Manager
Category Manager
Media Manager
Menu Manager
User Manager
Welcome to Joomla!
Done

```
<img alt="Google" height="110" src=
"/intl/en_ALL/images/logo.gif" width="276"><br>
<br>

<form action="/search"
  <table cellpadding="0"
    <tr valign="top">
      <td width="25%">&

      <td align="center" n
      value="en"><input t
      "ISO-8859-1"><inpu
      title="Google Search
      <input name="btnG"
      "Google Search"><ir
      "I'm Feeling Lucky">

      <td nowrap width="
      "-2">&nbsp;&nbsp;<
```

Congratulations on choosing Joomla! as your content management system. We hope you are able to create a successful website with our program and maybe, you are able to give something back to the community later.

To make your start with Joomla! as smooth as possible, we want to give you some pointers to documentation, common questions and help on securing your server.

You should start with the "Beginners guide to Joomla!" and for securing your server, you should read this security checklist.

Figure 2.5: Tools like Bluefish (left) or Joomla (right) can improve accessibility by preventing common mistakes, e.g. syntax errors, or by generating a consistent menu structure.

Ensuring Site-Wide Consistency

Content management systems (CMS) are used to create all pages for a website according to the developer's specifications. Typically, page content, page templates and other information is stored in a database and combined on demand when a page is requested. A CMS is an example for a system which is able to offer more advanced help for improving the usability of a site, by providing a separation of concerns – conceptually different types of data (page content vs. page template) are also stored separately. Furthermore, it allows generating parts of the HTML page content with a program rather than manually. This makes it easier to achieve site-wide consistency.

For instance, with a CMS like Joomla (`http://www.joomla.org`) it is usually easy to implement a consistent menu structure: If a new page is added, an appropriate menu entry is added to all "nearby" pages. In contrast to this, if the menu structure were updated manually on all pages whenever there are changes, this would be prone to errors which would negatively affect usability. A similar argument also holds for page templates: The same template is used as the basis for all pages, and updating the template is simple, whereas making the same change to all pages could easily lead to mistakes.

Automatic Generation of Pages From Models

Many web engineering solutions come with advanced tool support. (The most important approaches are examined in section 3 in greater depth.) A primary purpose of the tools is to automatically create HTML pages from the models that the developer has previously input. The generation process can produce output with good usability and accessibility. Compared to the simple text editor mentioned above, this can happen with a higher level of abstraction and with better quality, for the following reasons: First, the generation process has access to models which describe the complete website, so it need not work at the level of a single page at a time, but can also take inter-page relations into account. Second, models are more abstract than HTML code, so an advanced tool has more information available as a basis for its decisions.

For instance, web engineering models allow the specification of "web patterns" (see for example [Schwabe98OOHDM]) to describe the intended type of interaction with a page, such as the "guided tour", i.e. a sequential series of pages which provides documentation as well as prompting the user for input. The code generation process can use that information to create the right HTML controls which intuitively communicate this type of interaction – in the case of a guided tour, this will typically be a consistent page layout and "previous"/"next" buttons.

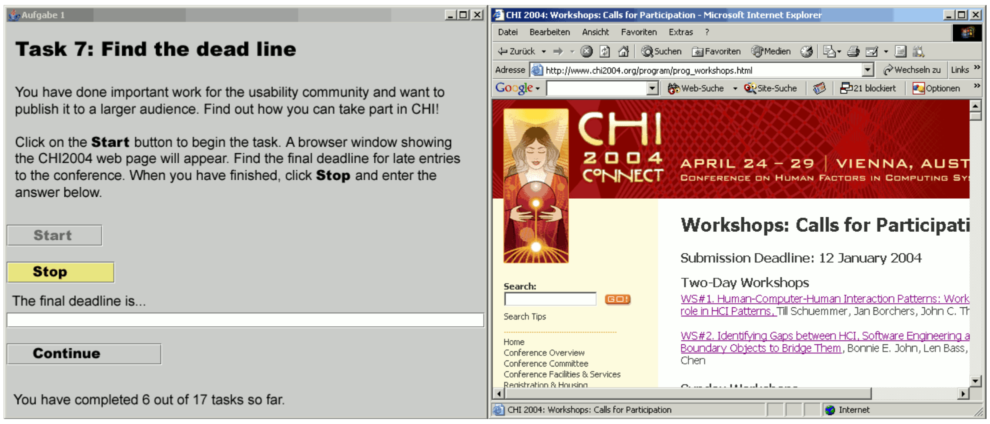
Outside the domain of web engineering, other types of "models" are also frequently used for generation of HTML content. Typically, this happens in a narrowly constrained domain of use: The model describes information which needs to be visualised, and a special-case tool transforms the model information and creates web pages from it. Only the model is ever edited, then the tool is re-run to update the web pages, or it creates the pages dynamically for each HTTP request. In a way, this description applies to all database-driven web applications which offer access to large amounts of information [Florescu98DB]. Automatic generation of the pages can lead to better usability – for example, if a special tool is written to present timetable information (e.g. for lectures) both in tabular form and as a weekly timetable, the information is guaranteed to be consistent for both views of the data. Without automatic generation, the versions would sooner or later deviate from one another because of human error, and the inconsistencies would make the information far less useful to visitors.

2.3.2 Automatic Capture of User Interaction With Web Pages

Once the pages of a site have been created, they can be analysed to determine the presence of any accessibility or usability problems. Following Ivory's classification, a number of approaches to this task can be distinguished, they are explained from this section up to section 2.3.4.

One of the standard procedures to evaluate a website with regard to its usability is to perform user testing [Constantine99Software, chapter 18]. Typically, users are invited to a usability lab, but they can also participate in the test remotely. They attempt to complete given tasks, and the degree of their success (e.g. how long it took to complete a task) and any problems, mistakes and other factors are measured. This helps the usability expert conducting the test to identify problems with the user interface. Performing all steps manually is possible, but opportunities exist in many places to automate tedious work, in particular with regard to the creation of a log of the test users' actions.

The employed techniques to record user actions vary, as does the level of detail of the recorded information. The different possibilities are listed below. In chapter 7, a specialized tool for user


Aufgabe 1
CHI 2004: Workshops: Calls for Participation - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten Medien
Adresse http://www.chi2004.org/program/prog_workshops.html
Wechseln zu Links
Google Web-Suche Site-Suche 21 blockiert Optionen
Start
Stop
Continue
CHI 2004 CONNECT
Search:
GO!
Search Tips
Home
Conference Overview
Conference Committee
Conference Facilities & Services
Registration & Housing
CHI 2004: Workshops: Calls for Participation
Internet

Task 7: Find the dead line

You have done important work for the usability community and want to publish it to a larger audience. Find out how you can take part in CHI!

Click on the **Start** button to begin the task. A browser window showing the CHI2004 web page will appear. Find the final deadline for late entries to the conference. When you have finished, click **Stop** and enter the answer below.

The final deadline is...

You have completed 6 out of 17 tasks so far.

APRIL 24 – 29 | VIENNA, AUST
CONFERENCE ON HUMAN FACTORS IN COMPUTING SY

Workshops: Calls for Participati

Submission Deadline: 12 January 2004

Two-Day Workshops

WS#1. Human-Computer-Human Interaction Patterns: Work role in HCI Patterns, Till Schuemmer, Jan Borchers, John C. Th

WS#2. Identifying Gaps between HCI, Software Engineering a Boundary Objects to Bridge Them, Bonnie E. John, Len Bass, Chen

Figure 2.6: SCONE/TEA allows many aspects of web-based user tests to be automated, such as giving instructions or noting when a task has been achieved. [Obendorf04Scone]

interaction logging on web pages is introduced. That chapter also includes a more detailed look at related work in the area of automatic capture of user actions.

- For web applications, a simple log file with access information of pages by users is usually created automatically by the web server, such as Apache (`http://httpd.apache.org`) or lighttpd (`http://www.lighttpd.net`). While the level of detail is not very high, this information is valuable because it documents *actual* usage by all users, rather than just usage under the artificial conditions of a user test. In the case that the log file is not available, for example because the website is not run by the usability expert, a proxy-based approach can be employed to collect comparable data.

  One possible way is to use a logging HTTP proxy which is transparent to the browser, for example Apache's mod_proxy module or SCONE [Obendorf04Scone]. SCONE/TEA is a system which allows almost full automation of user testing. In a preparation phase, the usability expert must first specify all aspects of a user test, such as instructions and questionnaires to be presented to the user (see figure 2.6), maximum time before a user is automatically taken to the next task, links which indicate completion of a task when clicked, and similar. Thus, the setup phase and manual work for each individual user test is reduced.

  Another option is URL rewriting: In the case of the WebQuilt system [Hong01WebQuilt], the tool runs on a website instead of a web server. Whenever a browser requests an URL from that website, that URL is re-requested from another, external site. Before returning the HTML data to the browser, any absolute URLs are modified to point back to the server WebQuilt is running on. This ensures that the user will continue to use the proxy rather than interacting with the external site directly.

- Keyboard input and mouse movements are recorded. This can either happen by installing special software on the computer which captures all mouse and keyboard input, by installing an instrumented browser which records actions on web pages [Reeder01WebEye], by adding an extension with this functionality to an existing browser (e.g. UserZoom, `http://www.userzoom.com`), or by using scripting on web pages [Mueller01Cheese] [Arroyo06MouseTrack].

- Eye tracking data is collected using special hardware, which continuously records the screen coordinates of the point the user is looking at. The gaze data augments mouse tracking data in the sense that it makes it clearer where the user is looking for information before making the decision about their next action. Each hardware vendor's system comes with its own recording software.

- The entire screen content of the test user's machine can be recorded and saved in a movie file. This can be useful to see the exact context of a user action during later analysis of log data, but has considerable CPU and storage requirements. An example of a program for this purpose is recordMyDesktop (`http://recordmydesktop.sourceforge.net`).

- A video and/or audio recording can be produced of the users' facial expression and any comments they make about their progress. Sometimes, this is used with a "think aloud" protocol [Lewis93TaskCentered, chapter 5], i.e. the test users are instructed to say what they are thinking while using the web application. Again, during later analysis of interaction log data, this can provide useful insights, e.g. with regard to the cause for a failure to complete the task. An example for a commercial application which offers this feature is Morae Recorder (`http://www.techsmith.com/morae/record.asp`). Furthermore, many general-purpose tools allow recording of video, e.g. from a webcam, and of an associated audio signal using a microphone.

2.3.3 Automatic Evaluation of User Interaction Logs

Recording the user interaction log data is usually not the most work-intensive step of a user test – it is comparatively easy to create a large amount of very detailed log data. However, the step following it can be more time-consuming: The data must be analysed and interpreted to identify the different usability problems of the application. Especially if audio and video material has been collected, this analysis can be work-intensive. Again, tool concepts have been developed to aid in the evaluation of this data. As the results of the test must ultimately be judged by a human, completely automatic evaluation of results is usually not possible.

Analysis of HTTP Server Logs

In the case of standard HTTP server or proxy logs, a variety of available tools can create statistics from the raw log data, for example AWStats (`http://awstats.sourceforge.net`), analog (`http://www.analog.cx`) or Visitors (`http://www.hping.org/visitors/`). They highlight information which might affect the site's usability in different ways. For example, through the analysis of "404 not found" errors, outdated links (possibly also from external sites outside one's control) can be identified. The statistics can also help to identify different user groups via the pages they visit, and reveal the primary interests of each group, e.g. with a list of most frequently

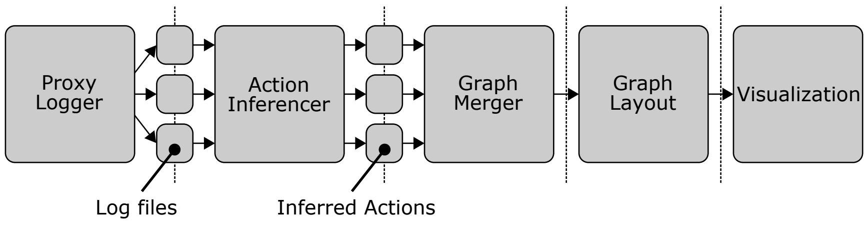


Figure 2.7: WebQuilt processes log data in several steps before visualising it. It is combined into a graph of web pages and the movements of users between them. [Hong01WebQuilt]

visited pages. Furthermore, the “Referer” information embedded in browsers’ HTTP requests is a valuable source of information: In the case that a visitor found the website via a search engine, it contains the words of their search engine query. Thus, the user’s intention can be compared to the information they found on the website.

The WebQuilt system mentioned above is not only suitable for recording user clickstreams, but also contains visualisation features for the recorded data [Hong01WebQuilt, section 2.2 to 2.5]. This is done in several steps (see figure 2.7): First, an action inferencer converts log entries into “actions”, where an action is defined as going to a new page in the browser by clicking on a link or using the back/forward button. Next, a graph merger processes the log files created by multiple users and aggregates them. It outputs a single graph of web pages and the actions which lead from one page to another. The graph is subsequently visualised (figure 2.8): The higher the number of users who went from one web page to another, the thicker the arrow between these pages gets. The usability expert can use this to determine whether test users follow the expected navigation paths when they attempt to solve a task. Furthermore, red arrows indicate users who

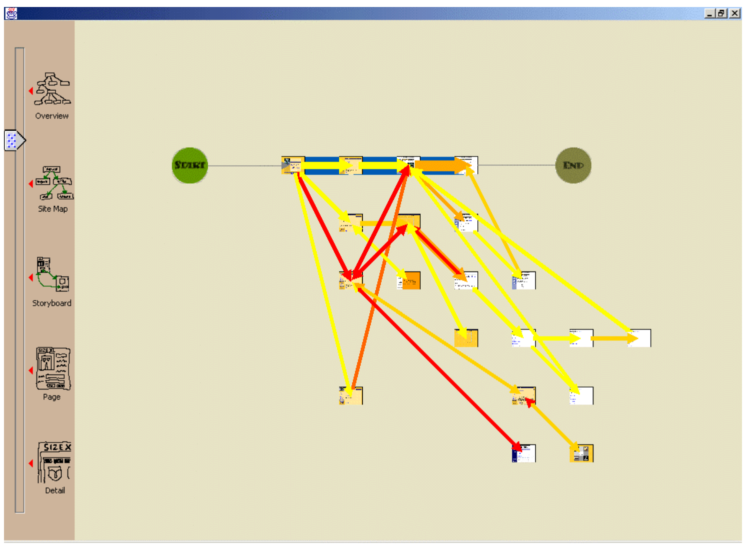


Figure 2.8: WebQuilt visualisation of user actions during a user test. Thicker arrows indicate that a higher number of users went from one page to another. [Hong01WebQuilt]

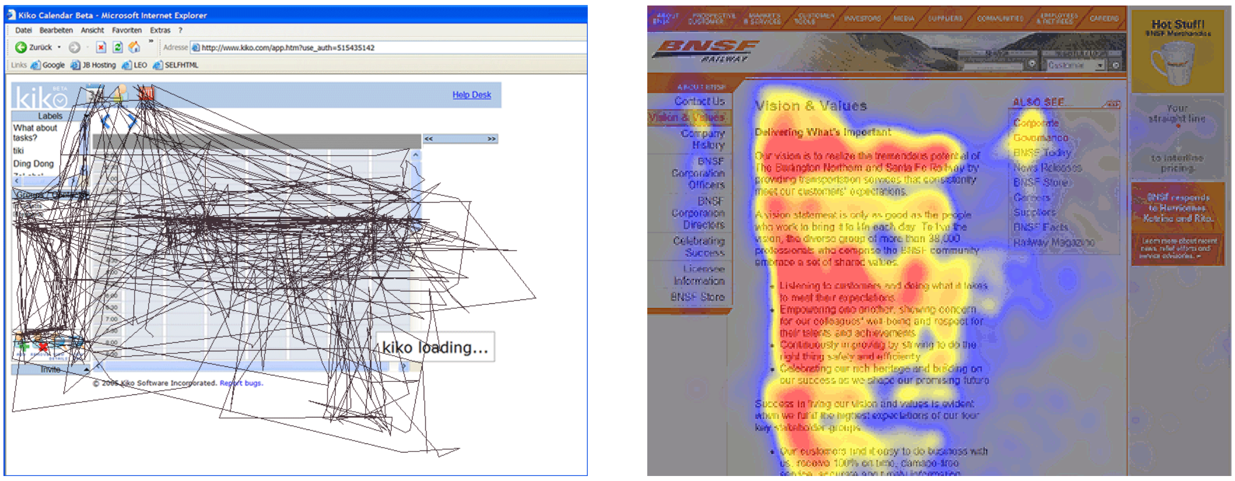


Figure 2.9: Examples for usage data visualisation: A single user's mouse trail (left), heat-map of eyetracking data from many users (right). [Atterer06UsaProxy] [NielsenAlertbox]

take a long time to go to the next page. The expert can quickly identify each page via a screenshot that is automatically created, and by using a built-in zoom feature.

Analysis of Mouse Tracking and Eye Gaze Data

When performing a usability test with automatic capture of mouse and eye tracking data, there is no manual alternative to automated analysis of this data, as it is too fine-grained and there are too many data points, whose coordinates have no immediate meaning to a person who views them.

One way of extracting concrete information from the data is to write small scripts to post-process it. Many questions can be answered this way. For example, the developer's goal may be to find out whether the size of buttons is sufficient for the fine motor skills of the audience. If this question is important enough to justify the effort, he can develop a script with heuristics to recognize and count cases where users click on a neighbour button instead of the one they mean to click. An advanced example of this type of analysis is presented in [Card01InfoScent], where data from several sources (mouse, gaze, keyboard, video) is combined into "web behavior graphs" and used to distinguish various types of problems encountered by test users.

Another way to provide an informative summary of the data is to generate data structures from it, e.g. graphs of pages, of screen areas within a page or of similarities between different users' navigation habits. This intermediate data can subsequently be displayed (e.g. as TreeMaps, hyperbolic trees, fisheye view) and explored interactively (using concepts like semantic zooming, focus & context etc.) using available visualisation toolkits [Heer05Prefuse].

Finally, a screenshot (or a recorded video of the screen) can be used as the background for a visual representation of the data:

- To visualise a short sample of a single user's mouse or gaze data, such as their actions on an HTML page, the individual coordinate samples can be connected with lines (figure 2.9, left [Atterer06UsaProxy]). The visualisation can be extended to include information such as timestamps [Card01InfoScent], clicks, the speed of movement and the dwell time when the mouse is not moving. It can be used by the usability expert to find out how exactly a goal was achieved (e.g. which link was clicked) and to identify parts of the user interface the mouse only hovered over without clicking.
- If the amount of data is too large, detailed features like lines would become indistinguishable. For this reason, a heatmap (figure 2.9, right [NielsenAlertbox, 17 April 2006]) is more appropriate in this case. For gaze data, it uses a colour encoding to show the total amount of time a particular part of the screen has been looked at by different users. The resulting cloud-like shapes reveal the way the "average user" scans and reads the page. With the example heatmap shown in figure 2.9, a web designer could infer that it would be a mistake to put important navigation items in the top row (above the logo), because almost none of the test users looked at that region of the screen.
- When mouse and/or eye gaze data has been recorded during a user test, replaying the data at a later time is another option. Typically, this will only be done for a few interesting datasets. For example, if a user fails to find the right user interface control on a web page, eye gaze data can help to distinguish the case that the control was not even looked at, and the case that it was looked at, but not recognized as the right way to proceed towards the user's goal. In the first case, the web developer should consider placing the control elsewhere on the screen, in the latter case he can try to improve usability by changing the control's on-screen representation.

Whenever the developer wants to analyse user actions at the level of individual controls on a page (links, buttons etc.), it is problematic that the recorded gaze or mouse data usually only consists of timestamps with screen coordinates. Additional work is required to map those coordinates to parts of the user interface, which can quickly become complicated if the user scrolls the viewed web pages. In the available literature, that step has often been performed manually (see e.g. [Chen01Correlation]), whereas the tool described in chapter 7 automates it.

One challenge when analysing the data from usability tests is the requirement to work with different types of data (URLs, mouse coordinates, keyboard) at the same time, annotating them using timestamps, extracting only interesting parts and visualising different extracted subsets of data side by side. Thus, simply integrating into one tool some of the concepts which were outlined above can further reduce the amount of work for the usability expert. Furthermore, easy access to the different data sources can help him to discover usability issues which are not obvious from a single data source.

2.3.4 Automatic Accessibility and Usability Validation

To assess the accessibility and usability of existing web pages, automated validators can be utilised. These programs are given the URL of a web page. They download the HTML code

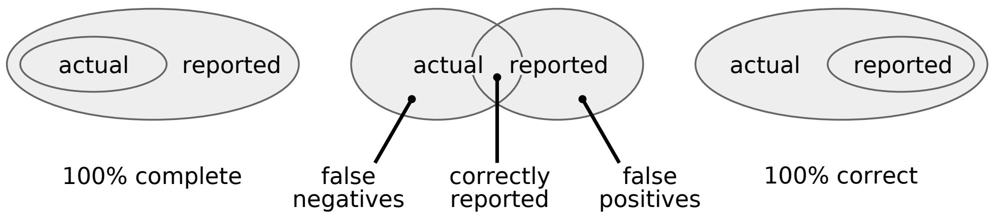


Figure 2.10: Sets of actual usability problems which are present in a website vs. usability problems reported by a validator. When comparing validators, higher completeness of a tool means it has fewer false negatives, higher correctness means fewer false positives.

of the page, or in some cases all pages of a given website. Subsequently, the HTML is analysed and possible problems with the page are listed in a report. In [Brajnik04Comparing], the quality of a good automated usability validator's report has been summarized using the three aspects below. Figure 2.10 helps to make them clear by comparing the set of actual usability problems that are present in a website with the set of problems that is reported by a validator.

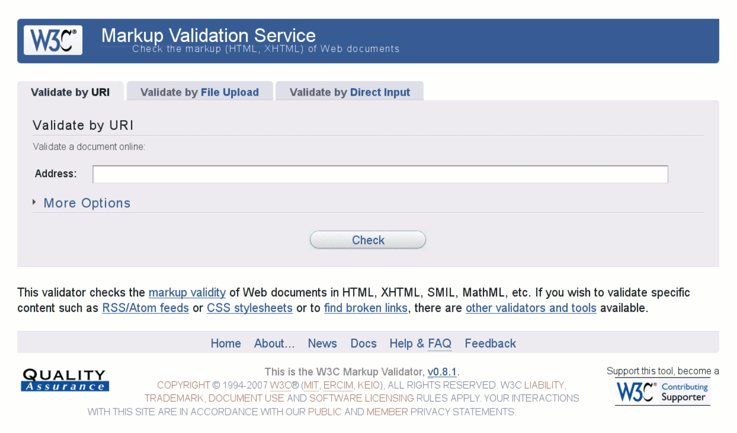
- Completeness: When restricting oneself to the usability problems which are actually present in the website, the completeness of a tool corresponds to the fraction of these usability problems that are correctly identified and reported. In other words, the completeness value specifies how well the tool reduces the amount of *false negatives*, i.e. actual usability problems which are not detected. It does not make any statement regarding issues which are reported even though they are not present in the site. The left part of the figure shows an extreme example: If a tool outputs an excessive amount of problems, it will reach a high completeness, even if many problems are not really present.

  Tools with high completeness can identify many issues – this way, their built-in expert knowledge can supplement the web developer's own knowledge about usability guidelines.

- Correctness: The more correct a tool, the higher is the percentage of usability issues in its reports which really are problems with the actual website. This means that higher correctness implies a reduction of *false positives*, i.e. properties which are incorrectly reported as problems even though they are not. Correctness is thus a counterpart to completeness. This symmetry is also visible in the figure. On the right, the actual/reported problem sets for a fully correct tool are shown. A tool can achieve high correctness by only outputting problems about whose presence it is certain.

  The important aspect of correctness is that tools must be able to suppress *incorrect* occurrances of problems in their reports to an acceptable level. A large number of false positives in the report can make it too long to read and can prevent actual problems from being spotted.

  Because false positives are less problematic (they only cause additional work for the developer), current tools tend to err on this side. However, this is far from ideal – too many false positives will cause the developer to stop using the tool because there is no perceived benefit compared to just checking the website manually.



This validator checks the markup validity of Web documents in HTML, XHTML, SMIL, MathML, etc. If you wish to validate specific content such as RSS/Atom feeds or CSS stylesheets or to find broken links, there are other validators and tools available.

This is the W3C Markup Validator, v0.8.1.
COPYRIGHT © 1994-2007 W3C® (MIT, ERCIM, KEIO), ALL RIGHTS RESERVED. W3C LIABILITY, TRADEMARK, DOCUMENT USE AND SOFTWARE LICENSING RULES APPLY. YOUR INTERACTIONS WITH THIS SITE ARE IN ACCORDANCE WITH OUR PUBLIC AND MEMBER PRIVACY STATEMENTS.

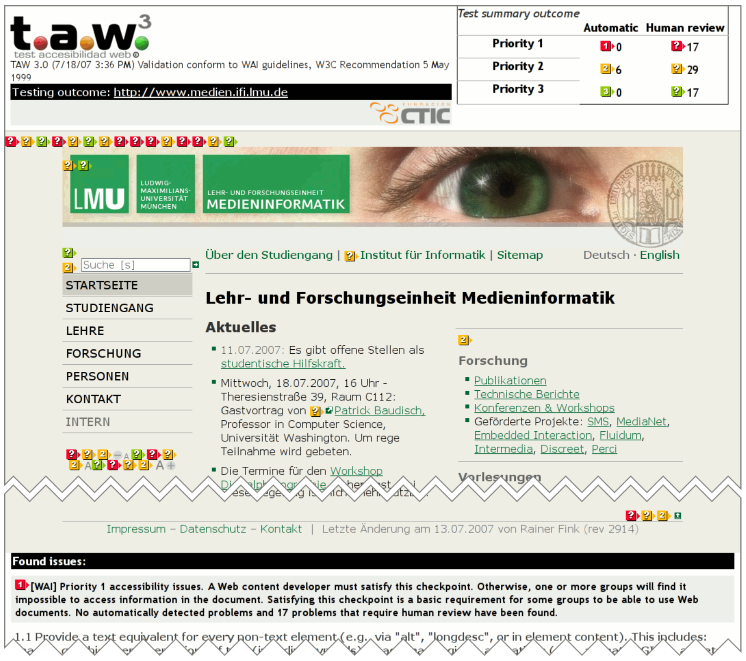
Figure 2.11: Ensuring syntactical correctness of web pages (e.g. using the W3C validator, `http://validator.w3.org`) is a first step towards good accessibility of a website.

- Specificity: The tool's report describes the problem in a way that is as detailed as possible, or, from another point of view, it distinguishes between different possible errors for each property that is checked. This requirement follows from the assumption that the web developer may not be a usability expert, so he must be given enough information to understand the reported problems.

Due to the fact that section 6.1 contains a detailed description of previous work in this area, only an overview of important tool properties is presented in the following.

Evaluation of single page or entire website: Many validators only download and analyse a single HTML page, for example the WAVE [Kasday00WAVE]. In contrast to this, others such as Bobby [WatchfireBobby] download all pages which are up to a specified number of clicks away from the website's homepage. In the latter case, the tool's validation heuristics can also include the website's link structure in its analysis, which can make the tool more powerful.

Simple syntax checks or complicated usability heuristics: Different tools validate the page using different approaches. At the conceptually simplest level (e.g. the W3C or Schneegans validators, `http://validator.w3.org`, figure 2.11, and `http://schneegans.de/sv/`), they only verify that the syntax of the page's (X)HTML and CSS is correct. More advanced validators instead aim at detecting a set of more abstract accessibility problems with the page – for example, the ATRC web accessibility checker (formerly A-Prompt, `http://checker.atrc.utoronto.ca`) contains code to detect problems with WCAG conformance of pages. Even more abstract tests are imaginable – chapter 6 will present a few additional ideas and approaches of implementing them.
WebTango [Ivory01Metrics] is special in that it does not directly check guidelines using an algorithmic implementation of usability guidelines. Instead, it measures "web quality"


t.a.w.3
test accesibilidad web ©
TAW 3.0 (7/18/07 3:36 PM) Validation conform to WAI guidelines, W3C Recommendation 5 May 1999
Testing outcome: http://www.medien.ifi.lmu.de

Test summary outcome

	Automatic	Human review
Priority 1	0	17
Priority 2	6	29
Priority 3	0	17

Über den Studiengang | Institut für Informatik | Sitemap   Deutsch · English

STARTSEITE
STUDIENGANG
LEHRE
FORSCHUNG
PERSONEN
KONTAKT
INTERN

Lehr- und Forschungseinheit Medieninformatik

Aktuelles

- 11.07.2007: Es gibt offene Stellen als studentische Hilfskraft.
- Mittwoch, 18.07.2007, 16 Uhr - Theresienstraße 39, Raum C112: Gastvortrag von Patrick Baudisch, Professor in Computer Science, Universität Washington. Um rege Teilnahme wird gebeten.
- Die Termine für den Workshop

Forschung

- Publikationen
- Technische Berichte
- Konferenzen & Workshops
- Geförderte Projekte: SMS, MediaNet, Embedded Interaction, Fluidum, Intermedia, Discreet, Perci

Vorlesungen

Impressum – Datenschutz – Kontakt | Letzte Änderung am 13.07.2007 von Rainer Fink (rev 2914)

Found issues:

[WAI] Priority 1 accessibility issues. A Web content developer must satisfy this checkpoint. Otherwise, one or more groups will find it impossible to access information in the document. Satisfying this checkpoint is a basic requirement for some groups to be able to use Web documents. No automatically detected problems and 17 problems that require human review have been found.

1.1 Provide a text equivalent for every non-text element (e.g., via "alt", "longdesc", or in element content). This includes:

Suche [s]
LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
LMU
LEHR- UND FORSCHUNGSEINHEIT MEDIENINFORMATIK
CTIC

Figure 2.12: The report of the TAW accessibility checker (`http://www.tawdis.net`) highlights problems using an annotated version of the analysed page.

of a site using a statistical method: First, a large number of sites with a given usability rating (determined by human experts) are inspected by the tool, and about 150 different measures are calculated, such as the amount of text on a page or its loading speed. Later, the measures are compared to those for a new site which is to be evaluated. Using e.g. linear regression, the new site's probable human expert rating is calculated.

Fully automatic and interactive operation: Many of the available validators operate fully automatically. This is advantageous because it facilitates integrating automatically scheduled checks into the workflow of web development, and for large-scale accessibility studies of many websites, e.g. the study carried out in [Marincu04Comparative].
However, there is an advantage to asking the user for additional information. For example, if the heuristic algorithm for a certain usability guideline encounters a case where it is unsure, it can ask the developer for details and take the answer into account when it encounters the same situation again in the future. This way, only a little additional information can improve validation quality by eliminating numerous identical false positives in the usability report. NAUTICUS [Correani04Nauticus] is an example for a tool which supports interactive operation.

Presentation of accessibility/usability report: When it comes to presenting the results of the validation, the available validators can be divided into several groups: The "classical" type of output is a web page (or a dialogue by a desktop GUI program) which lists the failed tests for inspection by the developer and optionally provides information on how to repair the problems. One of the first accessibility checkers, the original Bobby service (no longer available as an online service, now replaced by WebXACT), fell into this category. An alternative way of highlighting problems is used by TAW (Test de Accesibilidad Web, `http://www.tawdis.net`) and other tools: As the test result, the tool returns the web page it has just analysed, annotated with additional icons for each problematic feature (see figure 2.12). Finally, some validators such as ATRC can output their report in a machine-readable format – [W3C-EARL] is the predominant standard.

Web-based service or desktop GUI application: With regard to the way they are implemented, automated usability validators can be divided into two types of tools: Many programs are provided via a website on a server and download web pages that are specified using an HTML form (e.g. ATRC). Other tools are GUI applications that are installed on a desktop computer and used by a single user at a time (e.g. MAGENTA, `http://giove.isti.cnr.it/accessibility/magenta/`). In some cases, both is possible, such as with WebXACT, which offers a web-based service of the same name, and a desktop application under the name Bobby. (The original Bobby was a web-based service.)

Hard-coded heuristics and extensible rulesets: The majority of available tools implements accessibility and usability checks with specially-written code, using e.g. Java or C. Adding more rules usually requires good knowledge of the programming language that was used, and a recompilation of the validation program. Thus, adding rules can become quite complicated. Additionally, it is advantageous if the same person is both a programmer and a usability expert, the two cannot be separated easily.
For these reasons, some approaches concentrate on developing special-domain languages for the description of usability tests, and provide mechanisms for extending the validator's set of guidelines at runtime. This includes the Kwaresmi tool [Beirekdar02Kwaresmi] with its Guideline Definition Language (GDL) and EvalIris [Abascal03EvalIris] which uses an XML-based guideline language.
Statistics-based tools like WebTango do not fit in either of the two categories of hard-coded tests or extensible rulesets.

Automatic critique: In addition to identifying problems, a tool can provide suggestions of how to repair them. In many cases, it can achieve this simply by adding to the usability report some fixed text describing possible solutions. This is done by many validators, e.g. ATRC. A more advanced form of automated critique is to provide suggestions which are tailored to the exact instance of the problem. For example, rather than telling the user to add an `alt` attribute to his `<img>` tag, the tool will cite the relevant portion of the HTML document, with a dummy `alt` attribute already inserted.

Automatic repair: As an extension of automatic critique, a tool can make it easy for the developer to apply the suggested solution to the website, by integrating analysis and editing support into one application – this results in an automated interactive repair system, an example is NAUTICUS. Non-interactive operation is also imaginable; at the level of syntactical correctness of the HTML code, HTML tidy (`http://tidy.sourceforge.net`) can be used to turn documents into valid HTML.

Conclusion

It is noticeable that each of the tools and services listed above concentrates on a certain area of automatic validation. Taken together, the features of the tools are impressive, but each individual tool has its shortcomings, usually either in terms of completeness, i.e. the number of different usability aspects that are tested, or with regard to correctness, e.g. the number of incorrectly reported problems. Some academic prototypes are only meant to demonstrate a concept and are not suitable for productive use. Today, no single tool performs very well in all three aspects cited by [Brajnik04Comparing] – while the foundations of the different tool concepts have been laid, there is room for improvement in many areas.

2.3.5 Simulation of User Perception or Behaviour

The term *simulation* is used in different ways: On one hand, it means that the web page is displayed using a special program which imitates the effects of a personal disability, of a particular rendering technology, or similar. On the other hand, the term can also be employed to refer to the simulation of a user test, e.g. for determining whether the performance of a site will still be satisfactory when it is used by a large number of users.

Simulating Perception Differences of Web Pages

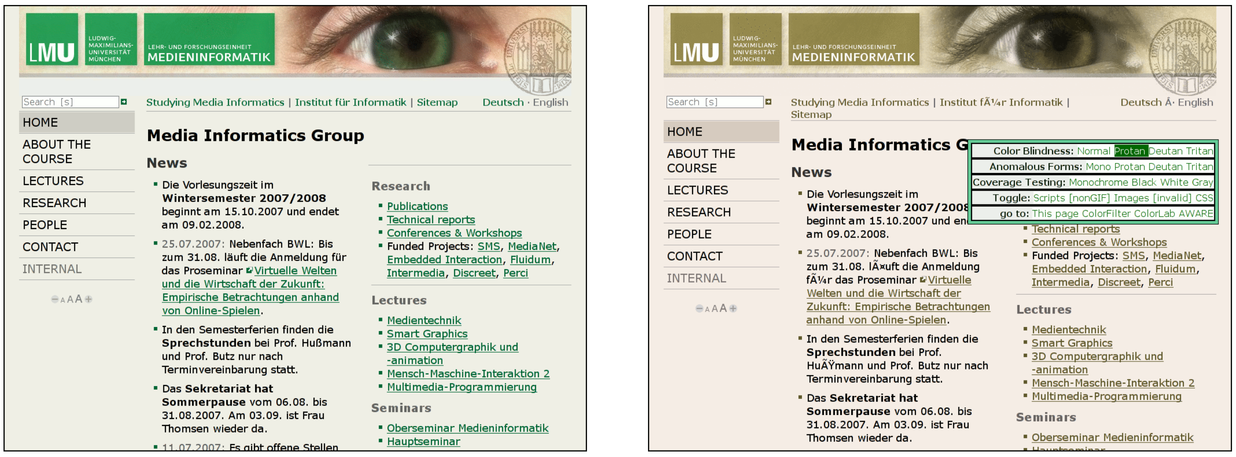
A variety of tools and techniques allows the web developer to "see websites through the eyes of his users" rather than with the browser and in the environment used for development and testing.

At the simplest level, this applies to the browser rendering engine that is used. Looking at the site with different browsers may reveal differences in the browser implementation which cause problems. For instance, JavaScript errors may prevent some of the GUI elements of an AJAX application from appearing and thus render the application unusable. Services such as BrowserCam (`http://www.browsercam.com`) allow tests of the page on rendering engines which are not readily available to the developer.

To simulate special browsers, such as the ones used on mobile devices, demo versions are often available. For example, Opera Software offers the "small screen rendering" technology which is used in Opera Mobile as a special mode of the desktop Opera browser. Furthermore, the Opera Mini browser (used for lower-performance devices than Opera Mobile) can be used on desktop computers via a web-based service (`http://www.operamini.com/demo/`).

A normal browser can also be used to simulate certain kinds of disabilities: Some users with bad vision configure their system to use a very large font size, a step that can easily be performed by the web developer as well. Alternatively, they magnify all aspects of the page (fonts, graphics, pixel-based CSS measurements) by a certain factor. If the developer's browser does not allow this, it can be simulated simply by using normal font settings, but then resizing the browser window to a much smaller than average size, e.g. 512×384 pixels. The web page should still be usable with both kinds of changes.

Some blind users use braille-based terminals running text browsers. By looking at the page with a text-based browser such as lynx (`http://lynx.isc.org`), a normally sighted person can


LMU
LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
LEHR- UND FORSCHUNGSEINHEIT MEDIENINFORMATIK

Search [s]

Studying Media Informatics | Institut für Informatik | Sitemap Deutsch · English

HOME
ABOUT THE COURSE
LECTURES
RESEARCH
PEOPLE
CONTACT
INTERNAL

A A A

Media Informatics Group

News

- Die Vorlesungszeit im **Wintersemester 2007/2008** beginnt am 15.10.2007 und endet am 09.02.2008.
- 25.07.2007: Nebenfach BWL: Bis zum 31.08. läuft die Anmeldung für das Proseminar Virtuelle Welten und die Wirtschaft der Zukunft: Empirische Betrachtungen anhand von Online-Spielen.
- In den Semesterferien finden die **Sprechstunden** bei Prof. Hußmann und Prof. Butz nur nach Terminvereinbarung statt.
- Das **Sekretariat hat Sommerpause** vom 06.08. bis 31.08.2007. Am 03.09. ist Frau Thomsen wieder da.
- 11.07.2007: Es gibt offene Stellen

Research

- Publications
- Technical reports
- Conferences & Workshops
- Funded Projects: SMS, MediaNet, Embedded Interaction, Fluidum, Intermedia, Discreet, Perci

Lectures

- Medientechnik
- Smart Graphics
- 3D Computergraphik und -animation
- Mensch-Maschine-Interaktion 2
- Multimedia-Programmierung

Seminars

- Oberseminar Medieninformatik
- Hauptseminar

Search [s]

Studying Media Informatics | Institut fÃ¼r Informatik | Sitemap Deutsch Â· English

HOME
ABOUT THE COURSE
LECTURES
RESEARCH
PEOPLE
CONTACT
INTERNAL

A A A

Media Informatics G

Color Blindness: Normal Protan Deutan Tritan
Anomalous Forms: Mono Protan Deutan Tritan
Coverage Testing: Monochrome Black White Gray
Toggle: Scripts [nonGIF] Images [invalid] CSS
go to: This page ColorFilter ColorLab AWARE

News

- Die Vorlesungszeit im **Wintersemester 2007/2008** beginnt am 15.10.2007 und en am 09.02.2008.
- 25.07.2007: Nebenfach BWL: Bis zum 31.08. lÃ¤uft die Anmeldung fÃ¼r das Proseminar Virtuelle Welten und die Wirtschaft der Zukunft: Empirische Betrachtungen anhand von Online-Spielen.
- In den Semesterferien finden die **Sprechstunden** bei Prof. HuÃŸmann und Prof. Butz nur nach Terminvereinbarung statt.
- Das **Sekretariat hat Sommerpause** vom 06.08. bis 31.08.2007. Am 03.09. ist Frau Thomsen wieder da.

- Technical reports
- Conferences & Workshops
- Funded Projects: SMS, MediaNet, Embedded Interaction, Fluidum, Intermedia, Discreet, Perci

Lectures

- Medientechnik
- Smart Graphics
- 3D Computergraphik und -animation
- Mensch-Maschine-Interaktion 2
- Multimedia-Programmierung

Seminars

- Oberseminar Medieninformatik

Figure 2.13: The Colorblind Web Page Filter (`http://colorfilter.wickline.org`) simulates colour blindness. Left: Normal version, right: reduced-colour version of page).

have a similar user experience. Alternatively, aural browsers are available for free, e.g. the "Fire Vox" Firefox screen reader extension (`http://firevox.clcworld.net`) or the Fangs screen reader (`http://sourceforge.net/projects/fangs/`).

Finally, it is possible to calculate the effect that colour blindness has [Brettel97Color], enabling a person with full colour vision to have a look at a web page with the eyes of a colourblind user. The "Colorblind Web Page Filter" (`http://colorfilter.wickline.org` and figure 2.13) operates on entire web pages, including graphics and stylesheets. Vischeck (`http://www.vischeck.com/vischeck/`) can work with arbitrary uploaded pictures and also many web pages. Different types of colour blindness can be selected with both tools, including Deuteranope, Protanope and Tritanope.

Simulation of Hypothetical Users

Simulating the actions of users is not easy – since it is hard to create rules which reliably predict human behaviour, simulation tools will often only roughly mirror actual behaviour. Nevertheless, a number of approaches promise to improve usability:

A website becomes unusable when it gets too slow under a high load of users. For this reason, performance testing can avoid later problems with a site. [Ivory03WebEval, chapter 10, 2.2.1] contains an extensive list of performance measurement approaches. The basic idea is to simulate many users which access the site concurrently and perform actions on it.

It is also possible to simulate the navigation steps that a user takes through the website: By analysing the so-called information scent of web pages based on existing usage data, a program can build up knowledge about visitors' goals and associated navigation paths. In the next step, this information can be used to predict the probable path of a simulated user through the site given his needs, or to predict an actual user's need based on his recorded path through the site [Chi00SiteScent]. The user's "needs" are expressed in the form of a list of natural-language words (similar to a search engine query), each of which is accompanied by a probability value. CoLiDeS [Kitajima00CoLiDeS] uses a similar idea. It is a model of the way users search for their goal on web pages, and uses a decomposition of web pages into areas like navigation links, headings, text paragraphs etc. to simulate the way the page is scanned. The probable path of the user can be predicted. The simulation of impasses and the resulting backtracking can also be represented.

Chapter 3

Usability Aspects in Existing Web Engineering Development Processes

This work bases its advances on two related fields of study: Apart from the area of web usability, which was discussed in the previous chapter, the area of web engineering with its methods, models and tools provides the basis for the development of new concepts. Later chapters will elaborate on the central idea of this thesis: Using models to improve usability tool support.

In this chapter, a closer look is taken at existing web engineering solutions: If the developer follows the recommended procedures by creating models, using the supplied tools, performing testing etc., then what usability-related artefacts (e.g. models, results of automated model or HTML analysis) will be produced, how much tool support exists for enhancing usability, and what other aspects of the development process (e.g. user tests) will ensure good usability of the result?

3.1 Web Engineering – Overview

Summary: Web engineering applies established software engineering practices to the domain of WWW applications. Due to the special properties of this medium, the development process must be adapted in many places in order for it to be cost-effective and to produce high-quality results.

Several definitions of web engineering have been proposed in the available literature to characterize the domain (see also [Dumke03WE, p. 47], [Kappel04WE, p. 4]):

> Web Engineering – The application of systematic, disciplined and quantifiable approaches to the cost-effective development and evolution of high-quality applications in the World Wide Web. [Gaedke00WE]

> Web Engineering is the application of systematic, disciplined and quantifiable approaches to development, operation, and maintenance of Web-based applications. It is both a pro-active approach and a growing collection of theoretical and empirical research in Web application development. [Deshpande02WE]

Web engineering can be regarded as a variant of software engineering which is specialized for the creation of applications for the World Wide Web, including special development, testing and maintenance methods. The changes are necessary because development of web applications and of conventional applications (such as GUI-based desktop applications) differs in a number of aspects. Due to the role of the WWW as a communication medium, many web applications have properties which would be unusual for desktop applications. Several of the differences which were identified in [Deshpande02WE] are related to the usability of websites:

- Web applications are information-centric ("content is king") – they deal with text-based content as well as multimedia content. This content is often only present in a way that is hard to structure.

- The users of web applications are different from the user groups of conventional software. As the web application is usually available globally, the users have a larger variety of backgrounds, differences in language, culture etc., which adds complexity to the task of making the application work for as many users as possible.

- On the web, there is a strong competition between sites which offer similar services, so users are more impatient than if they had installed software on their own computer. If a web application does not provide the right user experience, dissatisfied users will simply move on to the next search result [Card01InfoScent].

- Performance is more critical with web applications – a single server installation must provide the service to all users, and the exact number of users is hard to predict. This way, bottlenecks are more likely to occur. Furthermore, the network bandwidth available to some users may be very low.

- With web applications, the developer must be acquainted with a larger number of different standards, languages and tools, and with additional disciplines like graphic design or GUI design. Insufficient knowledge in only one of these fields may result in problems for a portion of the application's users.

Different sub-disciplines have formed in the research field in the last years to address particular aspects of the problem space. For example, the proceedings of a recent web engineering conference [ICWE07Proc] include work on topics such as the following:

- Modelling of the web application, e.g. of the website structure and the user interface
- The Semantic Web, e.g. annotating web content with machine-readable metadata
- Automatic creation of a web application from models
- Device independence, i.e. adapting the presentation of content to the screen resolution etc.
- Tool support for other development phases, e.g. testing
- Measuring and ensuring web quality, i.e. usability and accessibility

3.2 Web Engineering Approaches

Summary: In an overview of existing web engineering solutions, features which may influence the usability of the web application are highlighted. While some methods, models and tools include aspects which aim at improving usability, it cannot be said that all aspects of usability are fully covered by any solution.

The concepts of models, methods and tools are central to web engineering [Kappel04WE, section 3.3]: With models, the properties of the application are described. Most of the time, they are diagrams containing graph-like structures. A variety of different graphical notations has been developed, and different types of models exist for different aspects of development – for example, models can describe the navigational structure of the site, the layout of the web pages, or (similar to "classical" software engineering) the database structure or application logic.

Methods are comprised of procedures which in their entirety enable the developer to create the application in a systematic way, and to avoid mistakes which would result in lower quality. They specify the order and frequency of each of the development steps, such as starting with an initial feasibility and requirements analysis, continuing via design, modelling and implementation, and concluding with testing and maintenance.

Finally, the tool support is one of the reasons for detailed modelling of applications: While the models are also useful to the developer (or a team of developers) for understanding the problem to be solved, and the structure of the solution, the task of creating detailed models is only worth the effort because programs can use the models to automatically or semi-automatically create an implementation, or assist with other steps of the development process.

In the following, an overview is given of a number of web engineering solutions. In the context of this work, usability is a primary interest, so while a general description of the major components of all solutions is given, the focus is on aspects of the methods, models and tools which are relevant for improvements to the usability of the web application.

As shown later, it is advantageous to include additional information in the models, for example information about the target audience of a site. For this reason, it is also of interest whether there are provisions for extensions to the different web engineering solutions' models, and what these extensions would look like. In some cases, extensions may be necessary whereas another solution already includes models for the relevant aspects.

In classical software engineering, e.g. for the development of desktop applications, UML has been accepted as the successor of the three major solutions which were previously used for modelling [Booch99UMLUserGuide]. In comparison to this, no such convergence has so far been observed for web engineering – graphical notation, scope of models and development steps differ between web engineering solutions, although UML is increasingly used for modelling [Kappel04WE, p. 74].

In [Kappel04WE, section 3.9], existing web engineering approaches are classified according to their origin: Entity-relationship (ER) modelling, the Object Modelling Technique (OMT) or the Unified Modelling Language (UML). Furthermore, influences between methods are documented, resulting in a categorization of four method generations. Sections 3.2.1 to 3.2.3 describe the only 3rd- and 4th-generation methods, UWE, WebML and OO-H, and section 3.2.4 gives short descriptions of other methods.

3.2.1 UML-Based Web Engineering (UWE)

UML-Based web engineering [Hennicker01Modeling] is a web engineering solution which uses the Unified Modeling Language (UML) as the graphical notation for its models. It is also influenced by other concepts developed by the consortium which is responsible for the UML standard, the Object Management Group (OMG). They include: [Koch06Transformations]

- Model-Driven Architecture approach, MDA [OMG03MDA]: A system is first specified using models which are independent of the platform, then the platform (e.g. “web application”) is specified, and finally a transformation is performed to convert the platform-independent model (PIM) into the platform-specific model (PSM).
- XMI (XML Metadata Interchange): An XML dialect for the representation of models
- OCL (Object Constraint Language, [OMG06OCL]): A character-based language for the specification of conditions that apply to a model
- QVT (Queries/Views/Transformations): A standardized procedure for the transformation of one type of model into another

UWE Development Method

The UWE development process is based on the Unified Process [Jacobson99UnifiedProcess]. In addition to its four phases inception, elaboration, construction and transition, UWE has a fifth phase, the maintenance phase [Koch00Adaptive, chapter 7]. During each phase, attention is paid to the peculiarities of web applications. For example, in the construction phase, the developer needs to concern himself with issues such as dynamic page generation possibilities, user behaviour observation and usability. The additional maintenance phase is necessary due to the fast technological development of the web and the role of websites as sources of up-to-date information – the application needs to be adapted to changes in the environment, and the content of the web pages needs to be updated.

The use of models plays a central role in the development of applications with UWE. Figure 3.1 depicts the different models and the transformations between them. This stereotyped UML activity diagram shows models as object flow states and transformations as circle symbols. The more specific models are later generated from the more abstract ones. However, some of the more concrete models are also created directly by the developer, or are generated semi-automatically, i.e. they are based on an abstract model and then refined manually.

Development begins with the creation of requirements models [Koch02Expressive]. Use cases and task models are added to describe interactive aspects of the application. Later, a conceptual design as well as navigation and presentation models follow.

The UWE process does not prevent steps like rapid user interface prototyping or user tests from taking place. However, in the available literature on UWE, discussion of usability issues

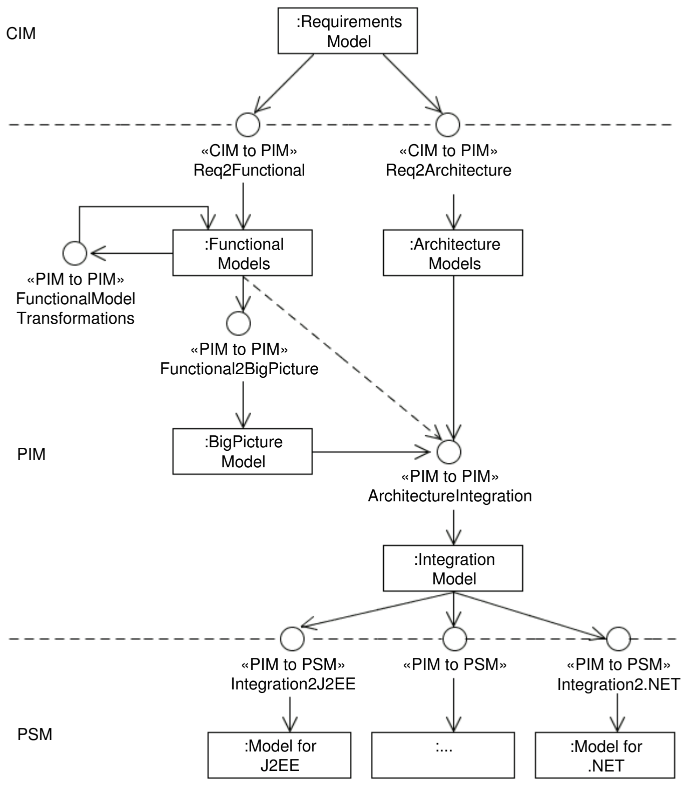

CIM
:Requirements Model
«CIM to PIM» Req2Functional
«CIM to PIM» Req2Architecture
:Functional Models
:Architecture Models
«PIM to PIM» FunctionalModel Transformations
«PIM to PIM» Functional2BigPicture
PIM
:BigPicture Model
«PIM to PIM» ArchitectureIntegration
:Integration Model
«PIM to PSM» Integration2J2EE
«PIM to PSM»
«PIM to PSM» Integration2.NET
PSM
:Model for J2EE
:...
:Model for .NET

Figure 3.1: With UWE, a series of model transformations is used to turn abstract models into more concrete ones, e.g. for a technology like J2EE. [Koch06Transformations]

only plays a minor role. This may be deliberate because the original work on the Unified Process already touches on many usability-related aspects – however, obviously without a look at the domain of web applications.

UWE Models

UWE uses lightweight UML profiles to define its models. With stereotypes, meanings are assigned to the different aspects that are modelled. Stereotype icons are provided to make it easier for the developer to distinguish between stereotypes. Where appropriate, the standard UML modelling facilities are used instead of specialized solutions. For example, the implementation is modelled using normal class diagrams. Moreover, the standard UML use case diagrams are available, and modelling of processes or tasks is achieved with activity diagrams. At the more concrete level of modelling, i.e. that of the platform-independent models which describe the web application, three major different kinds of models exist:

Conceptual model: This model is used to describe the data structures of the application from the programmer's point of view. A normal UML class model without UWE-specific extensions is employed for this task.

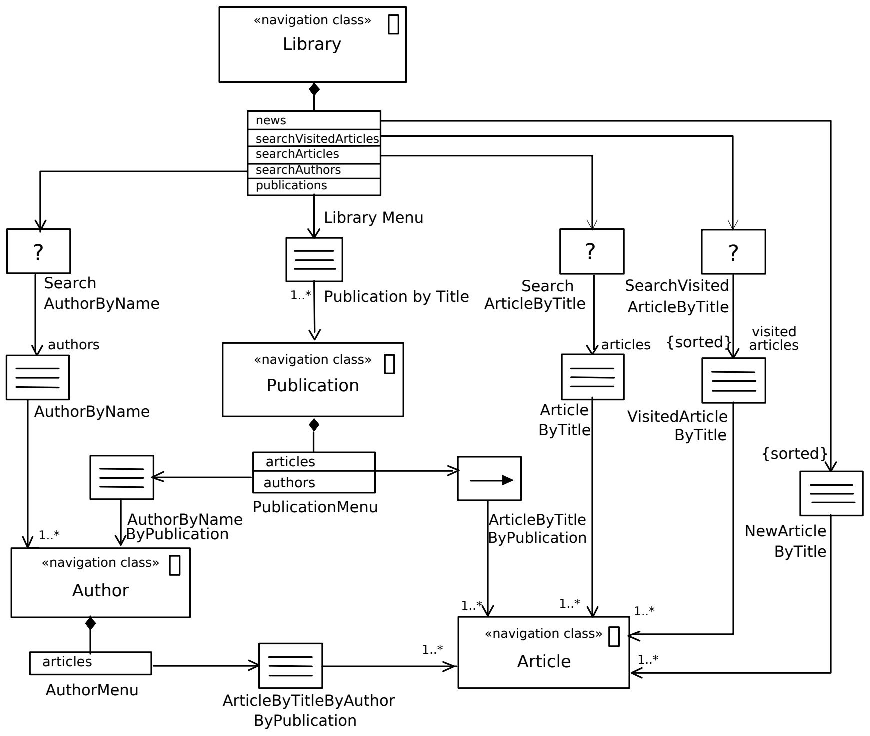


Figure 3.2: Example for a UWE navigation structure model. Stereotype icons are used instead of the names «index», «guided tour», «query» and «menu». [Koch02Expressive]

Navigation model: Hyperlinks are a feature which sets the web apart from other types of applications. With UWE, two different types of models specify navigation of users between different parts of the application: The *navigation space model* contains details on the different types of objects that can be visited. The classes used by the model are typically a subset of those given in the conceptual model from the “implementor’s point of view”. However, there is a restriction to those classes that will be displayed to the user, and associations in the model represent navigation possibilities rather than data references. The *navigation structure model* (figure 3.2) is a refinement of the navigation space model. It describes in greater detail how the user can reach objects, including information on patterns (such as “guided tour”) which characterize the way navigation should take place.

Presentation model: This model (figure 3.3) is a UML composition diagram. For each type of web page (e.g. homepage, article view), stereotypes in the model illustrate the different types of data which should appear on the page. A special extension of the UWE presentation model is that the exact positions and sizes of diagram elements have meaning: The model determines not only which parts make up the page, but also the page composition, i.e. where to place the page elements.

There are provisions for adapting content to properties of the user (e.g. tasks, knowledge, background, interests) and to the context of use (e.g. location of the user, hardware platform,

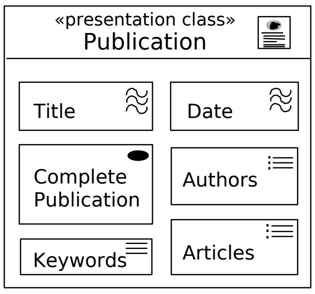


Figure 3.3: The UWE presentation model for a certain type of web page describes the parts that make up its content. [Koch02Expressive]

type of network access) [Baumeister05Adaptivity]. Consequently, a context model and a user model exist.

In the context model, the developer specifies the characteristics of the circumstances under which an application is used. This includes time and place of use and aspects of the rendering device and technologies that are used. There are no further details about the context model, for example no description of the exact properties that are modelled.

The user model is explained in [Koch00Adaptive, chapter 3] from the perspective of the primary domain of earlier UWE work, hypermedia e-learning applications. As a consequence, the user model concentrates on the representation of a learner's knowledge about the topic that the system teaches. Modelling of other user properties is not mentioned, for example their age or literacy. As UML is the basis of all models, it would be possible to add such information to the models if the need arises.

UWE Tool Support

As all UWE models are fully integrated into UML, including the respective meta-models, lower-level models or code can be generated from the models by employing standard modelling tool solutions such as MagicDraw UML (`http://www.magicdraw.com`). Earlier UWE work, which predates the availability of today's powerful commercial UML tools, also included prototypes for other tool concepts:

A tool named SmexWeb [Albrecht00SmexWeb] is provided as an example for an e-learning application, which was developed using the described method with the aim of adapting content depending on the user or context. It adapts web pages by changing the order of links, annotating them, deleting existent links or introducing new ones [Baumeister05Adaptivity]. At the time of writing (July 2007), the version found on `http://smexweb.pst.ifi.lmu.de` is not fully functional.

In the form of ArgoUWE [Knapp04ArgoUWE], the prototype of a modelling application is available – see figure 3.4 for a screenshot. It is based on the open-source ArgoUML editor and features special support for the creation of UWE models, using activity models to describe workflow-driven web applications, and an automated design critic for UWE models which can

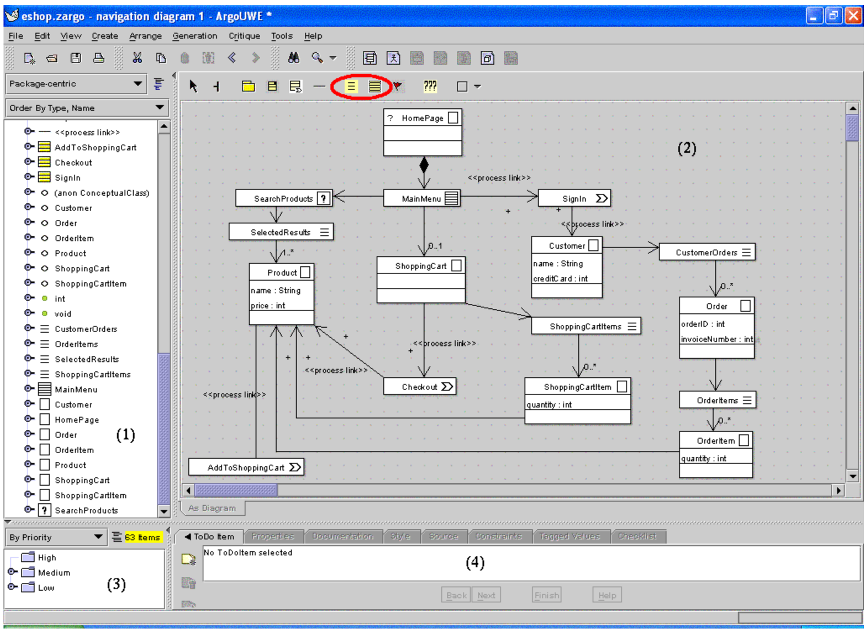


Figure 3.4: ArgoUWE allows the user to create models for UWE. It also features a critique functionality which highlights potential problems in the models. [Knapp04ArgoUWE]

alert the user to inconsistencies or deficiencies in the models. A wizard can help the user to correct mistakes that he has made.

Finally, the generation of web applications from UWE models is also possible using an XML-based framework called UWEXML [Kraus02Generation].

The current UWE tools do not have features designed to help the developer to improve the usability of the website. ArgoUWE's design critic could be used for this purpose, but the implemented critique algorithms concentrate on other aspects, e.g. well-formedness constraints of UWE models.

3.2.2 WebML

The WebML graphical notation, together with the accompanying development process and methodology, is an approach which is based on the assumption that most web applications are data-intensive – they can be regarded as front-ends to databases. Whereas UWE's roots in software engineering are recognizable in its models and method, the background of WebML is that of database management systems (DBMS): In the main WebML reference [Ceri03WebML], entity-relationship (ER) models are the basis of the graphical notation (although UML is also used), and the authors dedicate space to the use of ER modelling for organisation of the application's data, but not to the modelling of e.g. the class layout of an implementation. Furthermore, a textual definition of all WebML concepts is provided [Ceri03WebML, p. 79, app. B], it may have been inspired by SQL.

WebML Development Method

The WebML development process consists of a number of phases, it builds upon the classical spiral as well as modern methods [Ceri03WebML, section 6.4]. During a number of development cycles, the application is incrementally refined until it meets the expected goals. Feedback loops are possible, allowing the developer to go back to the previous phase of the current cycle. The following activity phases exist:

Requirements specification: The developer describes the desired properties of the application. The steps include the usability-related principles of identifying the prospective users and device types (e.g. mobile phones), and adapting the application for different user groups.

Data design: With standard ER modelling practices, the database layout of the application is specified.

Hypertext design: From the functional requirements, the developer creates the structure of the web application: Pages with information and links, and site views for different user groups.

Architecture design: Based on non-functional requirements and other constraints, it is decided which hardware, network and software components to use for the implementation.

Implementation: Code is written to transform the data and hypertext models into the web application. Furthermore, page templates determine the layout of HTML pages.

Testing and evaluation: Apart from functional and performance testing, there is also a usability testing step, possibly with different tests for each site view. A small section on usability [Ceri03WebML, section 9.6] contains advice on interaction patterns, navigation and consistency of design.

Maintenance and evolution: After the application has been made available to the end users, necessary changes to the data or hypertext model are applied and the documentation is kept up to date.

It should be noted that two tasks which may have a large impact on the usability of the website are only carried out in a quite late development phase: The creation of page templates and the evaluation of the interface through usability tests happen after the design phases. In itself, this is not a problem, as the WebML process is iterative. However, the developer should take care to use initial development cycles only to produce prototypes, otherwise design changes caused by usability problems could become expensive to fix. [Ceri03WebML] does not discuss this necessity. Generally, usability is not of significant interest for existing WebML work; other than the short list of usability hints in the context of testing and evaluation, it does not describe steps to improve applications in this regard.

WebML Models

Most of the WebML models are extended entity-relationship models. UML is used for some tasks like use case modelling. In addition to this, [Ceri03WebML] discusses alternative UML modelling options for all WebML models.

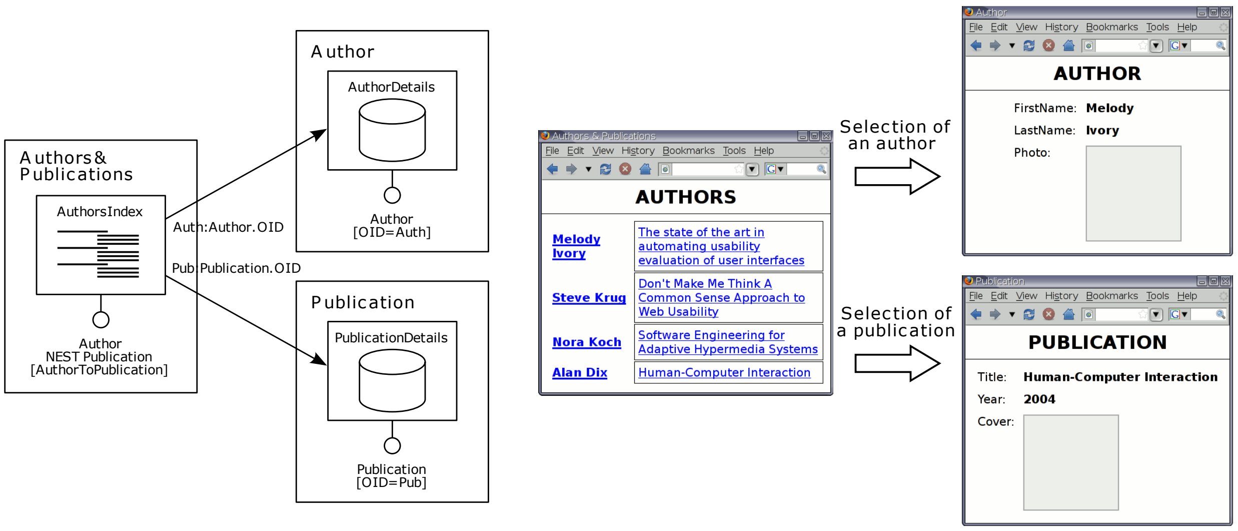

Authors & Publications
AuthorsIndex
Author NEST Publication [AuthorToPublication]
Auth:Author.OID
Pub:Publication.OID
Author
AuthorDetails
Author [OID=Auth]
Publication
PublicationDetails
Publication [OID=Pub]
Authors & Publications
File Edit View History Bookmarks Tools Help
AUTHORS
Melody Ivory
The state of the art in automating usability evaluation of user interfaces
Steve Krug
Don't Make Me Think A Common Sense Approach to Web Usability
Nora Koch
Software Engineering for Adaptive Hypermedia Systems
Alan Dix
Human-Computer Interaction
Selection of an author
Selection of a publication
Author
AUTHOR
FirstName: Melody
LastName: Ivory
Photo:
Publication
PUBLICATION
Title: Human-Computer Interaction
Year: 2004
Cover:

Figure 3.5: WebML hypertext model (left) with corresponding HTML output (right). The model specifies pages, possible navigation paths between them and the "units" displayed on pages – here a hierarchical index and two data units. [Ceri03WebML, p. 97]

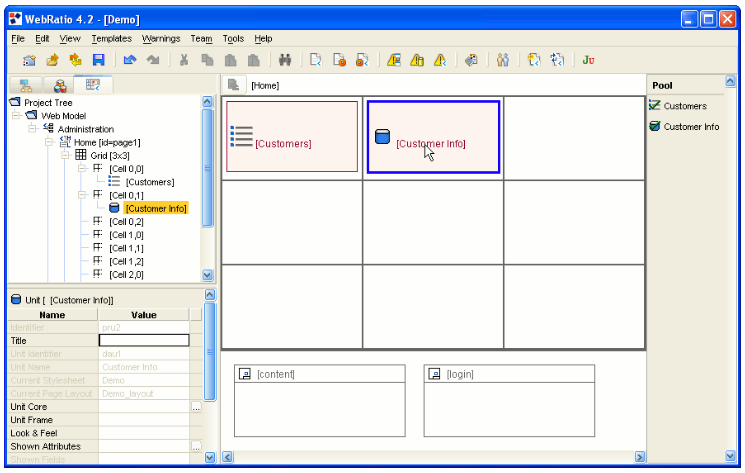
Data model: This model is used for the database part of the application. Essentially, it is an unmodified entity-relationship model.

Hypertext model: The user interface of the application is described with this model, including organisation of pieces of content (called "units", e.g. a link list or text input field, see figure 3.5) on the different pages that make up the application, and navigation possibilities between these pages. Pages are then grouped hierarchically into "site views" (e.g. one for each user group) and "areas" (clusters of related pages).

Content management model: The notation of this model is not separate from the hypertext model, but an extension to it: It allows the developer to annotate a link with the action taken by the application code during traversal of the link. In contrast to the hypertext model, the focus is not on the organisation of the user interface, but on modification of database entries, so-called "operations". The DBMS background becomes visible in details like cascaded delete operations, which translate directly into SQL constructs.

Presentation model: Only listed as a separate model in some sources [Ceri00WebML], this term denotes the page templates and their style sheets. The templates are written using an XML syntax, there are both generic templates which apply to many pages and templates which only apply to a single page.

The notation is well-suited for modelling both from the point of view of the user (hypertext model – what types of interaction are possible on each page?) and the implementation/database (content management model – what happens when the user navigates to another page?). Furthermore, by using the concept of "areas" and "site views" to create groups of pages (or of



Source: `http://www.webratio.com/Demo.do`, accessed 8 August 2007

Figure 3.6: WebRatio consists of a set of tools for WebML. The user interface editor allows editing the layout of a web page, the tool turns this into an XML page template.

further areas/site views), a hierarchical organisation of websites into different parts is possible [Ceri03WebML, chapter 3].

Patterns like "guided tour" exist in WebML models. However, they are less useful than the corresponding, more abstract UWE constructs because they only apply to a single "unit". For example, a "guided tour" allows browsing a list of objects, but not a series of pages with input fields.

The modelling concepts also include support for rich Internet applications (RIAs) [Bozzon06ConceptualRIA], e.g. to express the difference between content which is created on the server side and content which is created on the client side.

WebML provides a simple user model in which users are assigned to user groups with common properties [Ceri00WebML, section 5.2] and which stores usage information such as a list of recently purchased items in an online shop. The model does not contain information like user age or literacy, but could probably be extended to allow for it. A model describing the context of use (device, location etc.) was later proposed in [Ceri07MDDContext] – see section 5.1.

WebML Tool Support

Tool support for WebML has been commercialized and is available under the name WebRatio (`http://www.webratio.com`). The offered features include creation of WebML models, automatic creation of an implementation for the J2EE platform, and synchronisation of data models and database schemas. An extension also supports simple AJAX-style user interface concepts, e.g. drag & drop. The screenshot in figure 3.6 shows the editor for page templates.

As far as usability support is concerned, WebRatio does not appear to provide dedicated tools which aid the developer in improving the application. A positive aspect is the integration of the

HTML Tidy tool (see page 35): When existing HTML pages are imported into WebRatio as page templates, it can clean up the HTML code and make it standards-compliant. The software is built in a modular way, with provisions to extend its functionality through additional extensions. Thus, it is imaginable that a module with usability tool support is added.

3.2.3 OO-H

OO-H (Object-oriented hypertext) [Gomez01Conceptual] is a project which applies the results of a more general methodology and modelling approach, the OO-Method [Pastor97OOMethod], to the domain of web engineering. It consists of a method, associated models and prototypical tool support. Similar to UWE, it is UML-compliant and influenced by the MDA approach with its model transformations, but provides a solution tailored to the creation of web applications.

OO-H Development Method

OO-HMethod uses a simple design process to create the different models, which is based on OO-Method. Development is founded on user requirements which are determined early in the process [Gomez01Conceptual, section 5]. The design process [Gomez01Conceptual, section 3.1] starts with the creation of class diagrams which describes the concepts that the web application works with. Next, navigational models are created for each user type. From these models, a default web interface is generated in a number of automatic steps. The default interface is refined using another model which describes presentational aspects of the web pages.

The description of the development process is low on details – essential phases like testing are not even mentioned. Moreover, there is no information on other steps which may be performed to improve the usability of the web application.

OO-H Models

Similar to other approaches, a number of different models exist to describe the various aspects of a web application. OO-HMethod extends OO-Method with two new diagrams, NAD and APD:

Domain model: Only referred to as "class diagram" and "domain classes" in [Gomez01Conceptual], this UML model describes the domain information structure. (It is comparable to the "conceptual model" of other web engineering solutions, but that term is used for the NAD below by some articles.)

Navigation Access Diagram (NAD): One NAD is created for each view that is to be provided by the system. For instance, different NADs may be created for different user groups or different output devices. Figure 3.7 shows such a diagram. It consists of navigation classes (which include the domain classes, extended with special attributes), navigational targets (in the figure, only a single one named "ParticipateInDList"), navigational links (arrows, available in several variants) and collections.

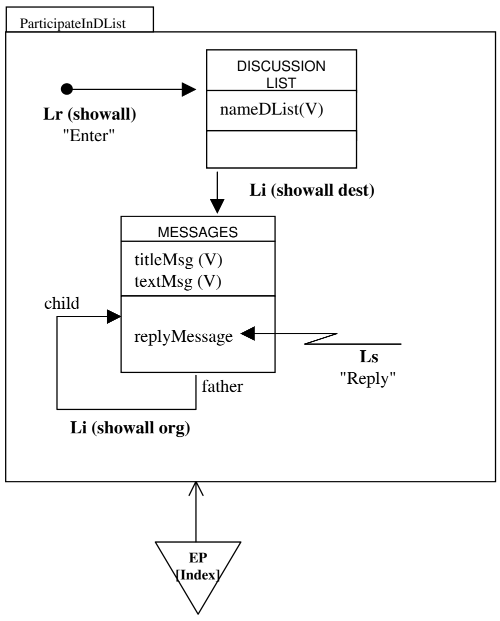


Figure 3.7: The OO-H Navigation Access Diagram (NAD) specifies the navigation possibilities between navigational classes. [Gomez01Conceptual]

Abstract Presentation Diagram (APD): [Cachero00PresOOH] This model is a refinement of the Navigation Access Diagram, it specifies the visual layout of the pages as well as their structure. There is one model for all pages in a view (rather than one model for each type of page, as with e.g. UWE). Five different kinds of objects are possible, each of which corresponds to an XML template that describes one aspect of the presentation. The five template types are `tStruct` (which content objects appear on the page), `tStyle` (visual formatting), `tForm` (interaction possibilities), `tFunction` (client functionality, e.g. JavaScript) and `tWindow` (available views). In figure 3.8, an APD which has already undergone refinement is shown. For example, each of the pages references the same kind of page header (`«tStruct» head`), which contains links to the home and `DList` pages.

It can be observed that the Abstract Presentation Diagram is elaborate and suited for modelling many aspects of the application, not just presentational ones. At the same time, it cannot get too large: On one hand, this is due to the fact that not too many details are modelled – for example, the exact positioning of content on pages is not specified, but left to the style sheet. On the other hand, if the web application consists of many pages, it is possible to organize the information in a hierarchy of diagrams to prevent any single diagram from becoming incomprehensible because of its complexity.

The OO-H models are complemented by a catalogue of patterns which provide standard techniques for typical problems encountered during modelling. Because they are a way of specifying common practice and established solutions, many of these patterns can be considered beneficial for the usability of the website. [Gomez01Conceptual] lists the example of the "Location Pat-

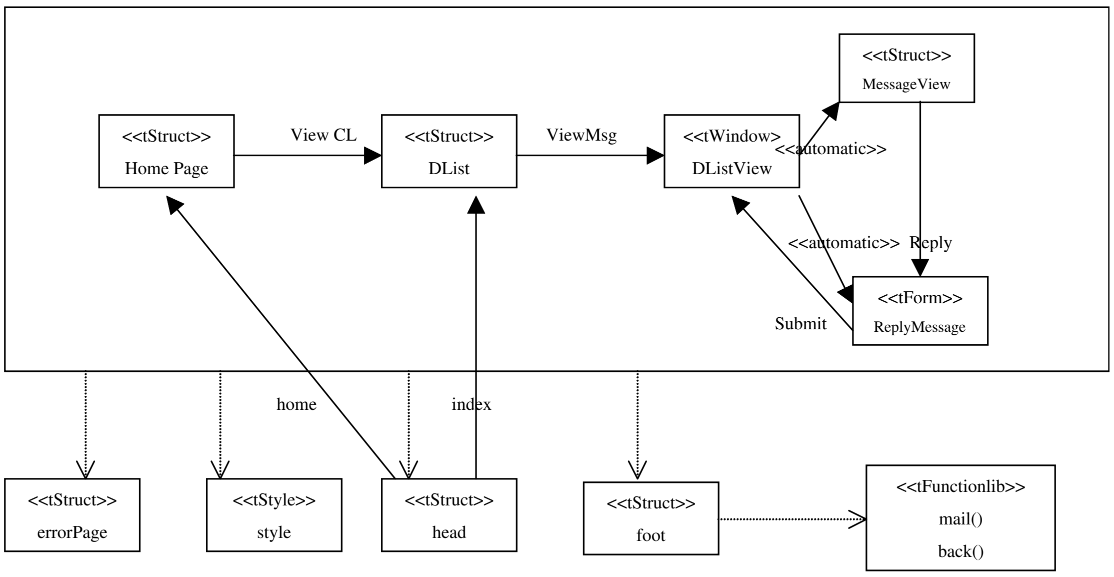


Figure 3.8: The OO-H Abstract Presentation Diagram (APD) contains details on the structure of pages, visual formatting, but also inter-page navigation. [Gomez01Conceptual]

tern", whose transformation rule will add a header and footer to each page to ensure that the resulting consistency gives the user clues about pages which belong to a certain view.

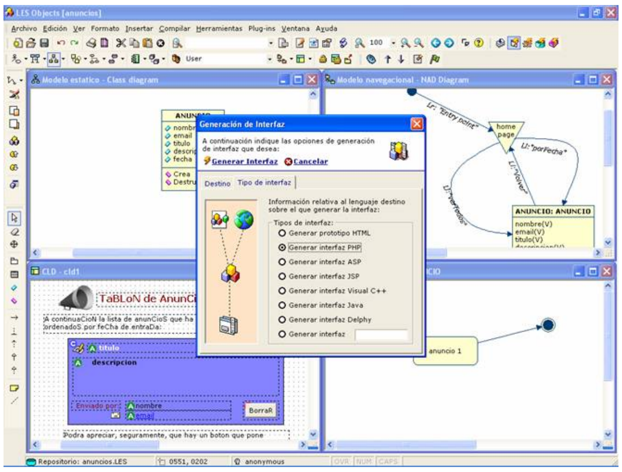
In the available literature, different user groups are mentioned frequently, but no explicit user model with a fixed notation and user properties is introduced. Similarly, references to device independence are made, but the context of use (including the type of the device) is not specified in a model. Typically, the developer is advised to create e.g. one NAD model for each user group, without further hints on what constitutes the concept of a user group.

OO-H Tool Support

The OO-H project has created a so-called CAWE (computer-aided web engineering) tool which helps with many tasks faced by the developer during his work. It is called VisualWADE (`http://gplsi.dlsi.ua.es/iwad/ooh_project/cawetool.htm` – see figure 3.9). In addition to creation and maintenance of all the models of the OO-HMethod, the tool supports editing the final page layout, for example the positioning of elements, formatting of text, and similar aspects.

Improvements to the usability of the page are cited as one of the purposes of the tool. This "usability support" is present via the predefined pattern catalogue: For each pattern, the tool provides an implementation which is thought to provide the best possible usability for that particular type of interaction. This way, the developer will need to explicitly override the tool's suggestion before he can supply an alternative way of implementing the functionality.

VisualWADE can generate runnable code from the models after only relatively little modelling work. This allows the use of rapid prototyping to try out different approaches for the


Generación de Interfaz

A continuación indique las opciones de generación de interfaz que desea:

Generar Interfaz Cancelar

Destino Tipo de interfaz

Información relativa al lenguaje destino sobre el que generar la interfaz:

Tipos de interfaz:
- Generar prototipo HTML
- [✓ Generar interfaz PHP]
- Generar interfaz ASP
- Generar interfaz JSP
- Generar interfaz Visual C++
- Generar interfaz Java
- Generar interfaz Delphy
- Generar interfaz

LES Objects [anuncios]
Archivo Edición Ver Formato Insertar Compilar Herramientas Plug-ins Ventana Ayuda
Modelo estatico - Class diagram
Modelo navegacional - NAD Diagram
CLD - cld1
TaBLoN de AnunCi
home page
ANUNCIO: ANUNCIO
anuncio 1
BorraR
Repositorio: anuncios.LES
0551, 0202
anonymous

Source: `http://gplsi.dlsi.ua.es/iwad/ooh_project/cawetool.htm`, accessed 9 August 2007

Figure 3.9: VisualWADE offers tool support for OO-H. Its features include modelling, creation of page layouts, and generation of code for a basic implementation.

implementation and user interface. Rapid prototyping can result in increased usability of the final application because problematic designs can be identified and replaced early during development. Of course, the code generation facilities can also be used to create the basis for the final implementation.

3.2.4 Other Approaches

While UWE, OO-H and WebML are probably among the most advanced web engineering solutions, they are influenced by and build upon a number of previous approaches. The list below gives a short overview of some of them:

WSDM: The Web Site Design Method [DeTroyer00WSDM] calls its models "schemas". The different types are similar to other approaches: A conceptual schema describes the application, it is subdivided into a content schema (specification of objects and the relations between them), a structure schema (elements of a website and navigational links) and a mapping schema (for converting information from the content schema to site content and structure). "Meta schemas" (metamodels) are also provided.

HDM: The Hypermedia Design Model is introduced in [Garzotto95HDM]. It organizes information in entities which correspond to real-world concepts. They are arranged in collections and connected using different types of links.

HDM-lite: This model [Fraternali98HDMlite] builds on HDM and modifies it with the aim of facilitating easy automatic generation of web pages from models. There is a separation between navigation and presentation: A so-called access schema defines the navigation paths, a presentation schema provides style sheets for page formatting. The tools Visual-HDM and Autoweb help with modelling and page generation.

WAE and WAE2: With Web Application Extension [Conallen99WebUML], web application modelling takes place with UML, and development follows a method which is based on an established software engineering method for non-web software. WAE uses stereotypes to describe pages, links and page elements, in a way similar to UWE.

W2000 This approach is based on HDM, but uses UML for all models [Baresi01W2000]. Functional and navigational use-case models serve to capture the requirements, and a state evolution diagram describes the stages which are required by the application to reach a goal. The information design schema is adapted from HDM. An improved navigational model covers the interaction with web applications as well as simple links, it also includes support for design patterns.

RMM: The Relationship Management Methodology [Isakowitz95RMM] can be considered a predecessor of OO-H. Based on ER modelling, the Relationship Management Data Model specifies the layout of a database, and the Relationship Management Design Diagram the navigation associations between entities. Patterns like a guided tour are possible.

OOHDM: Not to be confused with OO-HMethod, the Object-Oriented Hypermedia Design Method [Schwabe98OOHDM] precedes work on OO-H. A special notation rather than UML is used for specifying object properties. Its Navigational Schema and Navigational Context Schema are powerful and can represent adaptation of the navigation possibilities, such as links that only appear for certain user groups. A so-called Abstract Data View describes the user interface.

3.2.5 Conclusion

Many makers of web engineering solutions have thought of usability in one way or another. For example, a user or context model is present, patterns like a “guided tour” allow outputting a user-friendly default interface for the respective interaction type, or user testing takes place as part of the method. However, in the area of tool support, efforts have so far concentrated on support for a model-driven architecture approach, with automatic generation of an implementation (or in general more detailed models) from the developed web engineering models. Other types of tools include support for user interface design, but not usability or accessibility validation, nor model-based usability support as proposed in later chapters of this thesis.

Existent web engineering tools already provide many parts of the needed infrastructure (e.g. a GUI framework and modular extension mechanism), so adding usability tool support is potentially easier than if the tool support had to be implemented from scratch.

Chapter 4

Method and Tool Concepts for Improved Web Usability

Based on established development methods described in the literature ([Dumke03WE], [Ceri00WebML], [OMG03MDA], [Constantine99Software], [Dix04HCI]), this chapter proposes a number of ideas for method improvements and usability tool support for different phases of the design process. It draws on two different branches of research, combining their advantages: On one hand, a model-driven architecture (MDA) allows for advanced tool concepts due to the presence of models, and on the other hand user-centered approaches pay a lot of attention to the user and his interaction with the system. In general, the available literature tends to concentrate on either of the two fields, and includes some aspects of the other field in its discussions. However, to the best of the author's knowledge, no previous work exists which tries to use the concepts of model-driven development for usability improvements throughout the development process.

The development process is not described in its entirety in this chapter – instead, the focus is on the individual steps, such as modelling, prototyping and user testing. For each step, an analysis of the tasks that the developer or website maintainer must perform is followed by suggestions of what aspects of the step could be made more efficient with the help of special tool concepts.

The presented tool ideas differ noticeably from one another; while some can be considered extensions of the tools which are available for MDA-based development, others simply automate parts of a work-intensive development step, such as conducting a usability test, and yet others rather provide a discussion of how to adapt tools for better usability, e.g. by assigning more importance to the usability of the default tool output. Sometimes, this chapter encourages the use of already existing tools for a different purpose. For example, section 4.1.2 describes how existing web questionnaire software could be used for an extended, web-based requirements elicitation phase. Later chapters will describe two tool concepts in greater detail, including prototype implementations.

At first sight, it may seem that automated or semi-automated means to improve usability are problematic, due to the fact that the usability of the final web application is dependent upon many "soft" factors such as the background and subjective opinion of the user group. In fact, it *would* be impossible to replace measures like user testing with a tool which automatically eliminates all usability problems, but this is not the aim. Instead, a number of principles lead to a development process which places more stress on web usability, but which uses tools to streamline the development:

- Use models as an input for usability validation tools to improve their accuracy.
- Ensure a basic level of usability with automated analysis of the models and implementation.
- User tests are still necessary for less obvious usability problems – automate the process of conducting and evaluating them.
- Let users participate in the design process [Constantine99Software, chapter 20]; enable a tool-supported, two-way communication between them and the developers.

4.1 Requirements Analysis Phase

Summary: For many web applications, insights from the field of HCI can be applied to software engineering practices when gathering requirements. Certain types of web applications are more problematic – for example, nothing is known about the users and their wishes in advance. In the latter case, web-based tools and approaches can be used to gather information.

Requirements engineering is an important phase as far as the usability of the application is concerned. If the developer fails to identify the users' wishes at this point, they will not be satisfied with the final product: Even if its user interface is easy to understand, they will be unable to achieve their goals with it.

In software engineering and web engineering, the requirements analysis phase involves aspects such as defining the functional and non-functional requirements (the features and behaviour of the application), the required properties of the used platform (in terms of the hardware components and software technologies to use), and any process-specific requirements (e.g. necessary financial and human resources) [Dumke03WE, section 3.2], [Balzert96SE, chapter 1]. Thus, traditionally the application and its functionality tend to be the main focus during this phase of development.

Applying Existing Analysis Approaches to Web Engineering

For user-centered web engineering, it is necessary to give more thought to the users and their needs. This can happen by applying development principles which were developed by the HCI (human-computer interaction) community [Rosson02UsabEng]. In recent years, they have also been adopted in the field of requirements engineering [Robertson06RE]. They include interviews or workshops with prospective users and other stakeholders, prototyping for early tests with users, creation of personas, formulation of scenarios (a more user-oriented form of use cases [Rosson02UsabEng, p. 19]), and other similar approaches. Employing these procedures during the development of web applications is possible in most cases.

Easily usable UIs can be considered a non-functional requirement [Dumke03WE, section 3.3.1], so an attempt should be made by the developer to specify criteria in the requirements document which ensure that good usability will be achieved in the final product. For example, a certain level of satisfaction may have to be documented in a poll of the final users of the application, or compliance with the WCAG guidelines (section 2.2.1) may be demanded. Guideline compliance requirements are influenced by properties of the users (e.g. what disabilities may be present among the users?) and legal considerations (e.g. must BITV or similar legislation be followed?). In addition to conformance with guidelines, constraints like "buying an item on the website should take the average customer at most ten minutes" should be included. Many of these constraints can later be incorporated into the developed models.

However, it is vital not to reduce usability to user interfaces that are easy to understand and to use – it should also be ensured that the information behind the user interface is organized in the right way, that all possible goals are reachable in an efficient manner, etc. For web applications, it makes sense even at this early stage to keep the special navigation conventions of websites in mind; since the beginning of the web, they have been evolving towards de-facto standards which should not be ignored to avoid surprising the user. The use of patterns [Graham03Wu] can help with this, for example by requiring the developer to classify the site (e.g. "company site", "educational", "news"...) and then to develop the appropriate conceptual organisation based on this decision.

In many cases, the newly developed system is intended to replace an already running system with an existing user base. Asking these users about their expectations for the new system is a valuable resource during requirements elicitation. Usually, personal contact with the users will be the easiest and most effective way of gathering requirements. Apart from interviews, the developer can also do this by recording "day in the life" scenarios (i.e. observing an existing user's work routine for one day), or with focus groups (roundtable discussions) [Constantine99Software, chapter 16].

Difficulties of Requirements Analysis for Certain Web Applications

As outlined above, requirements analysis for web applications does not differ significantly from that of other types of programs as long as the application is written for a customer who has approached the developer with his wishes. However, development of a significant number of web applications begins under premises which are quite different from those assumed in software engineering:

- The site will be available on the public Internet rather than in a closed environment.
- The target users are not known in advance, or only inexact predictions about them are possible.
- If distinct user groups exist, they will all use the same website – it is not possible to deliver different versions of the site to different users until after they have declared which user group they belong to.
- The users are more impatient than with other types of software.

- The site must compete with other, similar offers which are only a click away.
- Conflicting goals may force the developer to deliberately reduce usability. For example, a website may have to display advertisement despite the fact that this is unpopular with most users. In a similar vein, the site may only offer basic functionality for free, and may have to stop the users just before they reach their goal, with a "you must pay to proceed" message.

A typical example is that a company wants to introduce a new web-based service, with the hope that enough visitors will be attracted for development to pay off, but without an existing user base for this particular service.

Together, the factors above can pose a risk for the success of a new web application. While some problems, such as the last point, must simply be taken for granted, others can be tackled. In particular, with appropriate tool support, information about the required features and the users can be gathered. In many cases, web technologies can be used for this task – two examples are given in the remainder of this section.

4.1.1 Using Activity Logs for Web-Based Requirements Elicitation

Web applications which are publically available on the Internet attract users from different countries, with different backgrounds and varying needs. As a first step towards gathering information about these needs, web server log analysis can be employed for applications which are similar in purpose to the planned system. For example, this includes the working predecessor of the planned web application, or a competitor's service with similar functionality. However, it should be stressed that log analysis can only give relatively vague impressions about the user's needs for a new application which does not yet exist. Whenever possible, it should be combined with other, more direct means of requirements elicitation.
Using log analysis, a number of things can be found out about the users and their demands:

- If a user arrived on the website via a search engine query, the words of this query can be obtained through the "`Referer`" HTTP header which is transmitted by the user's browser. While the most frequent search queries will usually not be a surprise for the developer, less frequent queries may provide interesting thoughts about possible new features or extensions to the system.

- If the application offers an online help system with a search facility, its log of search queries can provide valuable data about tasks that the users want to perform with the application, and problems they encounter during the process.

- From the IP addresses of visitors, different conclusions can be drawn: With good confidence, a mapping from IP to country name is possible. Furthermore, with acceptable accuracy, IP addresses can be classified as belonging to company networks or dial-up/DSL providers, which allows distinguishing between users who use the application at work and users which do so at home.

- Together with the country name, the timestamp of requests can give an approximate idea about the local time of day when the application is used. This can further help to distinguish work from leisure time.

More detailed analysis, e.g. collecting multiple requests from a single IP in a session and analysing navigation patterns, duration etc. may be less useful, as it is strongly tied to the currently existing application and does not provide clues about the user's expectations from any future system. However, analysing the "information scent" (see page 37) of links and pages may be helpful to understand the navigation flow and the actions of the users on the existing site.

4.1.2 Using a Support Website for Web-Based Requirements Elicitation

If the number of users of an existing web application is high or the user population is spread geographically, it would be expensive to personally collect feedback about the users' expectations for a new application. In other settings, personal contact is not possible at all, such as when the users are the general public from all over the Internet. In these cases, it should still be attempted to conduct focus groups, "day in the life" scenarios etc. However, if this proves difficult or can only be done for a fraction of the actual users, the results of these measures can be supplemented by information from questionnaires which are passed to users. Web-based surveys are a simple and effective means for this. Common questionnaire programs like phpESP (`http://www.butterfat.net/wiki/Projects/phpESP/`) help with their creation.

Statistical analysis of questionnaires can already answer many questions. For instance, based on a catalogue of features which are rated by the survey participants, the relative priority which should be assigned to features can be estimated. In addition to this, free-text comments enable users to suggest further requirements which the developer may not have thought of.

As an extension to this concept, it is imaginable that not a simple single-way interaction between users takes place (in the form of a questionnaire and the answers to it), but that a dialogue is started in which prospective users and developers can discuss ideas, e.g. explain the need for certain features or any reasons which would make the implementation of a particular feature difficult. This thought of creating a strong link between a number of users and the developers is based on the "Web 2.0" idea of encouraging collaboration and allowing normally passive end users to form a community and to participate in the creation of new concepts. At the same time, it is an extension of the idea to let a selected group of users participate in a beta test phase of a new product, with the difference that they are included in the development process at a much earlier stage.

The collaboration platform could be based on a number of different concepts: It can be a discussion forum as provided e.g. by the phpBB software (`http://www.phpbb.com`), an email-based mailing list, a "developer blog" with comments added by users, a "trouble ticket" system like BugZilla (`http://www.bugzilla.org`) which allows systematic tracking and discussion of individual user requests, or even a special-purpose tool which combines those aspects of the other types of software which are most relevant for gathering requirements.

A detailed study would be required to determine under which circumstances the concept outlined above would be successful. A number of potential problems exist:

- There must be users who are motivated enough to participate. Incentives in the form of small prizes (say, electronic gadgets) are probably not the best way, as they will only attract people who are interested in the prize, not the application. Instead, users with a real interest in the application should be motivated by the fact that their comments can make a difference.
- The opinions of the collaborating end users may not be representative for all future users – instead, these users may be early adopters or technically knowledgeable persons who differ considerably from later users.
- Long periods of inactivity from the side of the developers may cause users to abandon the collaboration platform – they may no longer be interested when a new prototype becomes available. A more open form of development (progress reports, public planning of future steps) could counteract this.
- As a result of the above points, the company developing the new web application would have to make significant parts of its plans public early during development. This may not be allowed because of concerns over being copied by competitors, or being unable to meet deadlines or expectations.

In conclusion, requirements analysis for some types of web applications brings with it significant challenges, as it is hard to make contact with prospective users to record their expectations for the new application.

4.2 Design Phase

Summary: During the design phase, tool support for quick prototyping can result in shorter development cycles. Furthermore, modelling tool support can improve the usability of the result, for example by suggesting default transformations which take usability into account, or by continuously evaluating the models during their creation and immediately alerting the developer when he is about to make a mistake.

In the design phase, the developer makes many decisions which have a fundamental influence on the functionality and behaviour of the web application. The result of his work is a number of models detailing most aspects of the application. When looking at this phase with usability in mind, the most important aspects of this work are the conceptual model ("conceptual" in the HCI sense, see page 4 and the next paragraph) which the application conveys to the user, and the user interface, in the form of interlinked web pages which contain controls.

In order to make it worth the effort for the developer to create models, usability-related tool concepts for this phase of development concentrate on processing them, preferably in a fully automatic way. The conceptual model, which is not specified directly by the developer but only forms in the mind of the person using the application, is hard to estimate by a tool. Thus, in practice, tools will concentrate on the models that are actually specified, i.e. the web engineering navigation and presentation models. Furthermore, the tools will help the developer to conduct user tests with prototypes to ensure that the conceptual model works. (The developer *does* create a conceptual model, e.g. in the form of a class diagram for objects maintained by the application, but this may not be related to the conceptual model perceived by the user. The difference in meaning of this term in the web engineering and HCI communities is particularly confusing here.)

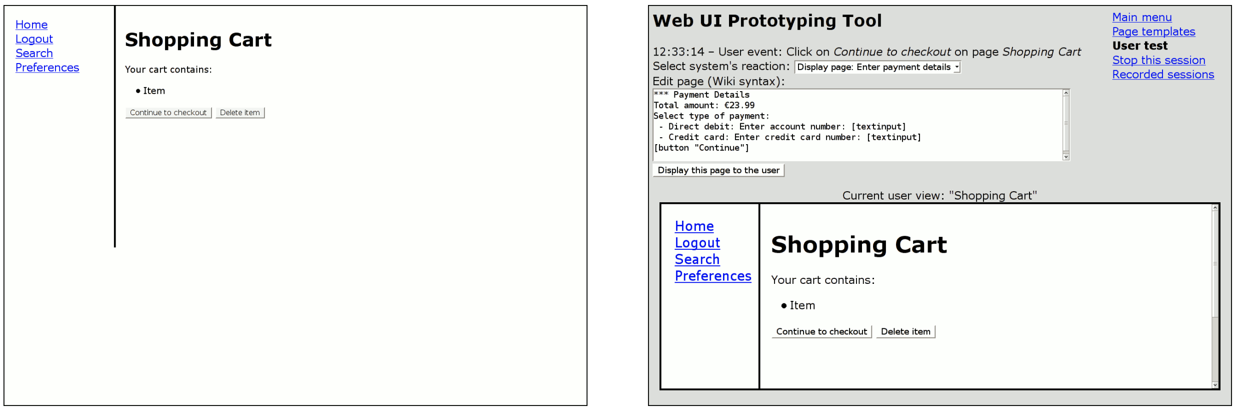
Three ways of working with the models are illustrated in the rest of this section: First, high-level models like use cases can be the foundation for building quick prototypes. Second, during automatic generation of lower-level modules or even the implementation, it can be of great advantage if the tools provide default transformations which have been carefully designed to incorporate common usability guidelines, thus requiring specific actions of the developer to deviate from the "safe defaults". Finally, it is possible to analyse the models themselves with regard to known usability problems. The rationale for this is that alerting the developer to possible issues should happen as early as possible to save time and effort.

4.2.1 "Virtual Paper Prototyping" of Web User Interfaces

It is often beneficial to try out various fundamentally different user interfaces, e.g. to test how well different conceptual models (in the HCI sense) are received by test users. In web engineering, an MDA approach to prototyping is usually preferred (e.g. [Gomez01Conceptual]), i.e. a model is created, then a first implementation is generated from it. User-centered methods propagate the "Wizard of Oz" technique or paper prototyping [Constantine99Software, chapter 10]. With paper prototyping, a test user is presented with a state of the user interface drawn on a piece of paper, indicates the action they want to take (e.g. clicking on a button), and is then shown the new state on another piece of paper. Before the test, the usability expert will prepare a number of sheets with expected states. For unexpected user actions, the expert can create ad-hoc drawings during the test on paper while the user is waiting.

For web applications, an alternative to paper prototyping is imaginable: A web-based tool can provide a very rough default page layout (e.g. header, navigation, main text) which is not normally modified – this is comparable to an empty sheet of paper. The tool then allows the developer to prepare a number of pages with predefined, fixed content. Apart from adding text content, it would probably be useful to allow simple drawings to be made on the page with the mouse or another input device.

Figure 4.1 shows a sketch of what a user session with the web-based tool might look like: The test user and the usability expert access the tool from two different computers. The user is only presented with the user view (left), whereas the expert view includes both what the user sees and a special authoring mode for pages (right). Interaction of the test user with the page is usually manual, in the sense that when the user e.g. clicks on a link or button, the expert is asked what page should be displayed next. Apart from selecting one of the prepared pages, the expert can also create a new page in real time, just like with proper paper prototyping. Possible advantages of this tool concept include the following:


Home
Logout
Search
Preferences

Shopping Cart

Your cart contains:

- Item

Continue to checkout | Delete item

Web UI Prototyping Tool

12:33:14 – User event: Click on *Continue to checkout* on page *Shopping Cart*
Select system's reaction: Display page: Enter payment details
Edit page (Wiki syntax):

```
*** Payment Details
Total amount: €23.99
Select type of payment:
 - Direct debit: Enter account number: [textinput]
 - Credit card: Enter credit card number: [textinput]
[button "Continue"]
```

Display this page to the user

Main menu
Page templates
User test
Stop this session
Recorded sessions

Current user view: "Shopping Cart"

Home
Logout
Search
Preferences

Shopping Cart

Your cart contains:

- Item

Continue to checkout | Delete item

Figure 4.1: Sketch of a web-based system for "virtual paper prototyping". The user sees a simple UI (left), the developer can influence system behaviour on the fly (right).

- Compared to paper prototyping, the technology used for the UI and the interaction with it (widgets, point-and-click) is close to what will be used for the final version. This way, problems that are specific to web applications (e.g. loss of orientation due to missing visual clues) might have a higher chance of being detected.

- It is possible to use predefined website "themes" with little extra effort. For example, the prototype could display the screen layout and graphical design of `amazon.com`. However, it would have to be evaluated whether ornaments on the displayed web pages are useful, as they may distract the user from the concepts that are to be evaluated during the test [Constantine99Software, page 497].

- Logging/recording and possibly even replay of test sessions is possible with little effort. With traditional paper prototyping, this is only possible using audio/video recording and manual analysis of the material. Simple measurements like task completion time could also be supported, but at this early stage, these numbers will usually have little significance.

- The different pages generated by the tool, together with the navigation paths taken by test participants, can provide the basis for a web engineering navigation model.

- Remote sessions are possible – a positive aspect e.g. if most members of the user group live far away from the developer.

Some existing work discusses computer-supported sketching of user interfaces for prototyping, e.g. [Lin00DENIM], [Bailey03Informal], but not the application of this technique to paper prototyping.

4.2.2 Usable Model Transformation and Code Generation Defaults

One of the main advantages of model-driven development is that it is possible to quickly and automatically create an implementation from the models that describe the application. Depending on how far work on the models has proceeded, they can be detailed enough to provide the basis for work on the final implementation. However, even before that, the models are already useful for automatic transformation into prototypes. All major web engineering solutions provide tools for prototype generation [Koch06Transformations] [Ceri03WebML] [Gomez01Conceptual].

With a simple approach, this automated tool support can have a significant positive impact on the usability of the generated web application: The tools just need to provide "usability by default", i.e. default templates and transformation rules which result in usable interfaces. As already outlined in section 2.3.1, existing tools such as HTML editors or content management systems offer default templates which have been designed with care to conform to current technical standards and established user interface conventions.

The WebComposition system [Gellersen97WebComposition] applies this approach to the entire web engineering development process. It promotes building web applications out of numerous components which can exist at different hierarchy levels, from entire pages or sites down to small controls on a web page. These components can be created and evaluated by usability experts first, and later be reused for new projects with a certain confidence in their quality. [Centeno06WebCompWCAG] investigates the effect of composition on the WCAG accessibility of the final web page, and provides rules to make sure that the complete user interface is at least as accessible as its parts. With the WebComposition Service Linking System (WSLS, [Gaedke05WSLS] [Nussbaumer08WSLS]), the WebComposition concept is improved further. In an aspect-oriented approach, it ensures that the different aspects content, navigation, presentation, dialogue, process and communication are dealt with adequately during development. Starting from a basic architecture, the system can be gradually refined. During each refinement phase, components evolve further until they are mature enough to be used in the final version of the application. Again, the usability and accessibility of the user interface are ensured by reusing components that are known to be of high quality. In [Nussbaumer08WSLS, chapter 6], a distinction is made between accessibility rules which can be automated in an objective way and rules which can be automated subjectively, i.e. while automation is possible, subjective judgement is also necessary. A third group of rules requires manual checking, for example the rule that clear navigation mechanisms should be provided. For the first two groups, XSLT programs are supplied for a number of tests, such as checking whether the `alt` attribute of an image element is present in the HTML source code. The tests can be parametrized, e.g. with regard to what WCAG conformance level is required (A, AA or AAA). For the task of providing adequate navigation possibilities, the WSLS approach includes navigation modelling with conceptual, logical and physical phases. In the conceptual phase, the fundamental domains and navigation patterns are specified. During the logical navigation design, a mapping of these concepts to sources of information takes place. Finally, the physical design involves the use of WSLS navigation components to build the overall navigation. Thus, the component-based creation of the user interface is also extended to areas for which usable defaults are more difficult to achieve.

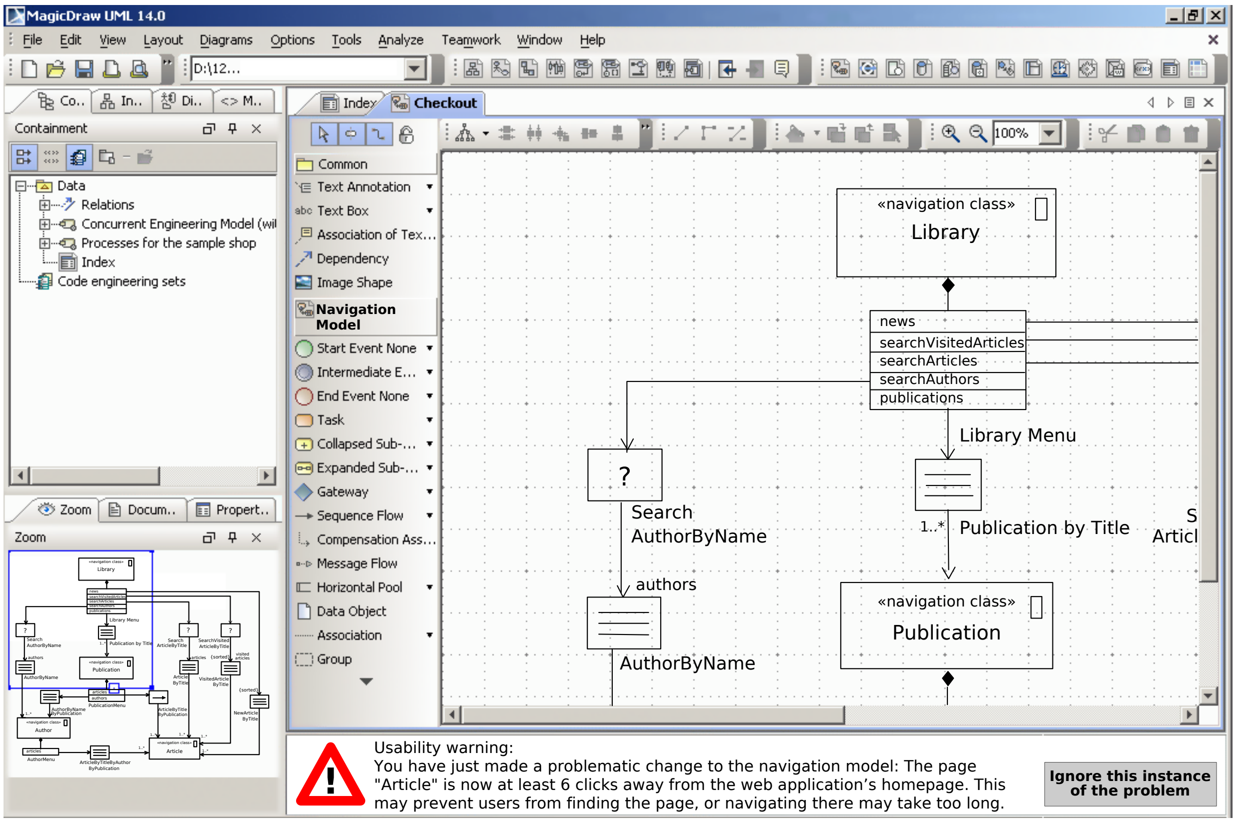
When applying the "usable defaults" approach to MDA tools, the tools can be supplied with default model transformation rulesets which ensure that common usability guidelines are adhered to. In fact, it is likely that the approach works better with model-based tools than e.g. with simple editors because then the tools have access to a more abstract view of the application.

When looking at existing web engineering tools such as VisualWADE (part of the OO-H web engineering solution), an aim of providing usable defaults for the implementation is noticeable. At the same time, further improvements are possible. In general, model transformation rules can use a variety of means to create results with good usability. A few examples are given below:

- For patterns like “guided tour”, the transformation tool can provide an evaluated implementation which represents the abstract concept with the most commonly used set of user interface elements. As mentioned before, in the case of “guided tour”, the user interface would consist of a number of pages which are connected with “previous” and “next” buttons.
- Support for patterns does not only extend to more concrete models, resulting in the arrangement of buttons and other controls on a page by the transformation tool, but also to more abstract ones, such as the navigation model. For example, online commerce almost always includes standard concepts like the shopping cart and a fixed set of pages (view/edit shopping cart, enter payment information). A tool can create an appropriate navigation model from a very general model (e.g. a use case) which only mentions that the web application will include electronic commerce elements. The generated model can then enforce certain rules, such as the rule that the page for entering payment information should only be displayed at the very end of a purchase, after the visitor has filled his shopping cart and indicated that he would like to buy the items it contains.
- Many web applications provide access to a database. The code generation tool can provide ways to browse common data structures like tables (e.g. sorting according to different columns, searching) and trees (traversing nodes, searching, collapsing/expanding subtrees). The developer is then free to modify the default behaviour in case additional or different types of navigation are required. Web engineeringsolutions with a background in databases, e.g. WebML, already feature this kind of support.
- The tool can provide a default for the visual layout of the web application’s pages, such as the common three-column layout. However, current web engineering models may not be sufficient to allow the tool to pick the most appropriate layout from a set of “standard” layouts: The models do not typically include information on the target user group (e.g. age), the overall type of the website (e.g. educational, informational, entertainment) etc., but this information would be useful when picking the default layout. Chapter 5 introduces a model which specifies properties of the user and the context of use.

4.2.3 Automatic Usability Analysis of Models

The later an error needs to be fixed during development, the greater the required effort becomes. Apart from prototyping, one way of improving website usability in the earlier development stages is to build expert knowledge into the modelling tool, and then to perform an automated usability



Usability warning:
You have just made a problematic change to the navigation model: The page "Article" is now at least 6 clicks away from the web application's homepage. This may prevent users from finding the page, or navigating there may take too long.

modified screenshot of MagicDraw UML, `http://www.magicdraw.com`, accessed 2 October 2007

Figure 4.2: Sketch of the interface of a web IDE with built-in automated model-based usability validation, showing a warning about a potential usability problem in the models.

and accessibility analysis in the background whenever the developer makes changes to the models. The advantage of this approach is that as long as the developer makes no usability-related mistakes, he will not be bothered by the system in any way. However, whenever he is in the process of making a problematic change, the IDE can immediately inform him of the risks involved, e.g. by displaying a warning message in the model editor. Because the model-based validation is built into the development environment, no additional work is required from the developer's side, i.e. there is no additional manual action to start the validation process which the developer could forget about.

The prototype of a model-based usability validator is presented and described in detail in chapter 6. Figure 4.2 shows a possible way the validator could be integrated in a modelling IDE: While the user is editing the model, the tool runs tests in the background at regular intervals. Any problematic changes to the model immediately result in a message being displayed at the bottom of the screen. The developer has the option of ignoring the message altogether, marking this instance as "not a problem in practice", or changing the model to make the message disappear.

Different model-based usability tests can be distinguished depending on the involved models:

- With more abstract, higher-level models, fundamental inconsistencies in the design can be detected. For instance, a use case diagram may show different types of users that participate in the use case, but the web engineering navigation model may show that additional user groups have access to the respective functionality. (This may not actually be a problem, and may be intended by the developer. A mechanism is needed to prevent the tool from reporting false positives more than once.)

- More concrete models include the presentation model, which web engineering solutions sometimes even regard as part of the implementation (see section 3.2.2). An example of a usability test for this model is a layout check: Does the visual layout of the pages conform to what the average user is expected to see on websites? The de-facto standard layout with three columns and and a header area will pass the test as long as the model assigns the right type of content to the different areas (e.g. main content in the middle column). On the other hand, some designs would be regarded as problematic, such as a layout in which the main navigation is centered at the top of the page – a spot which could be ignored by users because it tends to contain advertising [NielsenAlertbox, top 10 mistakes, nr. 7]

- Finally, usability analysis can be performed on the lowest-level "model" that is created with an MDA approach: The implementation. Validation tools which are solely based on an implementation already exist (section 2.3.4) – however, the quality of the evaluation can be increased significantly if the implementation is inspected *together with higher-level models*. Section 4.3.1 contains a discussion and further examples for this type of test.

For most of the usability tests mentioned above, the quality of the analysis can be improved if the tool has access to information about the users of the final system (age, background etc.) and the context of use (device used, speed of Internet access etc.). The developer can define these properties in a special model, see section 5.

When automated usability analysis is built into a web development IDE, it has the potential to keep the developer's workflow efficient, but at the same time to reduce the number of possible errors in the design. Due to the short time between a change and its being flagged as problematic, no mental context switch is necessary for the developer – he will be aware of why he is introducing the change, so he can immediately decide whether an alternative solution is possible. In contrast to this, if a usability problem is discovered e.g. during a prototype test, the developer will have to go back to the model, identify the aspect which is responsible for the usability problem, re-analyse the model and other sources (e.g. a requirements document) to understand the reasons for why the model was created in this particular way, and finally perform the necessary change.

Of course, the problems detected by automatic usability analysis of models are not comparable to the results of a usability test. In fact, user tests with real test persons are probably a more effective means of identifying usability problems than the static analysis of models: Automatic validation only has access to fairly high-level modelling information, whereas test users are presented with a user interface which is the result of further work by the developer to turn the models into something concrete. Moreover, the algorithms that can be built into an automatic analysis system have to make certain assumptions about the users (e.g. average cognitive skills at the age that is specified by the user model). In contrast to this, actual human users are available during a user test. Often, this includes persons who will be among the users of the final website, who have knowledge about the domain being addressed by the web application, an expectation about the workflows that should be supported, experience with the use of a predecessor system or related systems, etc.

As a result, automatic usability analysis of models can only detect some of the problems that a usability test would uncover. This means that the developer should not blindly trust such a tool during development, but instead regard it as an additional line of defence against the introduction of usability-related mistakes in the design. Due to its potential for cost savings, it should be employed together with user testing. For example, use of the tool may reduce the development costs because fewer development iterations (involving prototype building, real-life usability tests etc.) are necessary, as some errors are detected earlier.

Similar to the existent accessibility validators from section 2.3.4, proper handling of false positives is necessary with this type of tool to make it useful: Warnings about usability problems which are not actually present in the design, but which are nevertheless interpreted as problems by the tool, must not be too frequent. Two different strategies can be used together to achieve this. On one hand, careful design of the algorithms and heuristics which check for usability problems leads to fewer problems being reported (at the risk of an increase in false negatives). On the other hand, a mechanism should be present in the tool to mark specific instances of a reported problem as false positives, which will suppress future warnings about them.

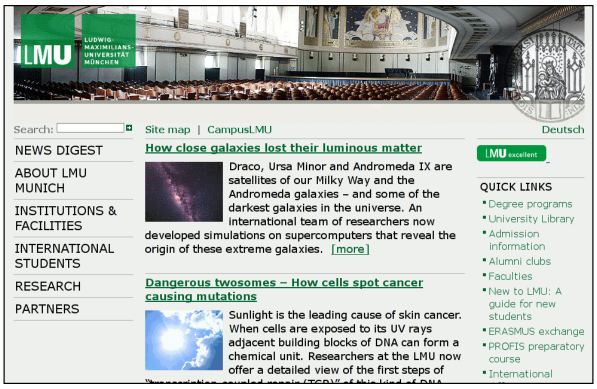
4.3 Implementation Phase

Summary: Implementation details can have a significant impact on usability. Automated, model-based usability checks can help to uncover the violation of usability guidelines. With an appropriately designed UI, a simple form of the the necessary modelling support can even be added to a web CMS for inexperienced users.

When the final implementation of a web application is created, most concepts of the user interface, including the site's navigation structure and the representation of data on screen, have been determined and specified in the different types of models. The developer will often concentrate on technical details which are not relevant for the usability of the application, such as scripting languages, the server-side software architecture and integration/installation of the application [Dumke03WE, chapter 6].

However, in fact the effect of the implementation phase on the usability of the final website is probably at least as strong as other development phases. The reason is that only at this point, the concrete properties of the pages are determined:

- The final page design is created. Predominantly, the presentation models of existing web engineering solutions only specify *what* appears on a page, not exactly *how* it is presented, what the text of the navigation links is, etc.


LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Search:

Site map | CampusLMU

Deutsch

NEWS DIGEST

ABOUT LMU MUNICH

INSTITUTIONS & FACILITIES

INTERNATIONAL STUDENTS

RESEARCH

PARTNERS

How close galaxies lost their luminous matter

Draco, Ursa Minor and Andromeda IX are satellites of our Milky Way and the Andromeda galaxies – and some of the darkest galaxies in the universe. An international team of researchers now developed simulations on supercomputers that reveal the origin of these extreme galaxies. [more]

Dangerous twosomes – How cells spot cancer causing mutations

Sunlight is the leading cause of skin cancer. When cells are exposed to its UV rays adjacent building blocks of DNA can form a chemical unit. Researchers at the LMU now offer a detailed view of the first steps of

LMU excellent

QUICK LINKS

- Degree programs
- University Library
- Admission information
- Alumni clubs
- Faculties
- New to LMU: A guide for new students
- ERASMUS exchange
- PROFIS preparatory course
- International

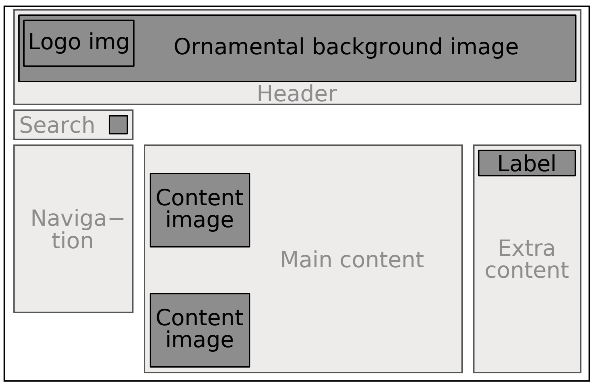


Figure 4.3: With access both to the implementation (left) and a model (right), automated usability validation can be improved. For example, ornamental images can be identified; their alternative text should be empty.

- The content of the pages is written. For instance, the style of a written piece of text has an influence on how well it is understood by users. Additionally, things like the graphical design influence the users' first impression and acceptance of the service.

In the following, two approaches for improving usability during the implementation phase are outlined: Model-based usability analysis of the website implementation, and tool support for less experienced web developers, which lets them switch easily between creating certain kinds of models and specifying the concrete instances of the models.

4.3.1 Automatic, Continual, Model-Based Usability Evaluation

Section 4.2.3 has already introduced the idea of using models as the basis for automated usability and accessibility analysis of various aspects of the web application. A promising extension of this concept is to include the implementation into the analysis as well. In general, this allows a tool to compare the abstract description of the web application and its behaviour (in the models) with the concrete realization, and to report any inconsistencies. Furthermore, because the tool has more knowledge about the concrete implementation, new heuristics for usability checks can be developed. Existing tools which only look at the final HTML code of a website are unable to implement such heuristics, as the abstract information cannot be deduced. A typical example is the accessibility guideline: *Images should have alternative text, except when they are ornamental* [Lynch02WebStyleGuide, chapter 7]. From the implementation alone, it is nearly impossible for a program to determine reliably whether an image is ornamental or not.

Depending on the type of model information that is consulted, a variety of usability checks are imaginable. Below, one example is given for different types of information. A more extensive list of possible tests can be found in chapter 6.

- From information in the user model (such as age), the right amount of text (number of words or sentences) can be estimated. For instance, there should be less text on a page that is intended for children.

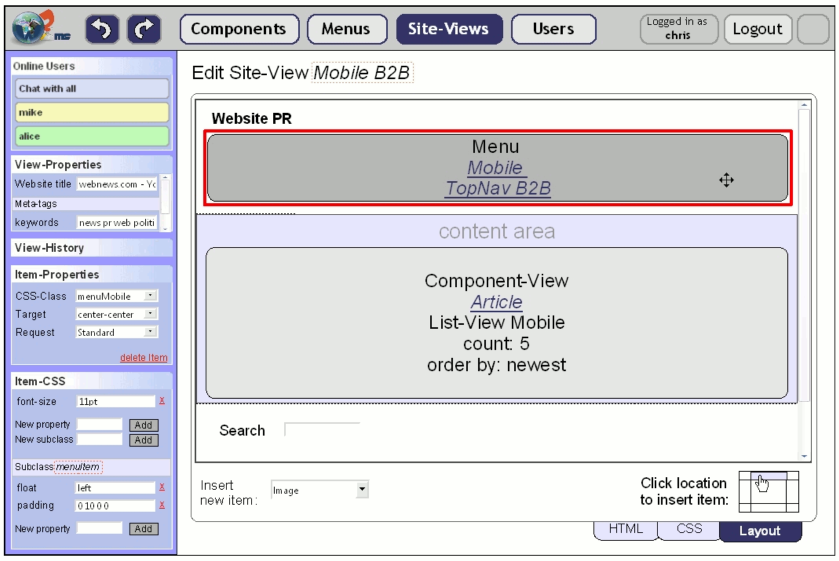
- The context model specifies many properties of the device, such as its screen size. A validator can check whether the pages will render fine at the given resolution, or if e.g. the need for horizontal scrolling would negatively affect the user experience.
- Some non-functional requirements, expressed for example using OCL constraints, can be tested. This includes the performance of pages, e.g. the time before a search result page is displayed to the users (see section 5.2.6).
- From the description of the environment of use, which is also given in the context model, certain desirable properties can be deduced. For example, if the environment is stressful for the users, they will not be interested in reading pages with large amounts of text, and will prefer pages which require a minimal amount of interaction to perform goals. Also, they may be more susceptible to having to wait for the web server, so the relevant warning threshold should be lowered.
- Using the navigation model together with the implementation, a program can find out whether important pages are reachable via prominently placed links and with an acceptable number of clicks. (Pages could be judged "important" based on a number of heuristics, such as whether they were visited during a previous development iteration's user test; see section 4.4.1.)
- The presentation model specifies the meaning of the different parts of the page layout. From this, the validator can deduce whether an image on the page is part of the content (so the page author should specify `alt` text for it) or whether it is ornamental only (so the `alt` text can be empty). Figure 4.3 shows the information contained in a presentation model on the right – most of the images (dark grey) have a function, but the background image in the header does not.

The model-based usability evaluation both of models and of the implementation are implemented in the prototype which is described in chapter 6. One possible way of adding this tool support to an integrated development environment has already been outlined in figure 4.2.

4.3.2 User-Friendly "Implicit Modelling Support" for Content Management Systems

Other sections in this chapter mostly assume a setting where the web application is developed by a person who is well-acquainted with this task, and who has some knowledge about the technologies that are involved, e.g. HTTP and HTML, JavaScript, server-side solutions like J2EE or relational databases. Once the infrastructure for creating and maintaining pages is present, the website is handed over to site maintainers whose technical expertise may not be as advanced, but who are able to supply the content. The developer is only consulted again whenever the maintainers have wishes which require his help.

However, today a significant portion of smaller websites is created by a single person who is keen on providing public access to information on a certain topic, but who is not well-acquainted with (or even interested in) the technical details. A content management system enables such



Edit Site-View *Mobile B2B*

Website title: webnews.com - Yc
Meta-tags
keywords: news pr web politi

CSS-Class: menuMobile
Target: center-center
Request: Standard

font-size: 11pt
New property
New subclass
Subclass *menuItem*
float: left
padding: 0 10 0 0
New property

Website PR

Menu
Mobile
TopNav B2B

content area

Component-View
Article
List-View Mobile
count: 5
order by: newest

Search

Insert new item: Image

Click location to insert item:

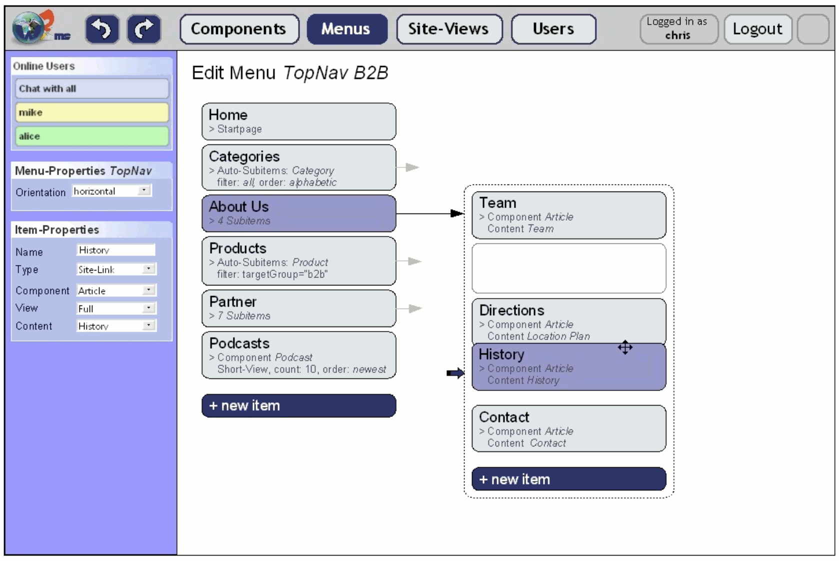
Figure 4.4: Simple modelling support in a prototype web CMS: Creation of a presentation model. [Metz06CMS, appendix B]

persons to perform all necessary steps of the design and implementation without in-depth knowledge of the server implementation. Since the functionality and ease of use of CMS has been increasing considerably over recent years, this type of "very inexperienced web developer" will probably become more common, so the following discussion presents some ideas for respective tool support.

Using CMS Meta Information for Implicit Generation of Models

The tool concepts which are proposed in other parts of this chapter often build upon the fact that models of the web application are available to the tools. In order to apply the same tool concepts to web authoring systems like content management systems, it would be necessary to add modelling support. Simpler systems (blogs, Wikis) hardly allow the site maintainer to change the predefined page format and navigation style, so in many cases, good usability can be achieved just by providing appropriate defaults. A CMS is more interesting: Because the maintainer has much more influence on the layout and navigation, modelling support would be more useful as the basis for automated usability evaluation and similar purposes.

At first sight, it might sound unrealistic to propose model editing support in a CMS which is intended for use by inexperienced web developers – these users cannot be expected to learn and understand the syntax and semantics of e.g. a web engineering navigation model. However, when taking a closer look at existing CMS solutions, several aspects of their functionality could already be interpreted as modelling support if the term "modelling" is loosely regarded as "specifying meta information about the website or its content". Often, the user-supplied information is sufficient to generate simple web engineering models.



Menu-Properties *TopNav*
Orientation horizontal

Item-Properties
Name History
Type Site-Link
Component Article
View Full
Content History

Edit Menu *TopNav B2B*

Home
> Startpage

Categories
> Auto-Subitems: *Category*
filter: *all*, order: *alphabetic*

About Us
> *4 Subitems*

Products
> Auto-Subitems: *Product*
filter: targetGroup="b2b"

Partner
> *7 Subitems*

Podcasts
> Component *Podcast*
Short-View, count: 10, order: *newest*

+ new item

Team
> Component *Article*
Content *Team*

Directions
> Component *Article*
Content *Location Plan*

History
> Component *Article*
Content *History*

Contact
> Component *Article*
Content *Contact*

+ new item

Figure 4.5: The menu editor lets the user specify the site structure. This information can be used to generate a menu, but also for automated usability checks. [Metz06CMS, app. B]

For instance, in a CMS the user creates a page layout and then defines the type of information that is displayed in each area of the page (main content, navigation etc.). The result could be regarded as a simple presentation model (figure 4.4). In a similar way, the CMS's menu editor can be viewed as an editor for an abstract specification of the site structure (figure 4.5). Another example is a feature of many content management systems which are intended for the publishing of news: Published articles can be assigned to one or more content categories (such as "software" or "hardware" on a site with computer-related news). This also creates a simple model describing the pieces of content on the web page: Categories and possibly sub-categories correspond to classes of a conceptual model, the individual articles to the instances of these classes (figure 4.6).

These examples have one thing in common: The users are not introduced to the concept of a model and there is no additional view where a diagram representation of the model needs to be edited. Instead, the "models" are created implicitly by settings that the users make while creating the website and entering content. Thus, advantage is taken of the fact that even inexperienced users are able to think in a more abstract way, while at the same time not requiring them to learn a new diagram language to specify their abstract thoughts.

User Interface Prototype for Modelling Support in a CMS

It is possible to implement model-based accessibility and usability evaluation (sections 4.2.3 and 4.3.1) for a web CMS rather than a web engineering IDE. When thinking about doing so, a similar approach should probably be taken as with the examples above. The users should be able to create and edit meta information quickly and in a way which is directly linked to the concrete instance that it describes, rather than in a separate model. Instead of diagrams, a normal web

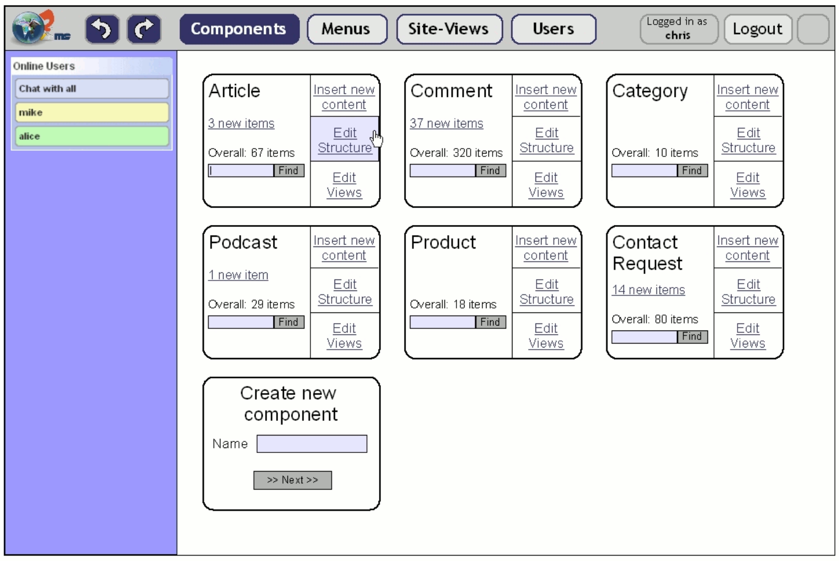


Figure 4.6: A CMS can also help with basic "conceptual modelling", here by specifying the different types of content that should appear on the website. [Metz06CMS, appendix B]

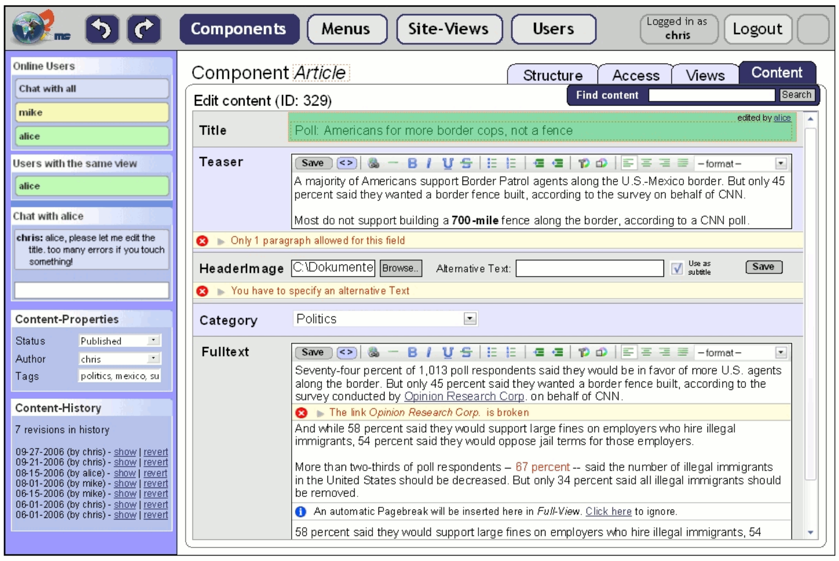
page GUI e.g. with HTML tables and buttons is preferable, as the users are already acquainted with the respective UI concepts.

It is a challenge to create a web CMS user interface which provides easily usable and intuitive ways for its users to specify the abstract information, notify them of any usability problems and provide hints on how to solve these problems. Because the users' knowledge about the topic of web usability varies even more than that of web developers, the UI must be suitable for absolute beginners (who may have to be guided using additional documentation or a tutorial) as well as experts (who will prefer controls which allow them to reach their goals quickly).

As part of a student project and diploma thesis on user interface concepts for content management systems [Metz06CMS], the aspect of creating easily usable controls was examined: How can quick operation of the different features be achieved while at the same time providing appropriate feedback? With the help of an AJAX-based prototype, a number of user interface concepts were implemented and evaluated:

Figure 4.4 shows a layout editor which offers a simple point-and-click interface for assigning different types of content to the different parts of the page. As mentioned above, this presentation model could subsequently be used for automated checks, such as verifying that the layout conforms to the de-facto standard on the web (navigation at the left-hand side of the screen, content in the middle, etc.).

The menu editor in figure 4.5 already bears some resemblance to a modelling tool, except that its editing functionality is limited to very simple, tree-shaped, directed navigation graphs. While these graphs are not sufficient to describe the intended navigation patterns of site visitors like a real web engineering solution's model, they are still useful for automated usability tool support –



Online Users
Chat with all
mike
alice
Users with the same view
alice
Chat with alice
chris: alice, please let me edit the title. too many errors if you touch something!

Content-Properties

Status	Published
Author	chris
Tags	politics, mexico, su

Content-History
7 revisions in history
09-27-2006 (by chris) - show | revert
09-21-2006 (by chris) - show | revert
08-15-2006 (by alice) - show | revert
08-01-2006 (by mike) - show | revert
06-15-2006 (by mike) - show | revert
06-01-2006 (by chris) - show | revert
06-01-2006 (by chris) - show | revert

Component *Article*
Edit content (ID: 329)

Title: Poll: Americans for more border cops, not a fence (edited by alice)

Teaser:
A majority of Americans support Border Patrol agents along the U.S.-Mexico border. But only 45 percent said they wanted a border fence built, according to the survey on behalf of CNN.

Most do not support building a **700-mile** fence along the border, according to a CNN poll.

Only 1 paragraph allowed for this field

HeaderImage: C:\Dokumente Alternative Text: [✓ Use as subtitle]

You have to specify an alternative Text

Category: Politics

Fulltext:
Seventy-four percent of 1,013 poll respondents said they would be in favor of more U.S. agents along the border. But only 45 percent said they wanted a border fence built, according to the survey conducted by Opinion Research Corp. on behalf of CNN.

The link *Opinion Research Corp.* is broken

And while 58 percent said they would support large fines on employers who hire illegal immigrants, 54 percent said they would oppose jail terms for those employers.

More than two-thirds of poll respondents – 67 percent -- said the number of illegal immigrants in the United States should be decreased. But only 34 percent said all illegal immigrants should be removed.

An automatic Pagebreak will be inserted here in *Full-View*. Click here to ignore.

58 percent said they would support large fines on employers who hire illegal immigrants, 54

Figure 4.7: The prototype integrates design, implementation and testing in a single user interface. [Metz06CMS, appendix B]

for example, similar to the check for a de-facto default page layout above, a check for commonly expected menu entries such as “About this site” or “Contact” could be implemented.

Figure 4.6 shows the prototype’s approach (called “components”) to make users supply information about the content they add to pages: Before the content itself can be entered, different types of content must be defined (e.g. “article”, “comment”). The presentational aspects of each type of content (placement on the page, CSS style elements) are specified alongside the definition. Because of this fact, the users will not object to the additional work of creating component types – the reason behind the abstraction and its utility are obvious. At the same time, the CMS is given valuable information which supplements the implicit presentation model. Furthermore, the components and their instances are a basic conceptual model, as they describe the different entities the website deals with. While the current prototype does not offer support for inheritance (i.e. basing one component class on another), such a feature would make sense for a more advanced version of the program.

Finally, figure 4.7 illustrates how implementation work, specifying abstract properties and model-based usability checks could be integrated: When the users add new content to the site, they are asked to provide further metadata about the content. In the screenshot, only a category must be assigned, but other properties such as the intended audience are imaginable. While the content is entered, warning messages are displayed about any accessibility or usability problems – here, an automatic check has revealed that a URL entered by the user is not valid. One effect of this approach is that it is very easy for users to switch between “modelling” (i.e., providing meta information), implementation (entering text) and testing (feedback from a usability validator) – a desirable feature because less experienced users may not be interested in rigorously following a predefined development method with separated design and implementation phases.

Evaluation

In a small evaluation of the concepts [Metz06CMS, section 4.2.4], five test users (four users with a technical background, one without any interest in the technology) were asked to create the structure of a website with the prototype system in a series of tasks. They were asked to think aloud, i.e. to comment on what they were doing and any problems they were having. In general, even inexperienced users were able to use the system. However, the non-technical user initially had problems understanding the abstract concept of "components" and did not see the necessity of defining types of content before entering the content itself. Further improvements to the UI may be necessary to address this – for example, the solution may be as simple as providing a few common default components, so the user does not have to define his own as the first step when using the system. The various other features of the prototype UI (immediate feedback due to AJAX interface, edit-in-place of data, concepts for simultaneous use by several users) were used by the test participants without major problems. All of them were able to complete the assigned tasks.

4.4 Testing Phase

Summary: Previous sections have described how fully automatic testing can be made part of earlier development phases. For usability tests with real test users, tool support cannot eliminate the need for test subjects to participate, but it can reduce the amount of work required by the developer to prepare, conduct and analyse a test.

Once a web application has been implemented, different kinds of tests are necessary to ensure that it works as expected. This includes checking the criteria of the service level agreement, a test of the final hardware and software, determining whether the assigned hardware, software and human resources lead to adequate performance, and planning for future changes, e.g. by setting aside a budget [Dumke03WE, chapter 7]. With regard to usability, established procedures include analysis of the finished web application by experts (cognitive walkthrough, heuristic evaluation) and tests with test users or the final users, in a lab setting or in the field (experimental evaluation, observation with a think-aloud protocol, interviews) [Dix04HCI, chapter 9], [Constantine99Software, chapter 16]. Especially with web applications, a beta phase is a popular means to achieve wider testing of the application. It allows the end users to access the application, possibly even already in a productive environment.

Some aspects of testing have already been covered in previous sections: By building expert knowledge about usability guidelines into a tool, some tests can be performed continuously during design and implementation of an application. If any problematic changes are discovered, the developer can be warned about them the moment he makes the change. However, this automated usability evaluation alone is by no means enough to ensure that the application has good usability. It needs to be complemented by the opinions of experts and the results from user tests.

It is difficult to come up with tool support for this phase of development: Humans are involved in the testing procedure in most cases, either as experts or end users, and a lot of human judgment is required to prepare the web application's user interface, design a test setup, perform the user test, and finally to identify and categorize the problems that are found. However, a closer examination of the procedures and tasks that are actually carried out reveals that a lot of time is spent e.g. organizing rooms or test persons, setting up the testing environment and any equipment used to record user actions, and to analyse the data that is recorded during the test. Tools can help with these simpler aspects of the task. One possible approach is described below, it has the potential to make setup easier while at the same time producing detailed logs.

4.4.1 Web-Based Interaction Logging For User Tests

Conducting user tests in the established way is a fairly work-intensive task: Test users must be recruited and invited, a lab must be booked, and the test environment needs to be set up. After the test, the analysis e.g. of recorded video footage can be time-consuming. This section proposes an alternative, "lightweight" usability test which is supported by a special tool. It aims at reducing the cost of user testing (both in terms of time and monetary cost). Compared to the classical user test in a laboratory, less data is typically collected during each user test (e.g. no video recordings), but the data that is collected is more suitable for automatic analysis, and the more automated nature means that the number of test users can be increased considerably without a lot of additional work.

The tool concept is as follows: The test user's actions in the web browser are recorded in detail. This includes all interaction made via the mouse or keyboard, and it is suitable for all aspects of web-based interaction, including advanced AJAX-based concepts like drag & drop. For each type of interaction, the exact web page element (e.g. a button) is identified automatically and written to a log file. After the test, the logged information can be used to analyse e.g. the time taken to complete a task. The implementation of the approach (presented in chapter 7) has a number of interesting properties:

- No installation of software is required on the client side. Because an HTTP proxy is used to intercept the page requests and (through an AJAX-based technique) to record user actions like mouse movement, the only change required on the client's side is a reconfiguration of the browser's proxy setting.
- As a result of the proxy architecture, no changes are necessary to the server either – the web application that needs to be tested does not have to be changed for the test. Figure 4.8 illustrates how the proxy is inserted between the client and server.
- Non-invasive operation: Use of the logging system does not alter the user experience.
- The solution works with with normal web pages and non-AJAX web applications, but is particularly useful for AJAX applications, which are not supported well by comparable alternative logging solutions.
- As the solution relies on established web programming technologies, it is compatible with a variety of browsers and operating systems.

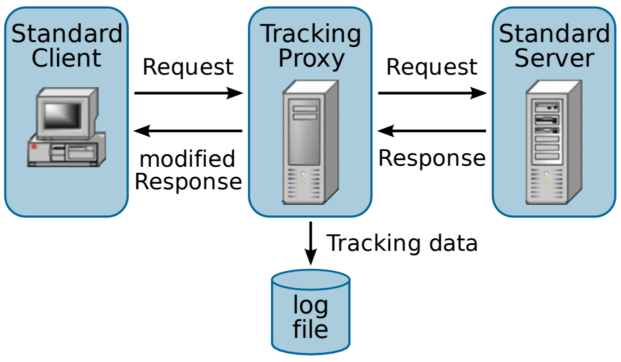


Figure 4.8: Architecture of a system for web-based logging of user interaction with web pages, based on a filtering HTTP proxy. [Atterer06UsaProxy] and chapter 7

- Detailed logging: Apart from the URLs that are requested, the interaction within a page can be recorded. For each type of interaction (mouse movement, clicks, key presses, scrolling, drop-down menu selection, selecting text in input fields), not only the directly available information is logged (pixel coordinates of the mouse pointer, key that was pressed etc.), but also the HTML element that the user interacted with, as identified by its ID in the document and its position in the DOM (document object model) tree.

The above ideas have been implemented in the form of "UsaProxy", the working prototype of an HTTP proxy for interaction logging. Its description in chapter 7 also includes a discussion of related work, problems and challenges of the implementation, as well as an evaluation.

The tool concept is suitable as an interaction logging facility for a lab-based usability test. When used for this purpose, it can collect data about the user actions which is more accurate than e.g. a screen capture program or a tool which records the mouse coordinates, because the exact links, buttons, input fields etc. on HTML pages are directly identified: Rather than a log entry like *"The mouse was clicked at screen coordinates 243/943"*, there is a log entry

> *"The mouse was clicked at coordinates 203/2034 on the page, on the link with* `id="moreinfo"`*, labelled 'More information', located in the DOM tree at position* `abadaacbb`*".*

(See chapter 7 for the exact format and meaning of log entries.) Automated analysis of the interaction log becomes much easier with this type of entry. Typically, only a few lines of code written in a scripting language like Perl or Python are enough to interpret the timestamps and events in the log file, and to extract information such as the time taken for completion of a task.

However, the concept is also capable of supporting other types of usability tests. Above all, the lightweight, flexible architecture allows user tests to take place remotely – the test persons do not need to be in the same place as the usability expert who conducts the user test, or the system on which UsaProxy is installed. Having been guided through the simple process of reconfiguring their browser's proxy setting, they can participate in the test immediately. If the server on which the web application runs is under the control of the usability expert, remote usability testing is even possible without requiring the users to reconfigure their browser. A further possible advantage of remote tests with UsaProxy is that the users can access the web application in their usual surroundings, with their existing computer, operating system and browser.

On one hand, performing a user test remotely has the disadvantage that the usability expert cannot observe the user's personal reaction to the user interface directly. To some extent, this can be counteracted by asking the test subjects to establish a voice connection to the expert, either by voice over IP or using a regular telephone line, and to have them "think aloud" about their experience.

On the other hand, remote usability tests can be used to the tester's advantage: Especially with web applications, the prospective audience is often spread geographically, possibly even all over the world. If the test users had to travel to a specific lab for all user tests, these tests would become far too expensive. Even if the audience is not spread geographically, some experts recommend not to recruit test subjects for a company in the same city that is dominated by the company's industry [NielsenAlertbox, 30 April 2007]. Furthermore, if no voice feedback is needed, or if the think-aloud protocol is recorded rather than immediately listened to by the expert, test sessions can be fully automated, and can take place in parallel for many users. At the same time, the results of remote testing are comparable to laboratory tests in terms of the number, type and severity of discovered problems [Brush04Comparison].

Taking the idea of remote usability testing one step further, the tool concept also supports recruiting test subjects over the Internet: An existing, similar application (for example, the predecessor of the new web application) can inform users about the testing process on its pages and ask them to participate in a test, the usability expert can post an invitation on support forums related to the topic, a database of existing customers can be used to contact interested users, and even online advertising is imaginable as a means to find test subjects.

Depending on the policies of the company which develops the new web application, it may be a problem that the test users cannot be expected to remain silent about what they saw. Requiring them to sign an NDA (non-disclosure agreement) would quickly make the process much more cumbersome, and it would significantly reduce the number of volunteers. However, especially with rich Internet applications, it is common even for large companies to conduct a public beta phase. If there are concerns about negative reactions to the initial user interface, one possibility is to perform a more closed usability test first and only to start public testing with a later iteration of the UI.

In conclusion, the tool concept facilitates very accurate logging of user actions such as mouse clicks, because the exact clicked link or button is identified. Due to its flexibility, it allows remote user testing, or even fully automated user testing in the sense that no manual action by the usability expert is necessary for each test subject. Because the log is less accurate than e.g. what can be achieved with recording video in a lab, the tool concept allows a tradeoff between detailed testing with few users, and slightly less detailed testing with many users. It is advisable to combine these two techniques: Usability problems can be identified by looking at a few users' reactions in detail (e.g. even including gaze tracking), but some problems will only become apparent once a greater number of people use an application in a wider variety of contexts. Chapter 7 contains details on the implementation, a comparison with related work, and an evaluation of the concept.

4.5 Maintenance Phase

Summary: When the site is in productive use, regular model-based analysis is still useful, e.g. because usability may be affected by changes to the content. Embedding web engineering models in the implementation allows for easier access by tools. Also, user-contributed feedback is a valuable source of information about problems.

After the web application has been tested and deployed, it is handed over to the customer for productive use. If the previous project phases were successful, major changes to the application should not be necessary at this point. Nevertheless, the running system should not be simply left to its users: A few adjustments and minor changes may be advantageous to improve productive usage, and after a while, the software may also need to be adapted to meet changing requirements [Dumke03WE, chapter 8].

Compared to classical GUI-based software which is installed on desktop computers, web applications are more likely to need changes after a while: The underlying technology changes more rapidly (e.g. programming techniques, standards for content presentation), differences in newer browser versions or implementations can cause problems, or the application may need to be protected against newly discovered security problems. In the area of usability, a similar development is possible: First, new user interface concepts for web applications may slowly emerge, requiring slight changes to the user interface to avoid that the users get confused. For publically available applications, innovations by a competitor can be a cause for a UI redesign, for instance using an AJAX-based approach. Finally, one primary task of many web applications is to organize and visualise content. Over time, this content may be subject to changes which decrease the site's usability. A simple example for this effect is the case that many items are added to a list on a web page over a longer period of time. At the time the page is created, the list may be short enough to be parsed visually, but once it gets too long, it may be necessary to split it up into several pages, or even to provide a server-based search facility.

4.5.1 Model-Based Usability Evaluation of Implementation and Usage Statistics

The concept which has already been introduced for the design and implementation phases (sections 4.2.3 and 4.3.1) can also be extended to the maintenance phase: At regular intervals, an automated accessibility and usability evaluation is performed, and a notification is sent whenever there any new issues which might affect the users' ability to work with the web application. As outlined above, the usual case should be that no major problems with the website become obvious, as these should have been identified and addressed in earlier development stages. However, problems could be triggered by changing requirements, new technologies or changes made to the site content.

Compared to the other ways of performing automated analysis, this type of automated check has an interesting advantage: It can draw on three different sources of information, which complement one another and provide a complete view of the web application and the way it is used:

- The models provide a more abstract description of the structure and behaviour.
- The implementation can be inspected for details about the HTML code, link structure etc.
- Logs are now available, allowing the tool to analyse how the final users work with the application.

With access to the usage logs, the tool concept has the potential to provide a more sophisticated alternative to the log analysis tools presented in section 2.3.3. The following is a list of example usability checks which become possible. It only includes checks which are neither possible using a model-based evaluation at an earlier stage, nor using log analysis alone. Roughly, they can be divided into comparisons of models with actual usage, creating warnings based on an analysis of log statistics, and re-evaluation of the content after changes have been made to it.

- If a constraint in the models or the requirements specification states that a purchase should be possible in no more than five minutes after entering a shopping site, the actual duration of the activity can be compared to this. This concept also extends to performance of the server or database setup – if a website becomes more popular after a while, the validator can alert the developer to the fact that the server resources may not be sufficient to sustain the increased amount of traffic, causing annoying delays for users.

- The models provide use cases, and these use cases have typically been made more concrete by specifying appropriate navigation paths in the navigation model, which in turn are represented by links on the HTML page. Reversing this series of steps, the usability validator can be envisioned to look at popular navigation paths of the site visitors and deduce a measure of how well actual and projected navigation paths coincide. Differences may reveal problems which could not be found with usability testing. For example, users might shy away from just trying out a link with a slightly misleading label because (in contrast to test persons) they fear that clicking on it would cost them money. The tool can immediately tell which use cases are affected by such problems.

- In general, models can define *thresholds* for a large number of criteria: Rather than requiring regular manual inspection of usage statistics to look for anything unusual, a tool can automatically output a warning once the threshold is crossed. For instance, if data about the popularity of navigation paths is accumulated over time, the tool can warn about sudden changes, e.g. if the “proceed to checkout” navigation path drops in popularity, or if the amount of visitors who reach the site via the Google search engine gets significantly lower.

- Section 4.3.1 has already mentioned the possibility of looking at the text content of a website, and using a heuristic to determine whether it would be understood by the audience as defined by the user model, e.g. children. It is important to continue this type of validation after the initial launch of the site: While the site content may be easy to understand at first, later changes may negatively affect this. Changes can also cause technical problems, such as rendering problems when the width of the page content no longer fits on the display of a supported device.

With this approach, one question that needs to be addressed is how the warnings of the usability validator should be delivered to someone who can react to them, and who that person should be. In contrast to earlier phases, the web developer will often have mostly passed responsibility for the site to someone who is more closely involved with its long-term operation, such as a person who maintains the content. For many of the warning messages, this person will be the right choice, as they can act quickly to resolve problems which were caused by recent changes to the site. Nevertheless, the person should also have enough technical background and knowledge to know when other warnings output by the validator, e.g. about a mismatch between models and observed user behaviour, are an indication of a major problem. Possible channels for delivering the warning messages include the content management system interface, email notification or a stand-alone web interface, similar to existing log analysis or web validation tools.

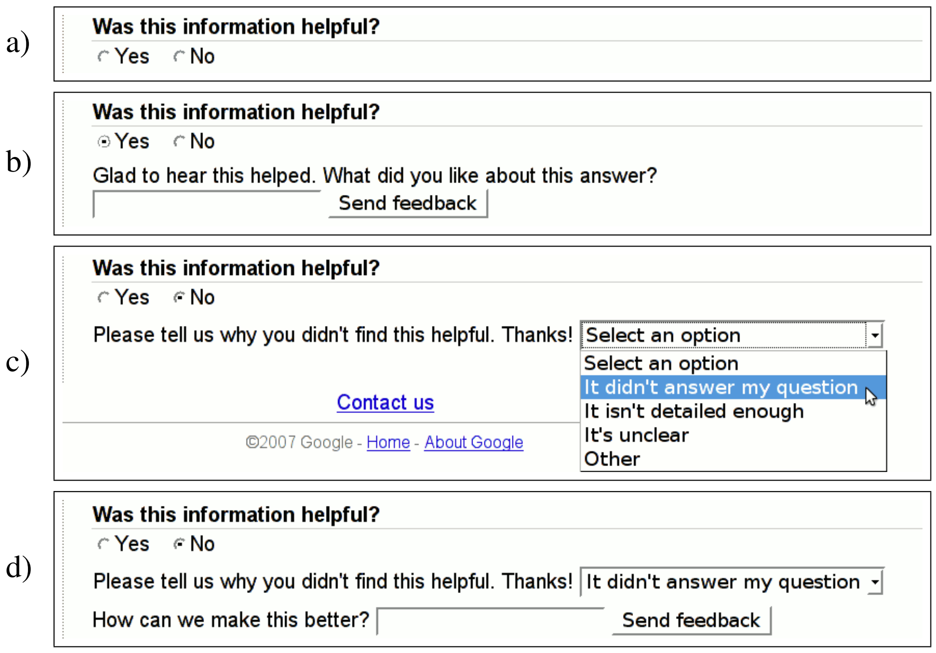
Embedding Model Information in the Implementation

In practice, performing model-based analysis during the maintenance phase also brings with it another problem: The models may have been readily available for processing in a web development IDE during earlier phases, but this is not necessarily the case at this point anymore. If a modelling solution is integrated into a content management system, some meta information about the website may be stored in the CMS's database, but the CMS-based solution outlined in section 4.3.2 is only expected to operate on fairly concrete information such as a presentation model, there is no provision to include e.g. use case models.

In addition to this, putting the model information into a particular CMS's database in an implementation-specific format is not desirable: Access to the models may not be possible from outside the CMS, e.g. using special-purpose validation software, and (unlike with existing accessibility validators) it may not be possible to perform model-based tests for external sites which are not under one's own control.

One promising solution to these problems is the use of standard file formats and technologies to describe and access the model data. Because web applications are the focus of this work and many tools in this area are also web-based, it seems appropriate to use web technologies for this purpose. Section 5.4 contains a discussion of the alternatives for specifying model information, including possible formats for their representation (e.g. XMI, RDF), the different ways of adding them to the website (embedded into the HTML document, referenced from it, specified using transformation rules), and ways to create references from the implementation to the appropriate models and vice versa. The standard HTTP protocol can be used to access these resources.

With this approach, model information remains available during the time when the final implementation is running, and it can even be provided publically to anonymous site visitors. This creates a powerful infrastructure for tool concepts which benefit from model information, and could even be regarded as a step towards the creation of the semantic web. However, the discussion of tools that are not related to usability analysis is beyond the scope of this thesis.


a)

Was this information helpful?

Yes No

b)

Was this information helpful?

[✓ Yes] No

Glad to hear this helped. What did you like about this answer?

Send feedback

c)

Was this information helpful?

Yes [✓ No]

Please tell us why you didn't find this helpful. Thanks! Select an option

Select an option
It didn't answer my question
It isn't detailed enough
It's unclear
Other

Contact us

©2007 Google - Home - About Google

d)

Was this information helpful?

Yes [✓ No]

Please tell us why you didn't find this helpful. Thanks! It didn't answer my question

How can we make this better?

Send feedback

Source: `http://maps.google.com/support/bin/answer.py?answer=68259`, accessed 24 October 2007

Figure 4.9: UI for automatic aggregation of submitted data: Users are asked to categorize their experience (a, c), only a few keywords of free text entry are possible (b, d).

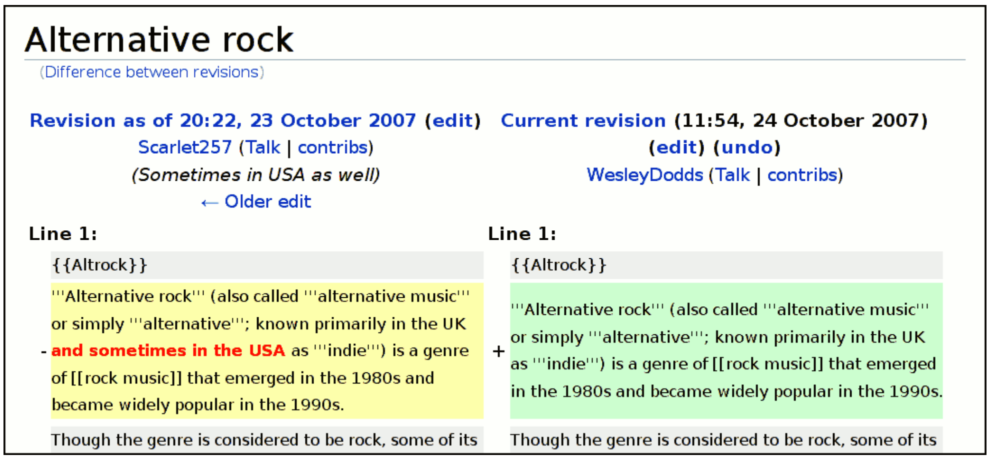
4.5.2 Support for User Feedback

When a website has been online for a while, it is possible that less attention is paid to usability than during development. The site will grow over time, and new information and features may be added without conducting rigorous usability testing. A re-evaluation of the site after some time is a promising way to check whether it still fulfils the requirements [Dumke03WE, section 8.5].

However, another reliable source about potential usability problems is available and should not be ignored: The users of the web application will be the first to notice problems. Only a small percentage will be inclined to provide feedback about problems to the operator of a website as long as they can still achieve their goals with it. However, these few users may be enough to help the site maintainer with the identification of problems. In general, the "Web 2.0" trend has resulted in more willingness of users to participate in the creation and maintenance of websites. With appropriate tool support, some types of websites can take advantage of this fact.

Aggregation of Data From Feedback Forms

Web-based forms are a simple way to allow the users to give feedback about problems they encounter. However, a look at many large-scale web applications (e.g. web-based mail accounts) reveals that forms are hardly used in practice for high-traffic sites. On the contrary, the way to reach support staff is usually well hidden, or even not present at all. The reason is obvious: The free-form text comments or email messages would be far too numerous to be processed by staff.



Alternative rock

(Difference between revisions)

Revision as of 20:22, 23 October 2007 (edit)
Scarlet257 (Talk | contribs)
(Sometimes in USA as well)
← Older edit

Current revision (11:54, 24 October 2007)
(edit) (undo)
WesleyDodds (Talk | contribs)

Line 1:

{{Altrock}}

- '''Alternative rock''' (also called '''alternative music''' or simply '''alternative'''; known primarily in the UK and sometimes in the USA as '''indie''') is a genre of [[rock music]] that emerged in the 1980s and became widely popular in the 1990s.

Though the genre is considered to be rock, some of its

Line 1:

{{Altrock}}

+ '''Alternative rock''' (also called '''alternative music''' or simply '''alternative'''; known primarily in the UK as '''indie''') is a genre of [[rock music]] that emerged in the 1980s and became widely popular in the 1990s.

Though the genre is considered to be rock, some of its

Source: Adapted from `http://en.wikipedia.org`, accessed 27 October 2007

Figure 4.10: When allowing user-supplied updates during the maintenance phase, visualisation tool support can help to monitor changes to the content.

This problem can be addressed by partially automating the analysis of data from feedback forms: The input is categorized according to different criteria, and only a summary is presented to the site maintainer. Similar to the analysis of keywords in search engine queries extracted from web server logs, aggregation is also possible for free-form text, by measuring the frequency of certain words or phrases, and also monitoring changes of the frequency over time. For instance, a sudden increase in the number of occurrences can alert the website maintainer to a newly introduced problem.

In figure 4.9, a possible user interface for such a feedback system is illustrated with the example of Google Maps (`http://maps.google.com/`): First, feedback is only possible from the online help section of the system, at the bottom of pages which provide FAQ answers. This potentially improves the quality of the feedback data, as only people who looked at the help section will be able to submit data. By asking a very simple “Was this information useful (y/n)?” question at first, users are not deterred by the prospect of filling out a lengthy form. After a click on “no”, a selection dialogue for a “problem category” is presented. From the “yes/no” answer and this category information, statistics can easily be generated. This is more difficult for the entry of free text, which is possible in a single-line field. By forcing the users to give a concise answer, analysis of word frequency may become easier because the ratio of keywords (e.g. “zoom” if there is a problem with the zoom functionality) to uninteresting words is higher than with longer texts. Furthermore, it seems inappropriate to let users enter detailed descriptions if these descriptions will hardly ever be read directly by a human.

User-Generated Content Updates

With the above automated analysis of user feedback, updates to the site content must be performed by the person or team which maintains the site. For some sites, this may be the only possible way, but for others, it may be acceptable to let the users participate in updating the site with new information, or to adapt it to changing requirements in other ways. The advent of sites with user-generated content such as Wikipedia (`http://wikipedia.org`) has shown that high-quality content can be created by letting site visitors participate in the maintenance process.

There is a tradeoff between the danger that the usability of the site is affected by outdated information, and the danger that incorrect or even malicious changes are performed by some users. The decision whether to allow updates by site users will often be motivated by the resources available for maintenance: If enough staff is available to perform updates, there is less need to let external visitors edit content. Consequently, small or “hobby” sites with little financial backing are more likely to allow user participation.

Tool support in this area needs to address two needs: First, facilities must be present for normal site visitors to contribute to the site. This is possible by using a CMS or Wiki where they can register, edit content, and where the write access rights to content can be adapted according to the needs of the site maintainer. Furthermore, users can be allowed to edit content directly, or their contributions can be prevented from being visible on the site until a moderator has approved them.

The second task of tool support is quality control: A system must be in place to monitor changes and allow a moderator to intervene in case of problems. Different aspects of this task can be supported by tools in different ways: For instance, a moderation system should provide an appropriate user interface to let the maintainer quickly view any changes made by a user, and to revert/reject/adapt them if they are not. Figure 4.10 shows an example which highlights added or deleted words. Furthermore, it makes sense to distinguish different kinds of users. Users who have a record of making useful contributions to the site need less attention. A system which automatically maintains confidence ratings for users can relieve the site maintainer of a lot of work. In particular, this is the case if a large number of people contribute, and if the system allows experienced, trusted users to monitor the changes of new users.

Above, only updates to the content of a website are discussed, rather than changes to its functionality. It is conceivable to extend the concept to updates to the business logic of the site. However, in contrast to the already publically readable content, allowing users to modify the code of the site requires the developer to permit public access to the previously non-public source code, which may not be desired. Moreover, much more careful review of user-supplied changes would be required, e.g. to avoid introducing security holes and to ensure that any private data entered by the site visitors is kept private. In practice, user-supplied updates to the code will probably only be feasible if the web application is an open-source application.

4.6 Conclusion: Tools Can Help to Identify Usability Problems More Easily

This chapter has looked at the different stages of the web engineering development process, exploring possibilities for tool support to improve the usability of the website that is being created.

Initially, it may seem unlikely that significant usability improvements are possible with special tool support, as programs can only ever have a very limited view of how humans might react to a particular user interface. However, as the discussion of tool concepts above shows, there is a potential to assist the developer with many usability-related aspects during his work on the application. Once some more manual parts of his work have been automated, he is not distracted by details and has more time to consider the possible implications of his design decisions on usability. As a result, many of the discussed tool concepts only indirectly improve usability, by giving the developer the means to identify usability problems more easily.

The basic ideas behind the different tool concepts can be summarized in the following five principles for usability tool support in model-based web development:

1. **Embed expert knowledge about usability in tools**, i.e. let the tools follow established guidelines by default when creating new applications, or let them analyse the developer's design with regard to these guidelines.
2. **Use models as a source of meta information** about the pages of the web application. This allows automated usability validators to achieve higher quality than without the presence of models.
3. **Do not eliminate humans from the development process** (e.g. for prototyping or user tests), but *do* automate preparation of the relevant steps in the process, as well as data gathering and analysis.
4. **Educate the developer** and increase his awareness that usability problems are important. It should not be assumed that he is an expert in usability, and not even that he attaches a lot of importance to usability issues.
5. **Leave the developer in control** – only assist him in coming to a conclusion about the impact of a design on usability, rather than making the decision for him.

Two of the most promising tool concepts are evaluated in chapters 6 and 7: Model-based usability validation (sections 4.2.3, 4.3.1, 4.5.1) and web-based interaction logging for user tests (section 4.4.1). Together, they incorporate all of the principles above. Many of the other concepts are less interesting for prototypical implementation, as they either employ already existing tools for new purpuses (such as requirements elicitation), or because it was felt that the functionality was similar to one of the two implemented concepts. For example, implicit modelling support for content management systems is conceptually similar to the "stand-alone" model-based usability validator that was implemented.

Chapter 5

A Model for Usability and Accessibility Information

As shown in the previous chapters, there is only limited support in existing web engineering models to record information which is needed by automated tools for usability and accessibility support. For example, none of the approaches from section 3.2 includes a user model which records the number of years of school education of the user – however, the "text complexity" usability check from section 4.3.1 needs this information to determine whether the page is likely to be understood by the users. In practice, many properties influence how users perceive and interact with the web application. A context model is required to specify this information, e.g. to assign different user groups to different parts of the application. The model must be integrated with the other web engineering models in order to allow an accurate mapping between parts of the application and the appropriate context.

After a look at related work, this chapter proposes extensions to existing web engineering models which allow the implementation of the tool concepts outlined in previous sections. This is achieved by introducing a new metamodel. The extensions represent a framework with defined semantics which can be used by different modelling solutions and tools. The meaning of its elements (such as string tags, see below) is described in a way which is sufficiently detailed to allow an implementation to be created. By providing a "standard" for the representation of usability-related information, it is hoped that future work on the integration of different tools will become easier.

The context model describes the user, platform and environment, and can be associated with existing classes which represent e.g. the pages of the application. The developer uses it to specify *expected static properties* of users etc. – the model is not intended for the description of parameters which change at runtime (although an extension to allow this would be feasible). In addition to the context model, a few properties are also directly added to existing presentation or navigation models, and yet others are attached to models by means of OCL constraints.

Finally, this chapter also discusses how to add the above model elements to the models of the different existing web engineering environments, and how to embed model information in the final HTML implementation.

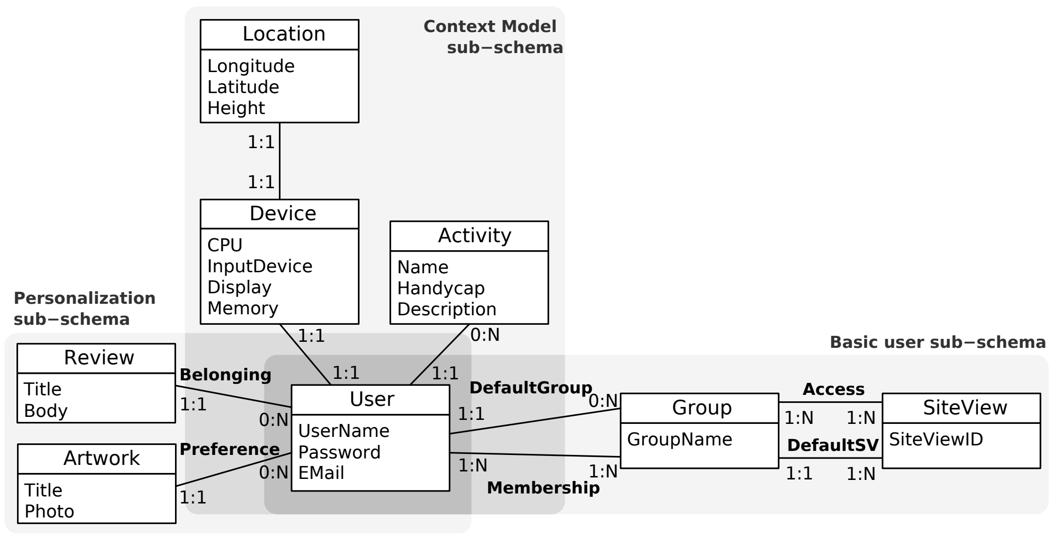


Figure 5.1: Example for a context model for WebML applications which adapt to changes of context. [Ceri07MDDContext]

5.1 Modelling of User Properties and Context

Summary: User models and context models are of particular interest for the tool concepts introduced in this thesis. Some existing web engineering solutions provide support for these models, but related work outside the domain of web engineering is more advanced in this respect.

The analysis of web engineering models in section 3.2 shows that some existing web engineering approaches include the concept of user models or context models. However, for this aspect, the state of the art appears less advanced in web engineering than in other research fields, such as model-driven development of interactive, adaptive applications. The following is an overview of some efforts to model users and context of use, both in web engineering and other areas. Some of the ideas will be used in subsequent sections to propose models which are suitable for usability tool support in web engineering.

Context-Aware Applications With WebML

In [Ceri07MDDContext], the development of context-aware applications with WebML (see section 3.2.2) is described. The focus of the publication is the “context-triggered adaptation” of web applications, i.e. updating content, navigation or presentation when there are changes to the environment in which the web application is used, but not necessarily changes to the user or device. This focus is different from the aim of the tools from chapter 4, which concentrate on aspects like the automated analysis of models. When developing context-aware applications with WebML, the context in which an application is accessed consists of properties and attributes for the following:

- The user of the application, and his current activities,
- the location in which the application is used,
- the devices with which the user accesses the application, and
- other aspects of the environment and of the application itself.

To represent the context information as part of the other WebML models, a data-centric approach is chosen – this is an obvious choice given WebML's background. The context model is attached as metadata to existing entities. Figure 5.1 shows an exemplary application data source together with a possible context model. Because the user is of particular interest, the corresponding User entity is the central entity through which a user interface view of the site (SiteView) is connected with other concepts, such as the used device.

The exact properties e.g. of the Device entity are not specified in the paper. Instead, it is expected that the model will be adapted to the needs of a context-aware application. In contrast, the different sub-schemas shown in the figure are general enough to be applicable to most web applications: Users are represented as belonging to one or more groups, context information is stored for each user, and personalization information is also recorded individually for each user.

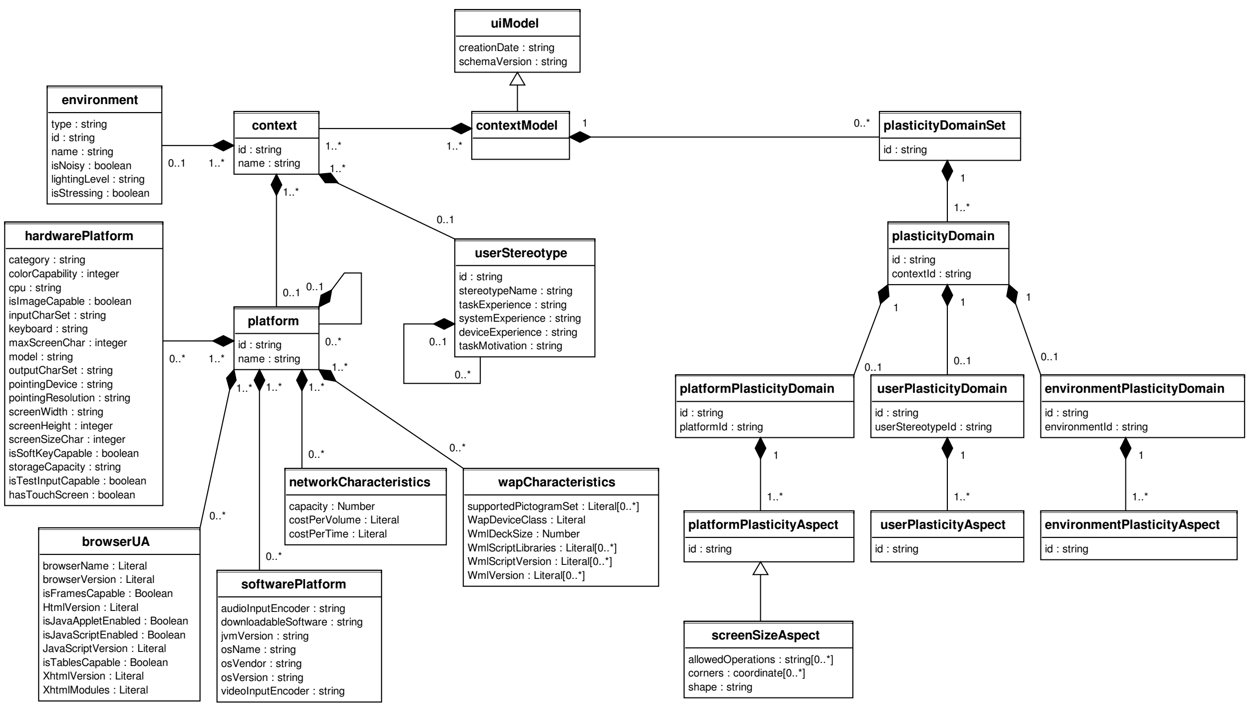
When trying to apply the concepts to the tool ideas presented in this thesis, one difference becomes obvious: The WebML work on context models assumes a setting where the web server will adapt the site to each individual user. In contrast to this, things like model-based usability validation work in a less detailed way, they have to generalize in the sense that they assume certain user *groups* to have properties like "uses a desktop browser" or "is aged 12 to 16". Thus, when adapting the models, it may make sense to associate user groups rather than users with the remaining context information.

Cameleon Reference Framework

In [Calvary03MultiTarget], the authors present a framework which is intended as a reference for the description of user interfaces in the area of context-aware computing. The context of use of an application is described as consisting of three major components:

- The human *users* of the application, described as stereotypical users, for example in terms of the things they are able to perceive or the actions they are able to perform,
- the hardware and software *platform* on which the application runs, which is characterized via the available resources (e.g. memory, screen resolution, bandwidth), and
- the physical *environment* in which it is used. This includes objects, persons and events which may influence the interaction between the user and the application.

The aim of the framework is to allow a developer to create so-called multi-target user interfaces which adapt to the context of use while remaining usable. The outlined process achieves this by first specifying the interaction with the application in an abstract way independent of context, and then successively refining the models until they correspond directly to a user interface for a specific user, device and environment. Due to its goal of providing a generic basis for existing and future work, the paper does not include details on the contents of the models beyond the three parts mentioned above.



Source: `http://www.usixml.org/index.php5?mod=download&file=usixml-doc/UsiXML_v1.8.0-Documentation.pdf`, p. 102, accessed 18 August 2007

Figure 5.2: The UsiXML context model can represent many facts about the users, platform and environment. They are used to adapt the user interface to different usage contexts.

UsiXML

Based on the Cameleon Reference Framework, [Limbourg04UsiXML] describes the USer Interface eXtensible Markup Language (UsiXML), a language for the specification of UIs at different levels of abstraction. Figure 5.2 shows the UsiXML context model. Because one of the declared goals of UsiXML is to ensure good usability of the user interface on different output devices, the model includes many aspects which may influence usability: The user's experience with the system and device can be represented, peculiarities of the platform like network bandwidth and screen resolution receive attention, and a few properties like noise and light conditions in the environment are also part of the model.

While the given model does not include some details like the user's age, extending it would be straightforward. Thus, the model is an adequate basis if context information needs to be represented in connection with the user interfaces of interactive applications, including web applications.

SOUPA Ontology

Ontologies are comparable to models in that both describe a portion of the real world. Thus, it makes sense to have a look at ontologies which concern themselves with the description of user interaction or usability. SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications, [Chen04SOUPA]) is a detailed ontology which is defined using the Web Ontology Language (OWL). It covers a wide range of concepts, including that of a person (with attributes like name, address and contact information), device (type, bluetooth capability) and the environment of use (location of the user).

However, a closer look reveals that the ontology is designed for a field of study which primarily concerns itself with new types of interaction (including physical interaction) with mobile or embedded devices, and less with the usability of any user interface that is presented. For example, this includes the use of mobile phones which employ sensors to receive information about their environment, or location-aware services.

From those properties that are important in the context of this work, only a small fraction are already present in the ontology. For most other things, the ontology would have to be extended. On the other hand, such an extended version of SOUPA would include many aspects which would never be needed for web usability, so all in all, this ontology may not be the best choice for the specification of usability-related user and context information for web applications.

5.2 Model Information for Advanced Usability and Accessibility Tool Support

Summary: To be useful for usability and accessibility tool support, existing models need to be extended with information about the user, the technical platform (e.g. browser) and the environment (e.g. "too noisy to hear any audible clues"). Moreover, information needs to be added to the existing presentation and navigation models.

This section categorizes and explains the different types of model information that are needed for the usability and accessibility tool concepts from section 4. As shown in section 3.2 as well as section 5.1 above, existing web engineering models do not support modelling all required aspects to allow the tools to work properly – in particular, some web engineering solutions do not allow the developer to specify properties of the users and the context of use. In general, three possibilities exist for each aspect that needs to be represented by the models:

- The existing models are sufficient to represent the information (e.g. the presentation model already specifies whether the leftmost column of the page layout contains the site's navigation menu)
- An existing model can readily be extended (e.g. a constraint can be added to the navigation model to demand that a page be ≤ 3 clicks away from the homepage)
- A separate model is necessary (e.g. some existing web engineering solutions do not feature a user model at present)

In this section, UML-based languages are used to represent the models, and the additions and extensions are designed to be compatible with existing models from UWE (section 3.2.1).

However, it is intended that the new model concepts are adaptable for all existing web engineering approaches, which would also enable the proposed tools to work with all these approaches. Section 5.3 describes ways to achieve this, for instance using model transformations and merging. These modelling concepts represent an improved approach over earlier work [Atterer05WebToolSupport] which proposed to store some information outside the web engineering models. The changes to the models are subdivided into the following areas:

- They introduce a context model which describes the user, platform and environment.
- They allow attaching context to the presentation model classes, and giving meaning to page areas, such as "this area of the page is the navigation menu".
- They allow attaching context to the navigation model classes, and specifying meta information about pages, including the purpose of sets of pages and the use of navigation patterns.

This thesis cannot give a definitive set of properties which will suffice for all tools which analyse and improve usability. One reason for this is that development in a special domain may require some aspects to be modelled in greater detail than proposed here. For example, a company which targets mobile devices with its web applications may want to be able to differentiate more accurately between different classes of mobile devices. Another reason is that the tools introduced in this work are only a subset of all possible tool concepts – future tools may require further meta information about the web application. Thus, the models and properties below should be regarded as suitable for a large number of tools related to usability (including all the ones mentioned in this thesis), but they should still be adapted when necessary.

String Tags as an Extensible Way to Categorize Information

One feature of the model extensions below deserves further explanation: In the models, many attributes which describe things like the users' goals are stored as a simple set of strings (`string[0..*]`). Each of these strings should be interpreted as a "keyword" or "tag" which summarizes the respective property, such as a tag `help` for users who are looking for documentation on the use of the web application.

This plain way of specifying the user's interests keeps the model simple. It was chosen over more elaborate, ontology-style modelling of different types of topics because the size and complexity of an ontology would make modelling unnecessarily difficult for the developer. Furthermore, another purpose of the model is to be read by automated tools. A sophisticated ontology would not only hinder the developer, but also require the tool to interpret a complex model, probably only to extract fairly simple information like "the user wants to buy music on the site". Thus, neither the person creating the model nor the program which interprets it would benefit from a more complex model.

However, using tags for user knowledge and goals also has a disadvantage: By default, no meaning is attached to particular keywords by the model. Thus, the same word might be interpreted in different, possibly contradicting ways by different tools. To counteract this, the descriptions below usually document a number of tag values with predefined meaning. Whenever possible, programs should use these predefined values for their task rather than newly chosen tags. On the other hand, the tools should also be configurable and allow the developer to assign meaning to tags that he has introduced himself. All in all, the tag concept follows the idea of UML tagged values, which also require that the person creating the model and the tool which interprets it agree on the semantics of particular values.

With regard to extensibility, a set of string tags represents a compromise between an expressive, formal ontology and a very simple enumeration type with a finite, predefined number of values. Despite the simple structure, it can be flexibly extended: First, new concepts can be represented by a hitherto unused word, which can be assigned a meaning in the preferences of a tool. Second, the following scheme is suggested to support simple hierarchies of tag values: By default, tools should ignore values in the set of tags that they do not understand, with one exception – if an encountered value is a known value followed by "-" and further characters, then the tool should behave as if the known value had been encountered. For example, if `help` is a predefined tag value, then a tag of the form `help-login` should also be interpreted as `help`, unless the tool has support for the "non-standard" `help-login`. Multi-level hierarchies are possible with words containing several "-" characters.

An alternative to string tags would have been the use of many different attributes for the classes of the context model, and to permit extensions by allowing the developer to introduce arbitrary new attributes. In practice, the result would have been similar to the approach proposed above, as string descriptions are the most suitable representation for many properties (e.g. a user goal "buy a CD on the website"), and would have worked equally well. For the purpose of tool support in this thesis, string tags were chosen for their potentially more compact representation and because they are easier to interpret by tools.

5.2.1 Overview of the Context Model

Figure 5.3 shows an overview of the proposed context metamodel classes. They provide the basis for the integration of context and other information into different web engineering models. As mentioned above, they should be sufficient for most new web development projects, so it is expected that the depicted metamodel will normally be reused unchanged by the developer for his models, e.g. without the introduction of new attributes in the various classes. Extensions to the model should only be necessary if there are special needs, such as a newly developed model-based usability support tool which needs context information that goes beyond the possibilities of the proposed model.

In this section, UWE [Hennicker01Modeling] is used as an example to show how to add new elements to an existing set of web engineering models. For clarity, its models are depicted inside an "UWE" package in the figures – UWE actually uses several packages for different types of models. Section 5.3 will show how these changes can actually be made available for use by the UWE method and its tools, by performing a «`merge`» operation of this metamodel with the UWE metamodel. It will also explain that this procedure is equally suitable for adding the information to other web engineering models, such as WebML's.

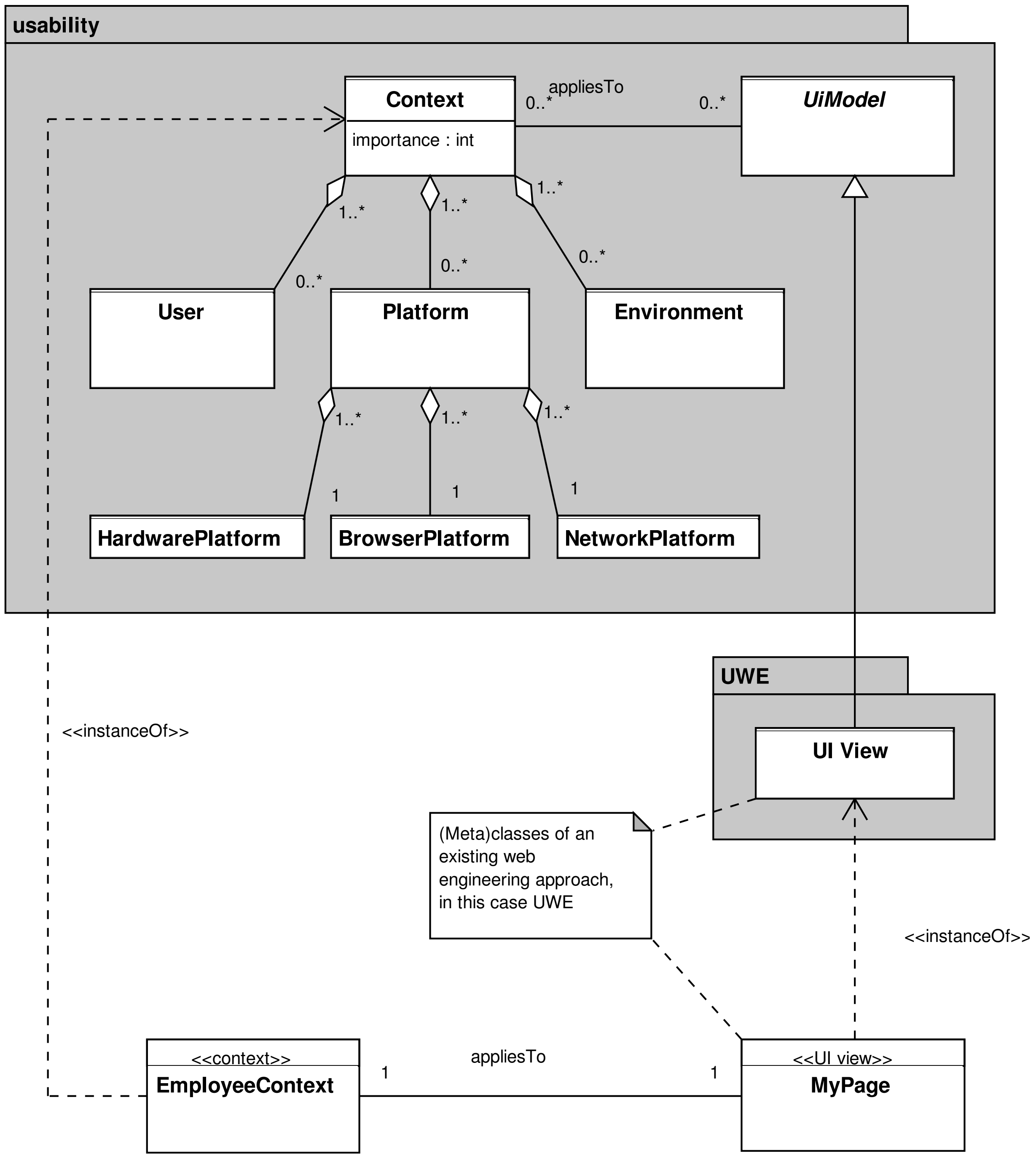


Figure 5.3: The Context metaclass in the usability package describes the user, platform and environment in which a web page is used. The bottom part shows a UI View from UWE as an example. A context object is attached to it via the appliesTo association.

Similar to the approach of UsiXML, the Context is the central concept. Most of the other information is attached to it. This includes the User with user properties, the technical Platform (device, software and network characteristics) and the Environment metaclass which specifies the environment of use and other context information. However, as described below, some further properties are also added directly to a navigation or presentation model.

Several users, platforms and environments can be associated with contexts if necessary. If a tool encounters a case where e.g. more than one User is present, this is interpreted to signify that only these types of users need to be supported by the web application. If no user is associated with the context, this means that tools should assume an "average" user or that all types of users should be treated as equally important. Instead of attaching several types of users to a context, it is also possible to create a hierarchy of users with inheritance. However, by attaching the parent user class to the context, only the common properties specified in that class will be interpreted by tools, instead of the union of all user properties. The same holds for Environment and Platform.

Using the appliesTo association, a context can be associated with other classes to show that it applies to them. In the figure, an example context class EmployeeContext is attached to the class MyPage (bottom right), which is shown as a UWE user interface view. In practice, the class can be any class whose metaclass derives from UiModel. As shown in figure 5.8 later in this chapter, in the case of UWE it makes sense not only to derive the metaclass for «UI view» from UiModel, but also those for «presentation class» and «navigation class». If no context is associated with a class, an "average default context" should be assumed by tools.

Some web engineering environments feature the concept of a set of related user interfaces, typically a collection of pages. Examples include UWE's «UI view» and WebML's site views. A context object can be associated with such a collection to indicate that all its member elements share the same context.

The context information should be consistent for a use case: If an actor participates in a use case, then all the navigation and presentation classes which are later modelled for that use case need to include a context with a User for the actor.

The model in figure 5.4 gives an example for a common case: Several user groups should be supported by a website. In this case, there are three different types of users: The general public, the sales department of the company which supplies the web application, and the administrators of the system. The homepage with a login screen needs to be usable by all target user groups. Other pages of the web application are assumed to be reachable only after a user has authenticated himself. They only need to be suitable for the respective user group. (Navigation possibilities, e.g. from LoginPage to InternalWeb, are not shown in this model.)

Sometimes, pages are *intended* to be used only by a certain users, but are still reachable by others. To represent this, it is possible to attach several Users to a context and to indicate the primary audience by specifying a higher "relative importance" value for the context of a single type of user. This will ensure that users who do not belong to the target group still receive marginal attention. For example, with this information model-based usability validation of the respective HTML page could warn that blind users of a certain website will not be able to select any link which takes them back to the part of the website which is intended for them.

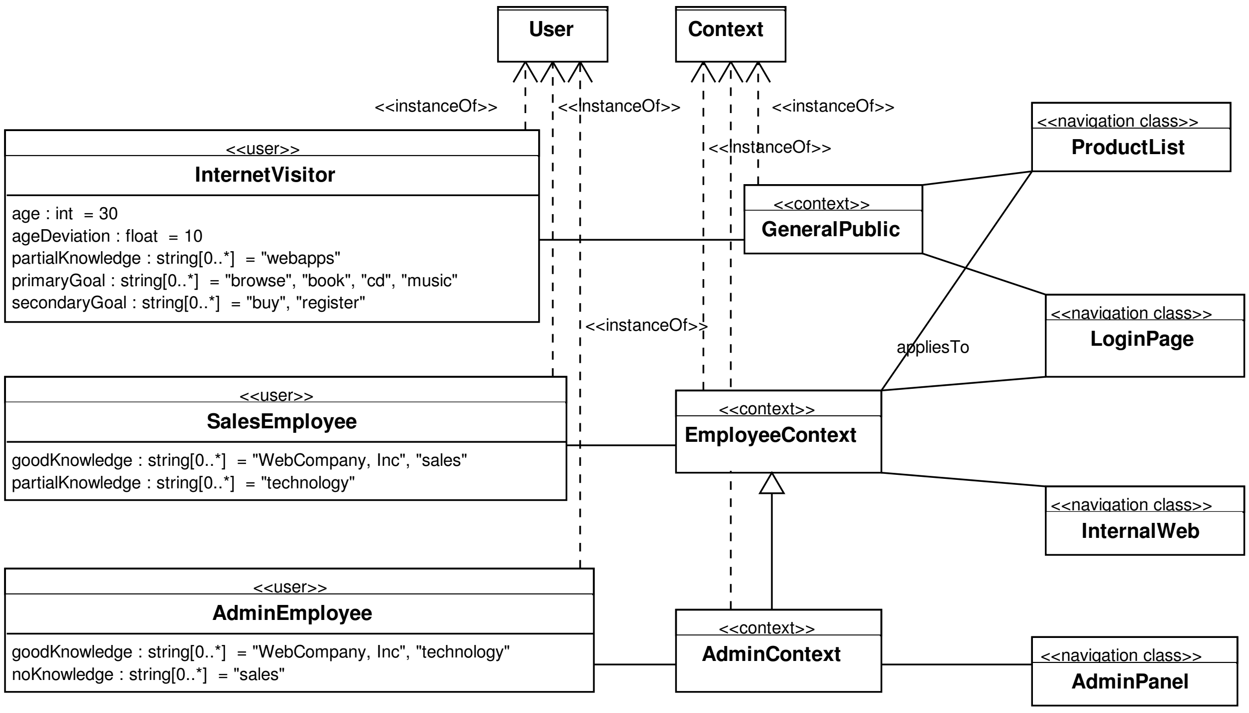

User
Context
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
<<user>>
InternetVisitor
age : int = 30
ageDeviation : float = 10
partialKnowledge : string[0..*] = "webapps"
primaryGoal : string[0..*] = "browse", "book", "cd", "music"
secondaryGoal : string[0..*] = "buy", "register"
<<user>>
SalesEmployee
goodKnowledge : string[0..*] = "WebCompany, Inc", "sales"
partialKnowledge : string[0..*] = "technology"
<<user>>
AdminEmployee
goodKnowledge : string[0..*] = "WebCompany, Inc", "technology"
noKnowledge : string[0..*] = "sales"
<<context>>
GeneralPublic
<<context>>
EmployeeContext
<<context>>
AdminContext
appliesTo
<<navigation class>>
ProductList
<<navigation class>>
LoginPage
<<navigation class>>
InternalWeb
<<navigation class>>
AdminPanel

Figure 5.4: Example for attaching contexts to user interface models: Three target user groups (left) are associated with web pages (right) via appropriate context classes.

The importance attribute of Context is a positive integer. If several contexts are associated with one UiModel, a higher importance value indicates that the respective context is the primary one. The values should be interpreted as specifying a ratio of context uses – for instance, for two contexts attached to a UI model, the two values 1 and 3 would mean that 25% of site visitors with that context are expected to belong to the first context and 75% to the second.

The model also supports another common case: The users of a website can use different devices to access the site, for instance desktop computers as well as mobile devices. The resulting model would look similar to figure 5.4, except that the Platform part of the context would vary for different pages. In the corresponding implementation, the demands of the various devices could be fulfilled either by creating a single HTML page which meets all devices' demands, or with a server-side mechanism that automatically outputs the variant which is more suitable for the requesting device.

5.2.2 Information About the Users

The model which contains information about the user of a web application is denoted by the metaclass User. A model diagram such as the one in figure 5.4 above can use an «instanceOf» dependency or the «user» keyword to clarify that a user is being represented. Each user class stands for a group of users with common properties. It is associated with one or more contexts.

In contrast to UsiXML, it is not intended that a hierarchy of user groups can be associated with a context; one can only attach multiple user classes directly to a context. This approach was chosen because in general, it is unlikely that usability tool support would be able to make use of such fine-grained user modelling, and because of a general goal of keeping the model as simple as possible.

For the specification of users' abilities, existing knowledge about a topic or their goals when they use the web application, the model uses string tags as described above. As an example, consider an automated usability check which looks at the main text content of a page, verifying whether the sentences are likely to be understood. This can also include an analysis of all words that are used – they can be looked up in a dictionary, and the developer alerted to any unknown words which might not be understood by the users (see section 6.3). By default, the corresponding tool will react to some of the standard string tags described below. For instance, when the tag science is present, it might include the Academic Word List [Coxhead00AWL] in its list of allowed words. But in addition to this, user-defined dictionaries should be possible, for example a dictionary which is associated with the company name and contains words which are commonly used within the company, but nowhere else. Going back to the model from figure 5.4, the tag "WebCompany, Inc" could be associated with such a dictionary. If a company-internal term is now used on the LoginPage, the tool can output a warning because the user group GeneralPublic, which is associated with the page, will not understand the term.

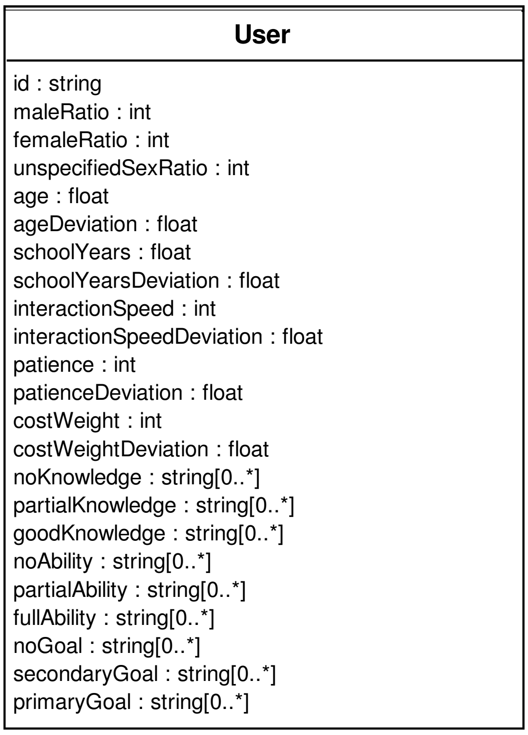
The User metaclass has the attributes listed below (see also figure 5.5). In addition to the requirements of accessibility and usability tests mentioned in this thesis, they were inspired by the UsiXML model (section 5.1) and the user model of SmexWeb (part of the UWE effort, section 3.2.1). Apart from a few general properties like sex and age, the information about the user is divided into three parts – his knowledge, his goals and his physical abilities to interact with the system. This partitioning into three parts is common in the literature [Albrecht00SmexWeb] [Benyon93UserModeling].

id A developer-supplied name for the user group. Apart from helping the developer to organize user classes, tools can use the name to make error and warning messages clearer, e.g. "The usability of HTML page 'X' is problematic because of the properties of user 'Y'."

maleRatio, femaleRatio, unspecifiedSexRatio The ratio of male users, female users and users of unknown sex. While not directly related to any usability test, this basic demographic information can be used to later compare the actual ratio (e.g. from user-supplied data) with the expected one, or to estimate the probability of red-green blindness, which is more frequent for males.

age, ageDeviation The average age of the users, and the standard deviation of the age distribution, normally distributed. As a simplification, a tool may also assume the range of $\mathsf{age} \pm 2 \cdot \mathsf{ageDeviation}$. This interval includes 95% of a normally distributed population.

For example, a tool which analyses text content on a page can use this simplification as follows: Due to the nature of a normal distribution, some (albeit possibly very small) part



```
id : string
maleRatio : int
femaleRatio : int
unspecifiedSexRatio : int
age : float
ageDeviation : float
schoolYears : float
schoolYearsDeviation : float
interactionSpeed : int
interactionSpeedDeviation : float
patience : int
patienceDeviation : float
costWeight : int
costWeightDeviation : float
noKnowledge : string[0..*]
partialKnowledge : string[0..*]
goodKnowledge : string[0..*]
noAbility : string[0..*]
partialAbility : string[0..*]
fullAbility : string[0..*]
noGoal : string[0..*]
secondaryGoal : string[0..*]
primaryGoal : string[0..*]
```

Figure 5.5: The proposed user model relies on sets of string tags to specify the knowledge (e.g. “technology”), abilities (e.g. “input-text”) and goals (e.g. “buy”) of a user group.

of the audience will *always* be too young to read. The tool should only flag this as an error if the age at which children start reading lies within the range $\mathtt{age} \pm 2 \cdot \mathtt{ageDeviation}$.

schoolYears, schoolYearsDeviation The average number of years of school education, and its standard deviation. Increasing values will shift the audience from illiterate persons towards people who went through higher education or university. If necessary, a program interpreting the model can combine this with the specified country (see below) for greater accuracy, e.g. if a learning application targets school children of a particular grade. Again, the deviation value can be simplified to a range as outlined above.

interactionSpeed, interactionSpeedDeviation How quickly the users can interact with the application. This includes both reading speed and speed when performing selections or more complicated operations. A value of 100 for `interactionSpeed` is assumed to mean “average for users described by the other attributes of the model”. Higher values can be employed to describe expert users, low values can be used for an audience with a low reading speed, learning disabilities or other conditions which prevent quicker response times, e.g. “distracted”. The deviation value can be simplified to a range as described above.

patience, patienceDeviation How patient the user is, again with a deviation as for the previous attributes. A value of 100 should be interpreted as “average for web applications and the users' age", which actually means "fairly impatient" in reality [NielsenAlertbox, 28 November 1999] [Krug06Dont, chapter 2]. Higher values imply that long waiting times or detailed text paragraphs are less acceptable and that the probability of errors is higher. In contrast, lower values indicate that the audience may be at leisure, e.g. willing to playfully explore the interaction possibilities on a site.

costWeight, costWeightDeviation As specified in the `NetworkPlatform` model below, accessing the web application may be associated with a monetary cost for the user. Similarly, the use of the web application itself (or some of its features) may not be for free. The willingness of user groups to spend money for online services differs. This value represents the users' willingness to pay, in the form of a ratio of probabilities. Higher values denote that the user group is more likely to pay. For instance, if two user groups have the values 1 and 2, then out of the latter group, half as many users will be deterred by the cost associated with an action. The deviation value allows the developer to express his (un)certainty about the given value.

noKnowledge, partialKnowledge, goodKnowledge Presence or lack of *acquired knowledge or background* for specific topics, indicated by sets of tags. The `noKnowledge` set should be interpreted as "no or almost no knowledge" rather than no knowledge at all. The difference of partial vs. full knowledge is left to a program interpreting the model – for instance, one possibility is to output warnings for partial knowledge and errors for no knowledge. The same tag may appear in more than one set to indicate that the user group is not homogeneous in this respect. Possible values include general topics like "gardening", more specific ones like "eBay auctions", professions like "dentist" or educational achievements like "high school". The tag has no relation to the web application that is being developed – so if e.g. the tag "shopping" is present, this means that the user has prior experience with online shopping, not necessarily that he is interested to buy something now. The following tags (mostly inspired by common use cases for web applications) are suggested:

`technology`	General background in technology
`web`	Experience with the world wide web in general
`webapps`	Experience with web applications such as web-based email
`search`	Searching for information, e.g. with detailed search forms
`commerce`	Online shopping and payment experience
`learning`	Experience with hypertext-based learning applications
`participation`	Contributing to a service rather than just consuming/using it
`science`	Scientific background

Furthermore, differences between different cultural backgrounds can be expressed. Below, `xx` is a two-letter country code as specified by the ISO 3166 standard:

`country-xx`	Familiarity with the customs and culture of a country or region `xx` (e.g. `country-de` for Germany)
`lang-xx`	Ability to understand and express oneself in language `xx`
`locale-xx`	Preferred formats for time and date, numbers, currency etc.

noAbility, partialAbility, fullAbility The sets of abilities which are present, partially present or not present. They affect the users' possibilities to perceive the website content and to operate the web application. Primarily, these sets specify the *available human senses* and output channels. Because they describe the abilities naturally available to a human, they can serve to express disabilities. Additionally, they are suitable to express temporary restrictions, e.g. "while driving his car, the user cannot watch his device, only hear it". noAbility also includes very little ability. Tags which are likely to be frequent include:

see, see-colour	Can see web content, or see it in full colour (partial ability: some difficulties reading, e.g. large font sizes or high contrast required)
hear	Can hear spoken content or audible clues
see-movement	Can consume moving content without problems
see-blinking	Can consume blinking content without problems
tactile	Can consume content using a braille display, vibration alarm or similar
input-text	Can input text characters (partial ability: only slow typing possible, e.g. gaze-tracking keyboard or speech-to-text interface)
select	Can navigate pages, select links or choose between given alternatives
point	Can move and click a mouse or other pointing device on visually rendered web pages (partial ability: reduced fine motor skills)

noGoal, secondaryGoal, primaryGoal The *current goals and interests* of the users, i.e. the aims which cause them to use the web application. The tags for these sets are comparable to what "information scent" algorithms (section 2.3.5) would produce for existing log data: A number of keywords, similar to search engine queries, describing either the general topic(s) of interest, e.g. "music", or the immediate goals such as "buy a CD". The model is not suitable if the granularity of goals or sub-goals gets too fine – in general, the goals should be applicable to an entire use case. A few goals are general and common:

information	The goal is to retrieve information only
transaction	The goal is to update a database or to otherwise change state
education	Special case of information: Looking for detailed explanations
entertainment	Passive or interactive entertainment, e.g. video content or games
collaboration	The users want to work together with others
help	Special case of information: Looking for instructions on how to achieve goals with the web application
buy	Special case of transaction: Buying an item online
register	Special case of transaction: Registering for a service
print	In addition to browsing them, documents will also be printed

5.2.3 Information About the Technical Platform

The Platform model describes the technical means by which the user consumes content and interacts with the website. Again, the UsiXML platform model served as an inspiration. However, this model is slightly more abstract than UsiXML's – for example, it does not include compatibility information such as the operating system version of a device, instead it concentrates on how web pages are rendered. The platform consists of the properties of the hardware device, the software (primarily the rendering engine), and the characteristics of the device's network access. As mentioned before, the model should be extended if a particular class of devices, such as mobile devices, is of special interest to a web application, and if special usability tool support requires additional properties to be specified. Similar to the user model, a set of tags is available as a simple basic facility for extensions, with the discussed disadvantage that the semantics of any newly introduced tags must be agreed upon "ad-hoc" by the developer and the tool.

A Platform consists of a HardwarePlatform, a BrowserPlatform and a NetworkPlatform. It is identified by a string – as with the User class, this string can be used by tools to refer to the platform, e.g. in error messages. If several alternatives are available in a context, such as both a mobile phone and a laptop computer, or two different browsers on a single device, this can be modelled by attaching several platforms to the context. The details of the platform model are shown in figure 5.6.

HardwarePlatform

This class describes the capabilities of the physical device that is used to access the web application.

deviceDescription A string describing the primary form factor of the device. Possible values include desktop, notebook, subnotebook, cellphone, pda and tv, others can be introduced if needed.

supportedAbility A set of tags with the same meaning as those in the user model's noAbility, partialAbility and fullAbility, which lists the types of abilities which are supported by the device.

inputDevices A list of available input devices. The values are more concrete than supportedAbility above, they include the following: keyboard (includes numeric-keyboard, handwriting-recognition), selection-buttons, pointing-device (includes mouse, trackball, touchpad, trackpoint, joystick, touchscreen), scrolling-device.
(For example, a typical mobile phone would be represented by numeric-keyboard, joystick, selection-buttons and possibly speech-recognition.)

outputDevices A list of available output devices. Possible values include: screen (graphical display), characters (i.e. non-graphical text display), loudspeaker, braille, vibration, printer. As with other sets, new tags for input or output devices may be added if necessary.

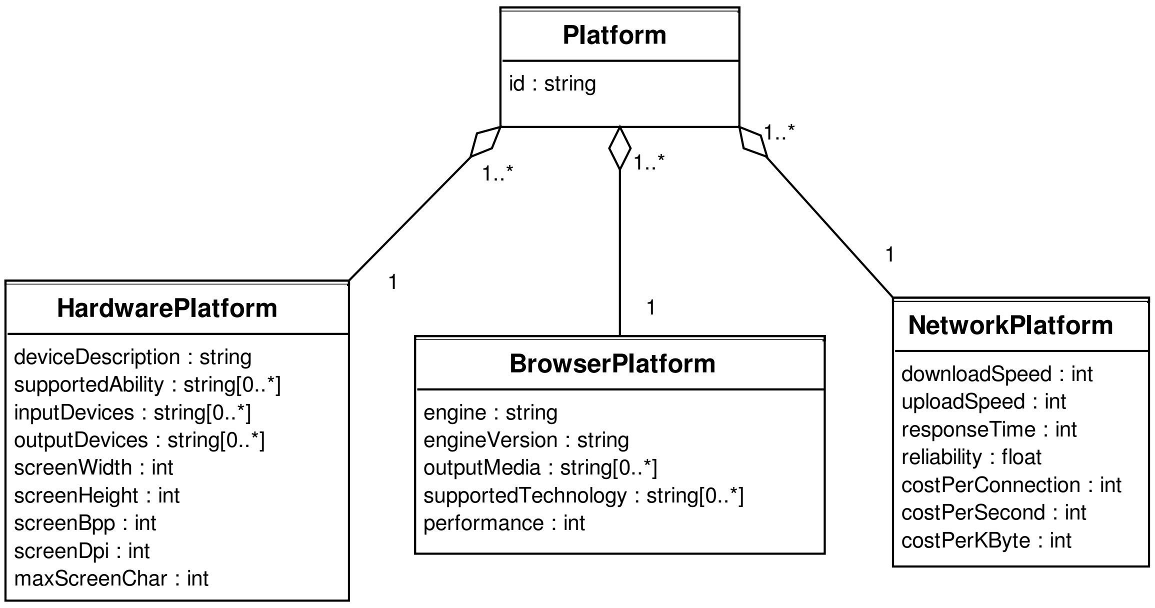


Figure 5.6: The platform model consists of information about the hardware device, the browser running on it and the type of network access.

screenWidth, screenHeight Screen resolution; number of pixels horizontally and vertically (0 if not applicable, i.e. not pixel-based)

screenBpp Number of colours of the display, in bits per pixel (1 for monochrome displays)

screenDpi Resolution of the screen, in pixels per inch (0 if not applicable, i.e. not pixel-based)

maxScreenChar Maximum number of characters that can be displayed (0 if not applicable). For pixel-based displays, this value is the number of characters that can be rendered at the default font setting and line spacing, with an assumed character width which corresponds to the average of the lowercase alphabet characters. The value can vary significantly – for instance, braille displays may only be able to display a single line of text. Thus, tools should take care not to implement overly simplistic algorithms, e.g. "text paragraphs on a web page should be shorter than maxScreenChar".

BrowserPlatform

In this class, properties of the rendering agent are specified. Because the primary focus is web applications, other, non-browser software which may be installed on the device is not described by the model.

engine The name of the rendering engine, such as mozilla, opera, webkit or msie, or the value all to include all engines which are standards-complaint as well as the ones which are most popular.

engineVersion Version information about the rendering engine

outputMedia The set of CSS media types [W3C-CSS2, section 7] which the rendering engine can create output for – one or more of the following: all, braille, embossed, handheld, print, projection, screen, speech, tty, tv. Values which are irrelevant in practice should be avoided. For example, the "FireVox" extension of the Firefox web browser allows speech output, and the Opera browser can switch to projection or handheld output, but unless these unusual capabilities are really important for a web development project, they should be omitted to avoid that tools are misled.

supportedTechnology A set of strings identifying different technologies which are implemented in the browser, including these: xhtml, frames, javascript, flash, svg, xml, xslt.

performance A measure of how quickly the software can render HTML pages or execute JavaScript code. It is difficult to devise a unit of speed measurement which takes into account different aspects, such as any tradeoff between speed and memory requirements, JavaScript execution speed, CSS rendering speed, first-time rendering vs. rendering of pages visited before, etc. For this reason, no such attempt is made here. Instead, the values only indicate a ratio of speeds of the different BrowserPlatforms present in a model. For example, if two platforms have two performance values of 1 and 4, then the second browser engine will be about 4 times as fast as the first one. The developer can assign values subjectively or base them on performance testing.

NetworkPlatform

This class characterizes the way the device accesses the Internet in general, and the web application in particular.

downloadSpeed, uploadSpeed Average available bandwidth for receiving and sending content, in bits per second

responseTime Typical time in milliseconds from the sending of an HTTP request until an answer is received. This includes the time to send the request, processing of the URL by the web server software, and receiving the response, but not the time of any server-side HTML generation mechanism, database lookup or similar. In other words, it is the time taken for the simplest imaginable HTTP request, e.g. an XMLHttpRequest which is answered with a zero-length reply.

reliability Probability in the range 0..1 of any single HTTP request to fail due to technical problems, such as bad wireless reception, DSL router re-connect etc. For wired networks, a very low value like 0.001 can usually be assumed, whereas wireless connections are generally more likely to fail.

costPerConnection, costPerSecond, costPerKByte The monetary cost associated with the use of the Internet connection. No particular currency is assumed, so the values in all models for a web development project need to be based on the same currency. costPerConnection

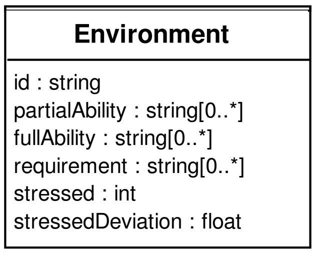

Environment
id : string
partialAbility : string[0..*]
fullAbility : string[0..*]
requirement : string[0..*]
stressed : int
stressedDeviation : float

Figure 5.7: The environment model describes the way the use of the web application is influenced by external factors.

is the cost for one "session" (e.g. in a mobile GPRS network), not the cost for one HTTP or TCP connection.

Rates which cannot be described directly by these properties should be approximated with values which would lead to the same user behaviour. For example, for flatrates a cost of zero could be used, on the grounds that the user will not get more impatient or be deterred by the cost of access, as it is already covered by the general flatrate fee.

5.2.4 Information About the Environment

As shown in figure 5.3, each Context is associated with zero or more User, Platform and Environment classes. The environment class contains details on the surroundings and other external factors which are present when the user accesses the web application. The model includes those factors which may influence the user experience (for example, the surroundings may repeatedly distract the user), the user's expectations from the system (for example, causing him to be less patient than usual) or the behaviour of the device (for example, there is too much noise for the user to hear any audible output).

The requirements for an environment model for usability and accessibility tool support are contradicting: On one hand, the model should describe the situation of use well, and allow the representation of any aspects which might affect the users' ability to reach their goals. On the other hand, the model should be simple enough to permit it to be specified quickly by the developer, and to allow tools to extract information relevant for them. The solution of making the model easily extensible is not ideal, as any existing tools will not be able to make use of information added via a later extension of the model.

With regard to related work, some existing models choose the approach of detailed modelling. For instance, with SOUPA there is a set of core concepts like persons, actions or space, a number of existing extensions for situations such as "meeting", and a way to supply further extensions for specific circumstances. In contrast to this, the environment class in UsiXML simply restricts the conditions in the environment to a very small number of parameters, namely noise, lighting level and how stressing the environment is. While such a model will be easy to interpret both by humans and programs, it may not be able to describe the environment accurately enough.

The environment model proposed below tries to balance the advantages and disadvantages of a simple and a more complex model. Due to the fact that only the web application running on the device is of interest, it is possible to follow UsiXML's general approach and only to model the *influence* of the environment on the use of the web application, rather than the environment itself. At the same time, the model is more elaborate than UsiXML's: There is more fine-grained control over which human senses are affected by a particular environment, and some additional parameters can be represented, such as how concentrated the users are. In general, the model is fairly abstract, which makes it applicable to many types of environment. If more detailed information is required for a specific environment, it is possible to extend the classes with additional attributes, or to assign meaning to additional string tag values. Figure 5.7 shows an overview of the attributes of the Environment metaclass.

id A developer-supplied name for the environment. Similar to the user and platform IDs, the name can help the developer to organize different platforms, and it can be used by tools for error reporting and related purposes.

partialAbility, fullAbility A set of tags with the same meaning as those in the user model's noAbility, partialAbility and fullAbility, which lists the types of abilities which are supported or partially supported by the current environment. For instance, the see ability will not be present in this model if the application is intended for use while the user is driving a car, or while carrying the device in his pocket. (If the device is used both with and without looking at it, this can be modelled by two different contexts, which are also distinguished via different user goals.) Tools can form an intersection of this set of tags with the ones in the User and HardwarePlatform classes to find the senses that are available in a particular context.

requirement A set of tags, each of which is a description of certain constraints or requirements imposed by the environment on the use of the application. Tools may check for the presence of specific tags and enable a set of special checks if a supported tag is found. This can be used to further describe the activity of the user with tag values like travelling, driving, athome or onholiday – any tool which makes use of the model in this way needs to document which tag values it recognizes, as well as the resulting changes in the tool's behaviour.

However, this attribute also serves to record legal requirements which are related to accessibility or usability: While the targeted user group of an official government institution may not include people with disabilities, the institution may nevertheless be required by law to apply measures to support access by the disabled (see section 2.2.2). Suggested values for guidelines include the following: wcag1-a, wcag1-aa, wcag1-aaa, wcag1 (identical to wcag1-aaa), wcag2-a, wcag2-aa, wcag2-aaa, wcag2 (identical to wcag2-aaa), bitv, section508, stanca-act or generally accessibility-law-xx where xx is a two-letter country code like de.

stressed, stressedDeviation A measure of how much at ease the users are the moment they use the application, or how stressed they are. A value of -100 indicates that they are completely at leisure, or even bored, with no time pressure at all, +100 means they are extremely stressed and may only have a few seconds of time to use the application. Programs interpreting the model should combine this value with the general patience level of a user group to estimate the actual level of patience. For instance, a simple heuristic would be to demand that the web application fulfils the stricter of the two values. For a tool which measures the response time of a database query, this would mean that short response times are required if a user group is impatient by nature or if the environment is stressful. Longer response times are only allowed if *both* the users are patient and the environment is not stressful.

5.2.5 Presentational Aspects

Existing web engineering approaches either include a presentation model which specifies the major components that make up pages of a certain kind (e.g. UWE), or a template document with the same purpose which can be transformed into such a model (e.g. WebML). For some of the tool concepts outlined in other chapters, this model needs to be augmented with additional information. For example, model-based usability analysis of the final implementation may include a test which checks for the presence of a de-facto standard page layout, with e.g. a navigation menu at the left side of the main content.

Adding Context Information to Presentation Classes

As already explained above, for any model which describes the contents of a page or the navigation layout, it must be possible to attach a context class with information about the users, platform and the environment. Figure 5.8 shows that this is achieved in the metamodel by introducing the abstract metaclass UiModel, which supports the appliesTo association. The UWE metaclasses like NavigationalClass are then derived from the UiModel class.

Assigning Meaning to Areas on the Page: "role", "importance"

Furthermore, it must be possible for tools to perform a mapping from an area on the final HTML page to the functionality which is provided in this area, e.g. "this area of the page is the navigation menu". This information needs to be added to the metaclass which represents one page area – in the case of UWE, PresentationClass [Hennicker01Modeling]. For this purpose, as shown in figure 5.8, the abstract metaclass PresUiModel is derived from UiModel. Its role attribute is a string which describes the purpose of the page area, and PresentationClass inherits the attribute.

Sometimes, functionality is distributed in a finer-grained way inside a page area – for instance, the role of a single text field may differ from that of its parent page area. To avoid the need for the developer to introduce «presentation class» classes in this case for the sole purpose of assigning a role to them, the UI Element metaclass (whose instances like «text» make up the content of a UWE PresentationClass, see figure 3.3 on page 45) inherits from PresUiModel as well.

For the keyword string stored in the role attribute, the following values have a predefined meaning. In part, they were inspired by the proposed XHTML role attribute [W3C-XHTMLrole]:

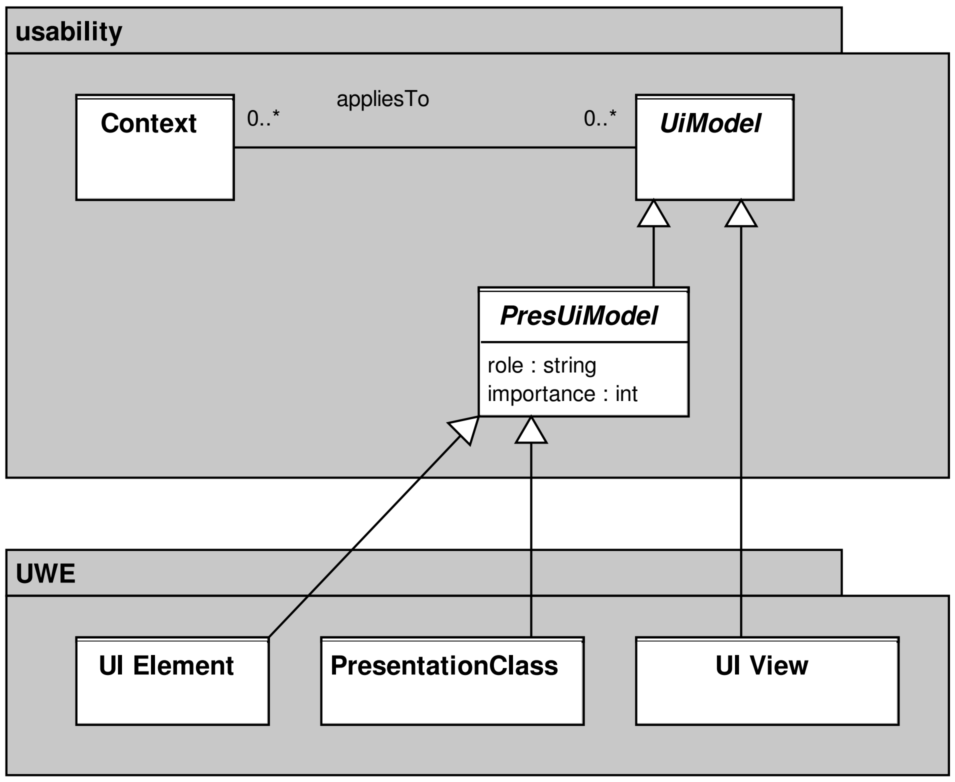


Figure 5.8: Extensions to the presentational models (e.g. assigning roles like "navigation menu" to page areas) are introduced via the metaclasses `UiModel` and `PresUiModel`.

main	The main text (or other content) of the page
seealso	Additional content whose topic is related to the main content, e.g. an abstract in a sidebar.
secondary	Additional content which is not related to the main content, e.g. information about other parts of the site or a "weather forecast" box in a news article
logo	The logo of the company or website
ad	Advertisement in the form of banners or animations, or in the form of text links (so-called "sponsored links")
empty	Empty space on the page, uniformly coloured or at most showing a faint, unobtrusive background image
ornamental	Space on the page which is not empty, but which serves no purpose other than contributing to the visual page design
navigation	Links for navigating the document or site. If possible, the more specific values below should be used.
navigation-menu	The main menu of navigation links, possibly with multiple hierarchy levels

navigation-bar	A secondary, shorter navigation menu – this includes “tab bars” with links to major parts of a large site, and breadcrumb links
navigation-other	Other navigation aids on the page, such as tag clouds. In an online shop, this can also be a “related items” section or the shopping cart
search	A field to search the site, or a link to a search page
login	User account management options, including “login” and “register user”
help	A link to a help page, or a section with instructions on how to use the site
contact	A link or other element to contact the site maintainer
other	Other content, which does not fit the categories above or which should be ignored by tools

Other values may be used by developers, and programs may include special support for additional keywords. For instance, a tool which specializes on analysis of the text on a page could potentially support keywords which classify different types of text content.

With the ornamental keyword, a facility exists to declare parts of the page as ornamental. This value is mostly useful to assign to images which act as spacers, rules or graphical frames around areas. It is also appropriate for “mood images” which are intended to attract the curiosity of a sighted reader, but which are not essential for understanding the content.

As described at the beginning of section 5.2, tools should ignore (i.e. treat as type other) values they do not understand, unless an encountered value is a known value followed by “-” and further characters. For example, a program which works with main page areas should also treat any main-text area like a main area.

Finally, the role value can also be the empty string (which is not to be confused with the value “empty”). For nested user interface elements, the empty string indicates that the developer did not assign a role value to the UI element in question, but only to one of its parent objects (a page area that contains the element). Thus, the element itself can be considered to be part of the implementation of the higher-level area. For example, a “search” button will usually have an empty role value because the screen area which contains the search field and the button will already have been assigned the search functionality.

The importance attribute is intended to be used by the developer to assign more importance to those areas of the page which he considers central for the web application, and for which usability problems are less acceptable than elsewhere. Similar to the importance attribute of contexts (section 5.2.1), the positive integers stored in this attribute represent the relative importance of one area over another, with higher values indicating higher importance. Tools can use this information e.g. for ordering the results of their analysis so that important areas are listed first, or possibly to increase their demands for the usability-related properties of this part of the user interface. For example, the developer of an online shop may assign the highest importance value to the online shopping cart, and this may cause his web engineering tool to require that a keyboard shortcut is assigned to the respective link on the page.

5.2.6 Navigational and Functional Aspects

Similar to the presentation model, current web engineering solutions also include a model which serves to describe the navigational behaviour of the web application. For the approaches described in section 3.2, a navigation or hypertext model records the pages and the possible paths that users can take while using the application. Some extensions to the information in these models can provide valuable input for automated tools which want to analyse and improve the usability of the application.

NavUiModel (figure 5.9) is the metaclass which corresponds to a page in the web application. Similar to PresUiModel, a pageImportance attribute can be used by the developer to assign more importance to pages which are central for the success of the web application (e.g. the “proceed to checkout” page of an online shop). The positive integers in the attribute only have a meaning when compared with other pages’ pageImportance values, higher values indicate greater importance.

Adding Context Information to Navigation Classes

It makes sense to associate different parts of the navigation model with Context classes, in the same way as for the presentation model in the previous section. Usually, as outlined by the example in figure 5.4 (page 94), one or more context classes will be connected with each navigation class, and thus each HTML page. Figure 5.9 shows the metamodel which allows a UWE NavigationalClass to be associated with a context class by deriving it (indirectly via NavUiModel) from UiModel. It is at the developer’s discretion to decide whether to include context in the presentational model only, or in both the presentational and navigational model. (With UWE, the presentation model is a refinement of the navigation model, so providing context modelling in the presentation model alone is not advisable.)

Goals Which Should be Achievable on the Page: “purpose”

Many tool concepts benefit from a more abstract description of the purpose that a page or set of pages fulfils on the site. For example, model-based analysis of the website navigation could reveal that access to online help content is not possible at a certain stage while performing a task on the site – but this is only possible if the tool knows which part of the content contains help. The purpose attribute of the NavUiModel metaclass is a set of string tags which specifies the main purposes of the page(s). The possible values are identical to the ones which are available for the noGoal, secondaryGoal and primaryGoal attributes in the user model (see section 5.2.2). In other words, the purpose attribute specifies which goals of the users are supposed to be achievable using the page. The goals can be specified both at an abstract level using the predefined values (information, transaction, education, entertainment, collaboration, help, buy, register, print), and at a more concrete level using tags to which the developer has assigned special meaning in a tool.

It should be stressed that there is a significant difference between this purpose attribute and the role attribute in PresUiModel. The tag value help is even available and can be used for both,

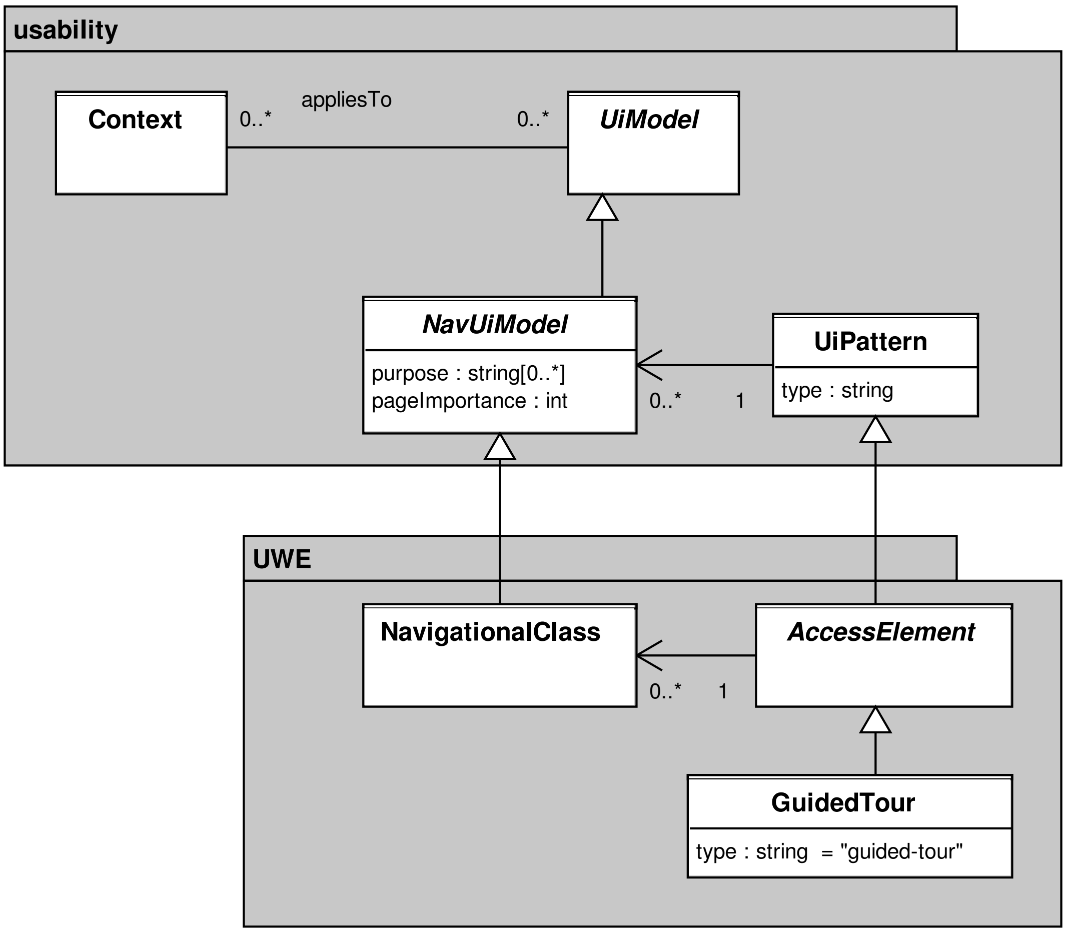


Figure 5.9: Navigation classes can be associated with contexts. Additionally, the metaclasses allow specifying the purpose of a page, its importance and the use of any design patterns.

but it does not have the same meaning: In a NavUiModel, a purpose of help indicates that the entire class (and eventually the resulting HTML page) is dedicated to providing online help to the user. In the PresUiModel, the role help instead means that the page contains some user interface elements which have the purpose of helping the user – in the simplest case, this will be a "Help" link appearing on the page. Other parts of the page may have different roles. To understand the relation between purpose and role, it is beneficial to keep in mind that if the role in the presentation model for a page area is main, then this main content is further described by the purpose of the same page in the navigation model.

Navigation Patterns

The navigation model is also the right location to store information about navigation patterns. For instance, if the model includes the fact that a "guided tour" pattern is being employed, a tool could use this to verify whether the final website correctly implements the pattern. In figure 5.9, a pattern is represented using a UiPattern class. It uses a directed association to reference the NavUiModel classes which participate in any instance of a pattern. The type of the pattern is given as a string, which allows new pattern types to be added simply by using a new string value. Patterns which are commonly supported by existing web engineering solutions include the following: index, guided-tour, query, menu, landmark, home [Koch00Adaptive, section 6.4.2] [Ceri00WebML, section 3.7] [Kappel04WE, section 3.6]. If necessary, this list can be extended – for example, a WebML developer might more accurately use guided-tour-indexed for a particular kind of guided tour. In case a pattern cannot be described by its name alone, it is also feasible to introduce a subclass of UiPattern with additional attributes.

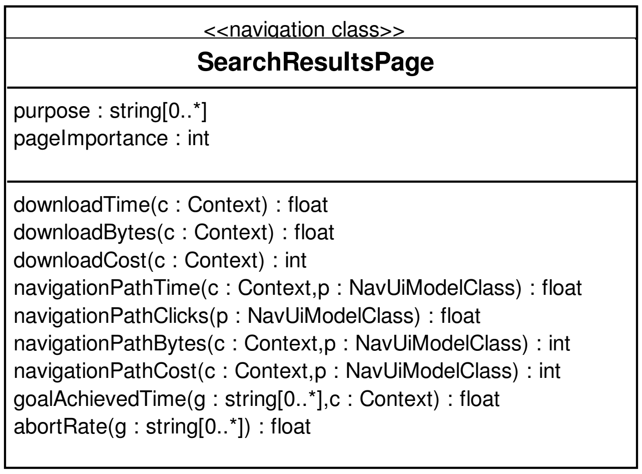
When adding a representation of patterns to existing web engineering models, the task is complicated by the fact that different solutions' models already include support for patterns, and that the concept of a pattern varies slightly between them. The aim of the metamodel proposed in this chapter is to make usability tool support independent from any particular web engineering environment. Consequently, it is desirable to have the information "the pattern consists of the following pages" present in the part of the model which will not change for different web engineering environments, to avoid that tools have to be adapted to extract the information for each specific web engineering environment.

Figure 5.9 shows a relatively straightforward case: With UWE in mind, the NavUiModel and UiPattern metaclasses as well as the association which connects them were designed in such a way that the UWE-specific metaclasses (NavigationalClass and AccessElement) exactly match their parent classes. For WebML, the process is also fairly unproblematic: There exists no concept which is directly equivalent to UiPattern, as patterns are modelled in a more indirect way. Thus, when the developer makes the presence of patterns more explicit, this does not result in duplicate information being added to the model. For other web engineering solutions, model transformations may possibly be required to cleanly integrate the NavUiModel/UiPattern pattern information.

OCL Constraints for Detailed Requirements

For some aspects which are related to usability, the most promising way to add them to the existing set of models is to use constraints in a formal constraint language. For the models in this work, which are mostly UML-based, the Object Constraint Language (OCL, [Warmer98OCL] [OMG06OCL]) is a good choice. Constraints are particularly suitable in the case of non-functional requirements which are measurable in some way: The developer may want to combine their values in non-trivial ways, so a mathematical expression in the form of an OCL constraint is more flexible than e.g. an additional class attribute.

A number of examples for useful constraints are listed below. Appropriate tools can determine whether the constraints are fulfilled by an implementation, especially in the controlled environment of a user test. The constraints rely on the presence of a number of methods for any classes which are instances of the NavUiModel metaclass. Figure 5.10 shows an example UWE navigational class, whose attributes are provided by NavUiModel and whose methods need to be



<<navigation class>>
SearchResultsPage
purpose : string[0..*]
pageImportance : int
downloadTime(c : Context) : float
downloadBytes(c : Context) : float
downloadCost(c : Context) : int
navigationPathTime(c : Context,p : NavUiModelClass) : float
navigationPathClicks(p : NavUiModelClass) : float
navigationPathBytes(c : Context,p : NavUiModelClass) : int
navigationPathCost(c : Context,p : NavUiModelClass) : int
goalAchievedTime(g : string[0..*],c : Context) : float
abortRate(g : string[0..*]) : float

Figure 5.10: Example navigation class based on the NavUiModel metaclass. The methods are useful for OCL constraints, e.g. to measure the success rate of user goals.

realized in the model and implementation as described in the rest of this section. The method signatures use a hypothetical NavUiModelClass to denote classes which are instances of NavUiModel or one of its child classes.

downloadTime(c : Context) The average time in seconds it takes to download the page including all embedded images and other data, under the circumstances described by the Context. This method is useful in constraints about timing requirements for operations such as database operations, but also as a general "all pages should not take longer than *n* seconds for loading":
context NavUiModelClass **inv**: self.downloadTime(c) < 5.0
In general, it makes sense to define the invariant in such a way that it demands the timing constraint to be fulfilled by all Context objects which are associated with a page – see figure 5.11:
context NavUiModelClass **inv**: self.downloadTime(self.context) < 5.0

downloadBytes(c : Context) As above, but the method returns the total number of bytes which need to be downloaded in the given context for the page to be displayed completely.

downloadCost(c : Context) As above, but the cost of downloading the page is returned. Rather than measuring it directly, it is calculated from the information in the context's Environment class by default, but an implementation may override the method with code which can return a more accurate value.

navigationPathTime(c : Context, p : NavUiModel) The time in seconds it takes to navigate from page p to the current page, i.e. the one whose navigationPathTime() method is called. During a user test or after the site has gone live, the navigation paths of users through the site are recorded. The returned value is the average time that users took to reach the current

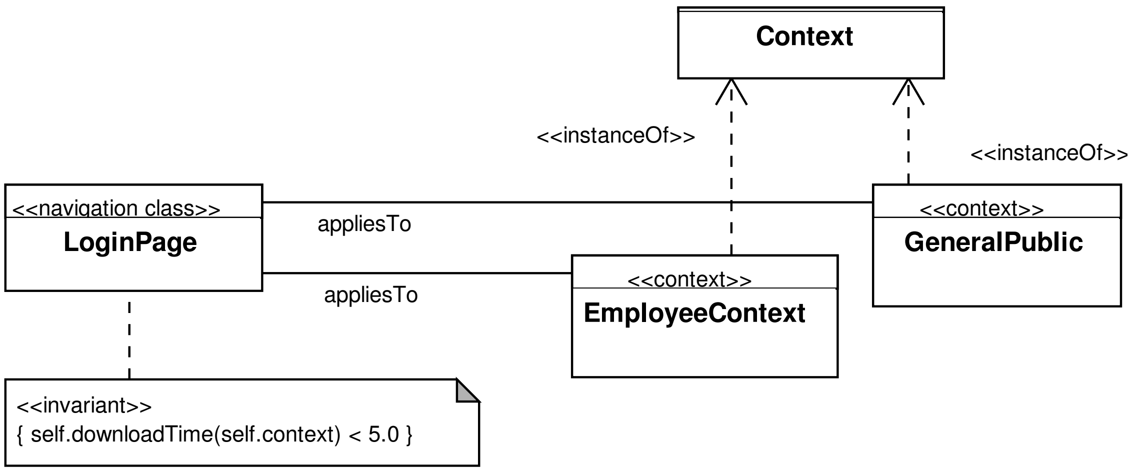


Figure 5.11: Usability-related requirements can be expressed using OCL. In this example, a download time of < 5 seconds is needed in both usage contexts of LoginPage.

page from page p. Heuristics are required to prevent that some users' paths are included, for example by eliminating browsing sessions with long times of inactivity.

navigationPathClicks(p : NavUiModel) The minimum number of clicks necessary to reach the current page from page p. This method can be used to specify constraints like "this page should be no more than three clicks from the homepage". Problematic cases are possible where this measurement can give a wrong impression – the constraint can be fulfilled, but only if the user traverses links which are not placed prominently on the page. To avoid this, the *calculated* minimum number of clicks should only be used for automated usability analysis. During a user test, the method can instead return the average actual number of clicks needed by the test users.

navigationPathBytes(c : Context, p : NavUiModel) As above, but returns the number of bytes associated with the navigation path.

navigationPathCost(c : Context, p : NavUiModel) As above, but returns the cost associated with the navigation path.

goalAchievedTime(g : set of strings, c : Context) When called in the context of a page, this indicates that by reaching the page, the users are supposed to have reached the goals listed in g. The method returns the number of seconds which the users required to achieve their goal. Useful values for the goal parameter include all the general ones defined for the secondaryGoal and primaryGoal attributes of users (page 98), but also others which are more specific to the web application. It is assumed that the system has a way of knowing when the users' search for the goal started, and that it was identical to one of the entries in g. During a user study, these parameters can be controlled. At a later time, when the site is in productive use, it is possible to some extent to employ probabilistic approaches to determine

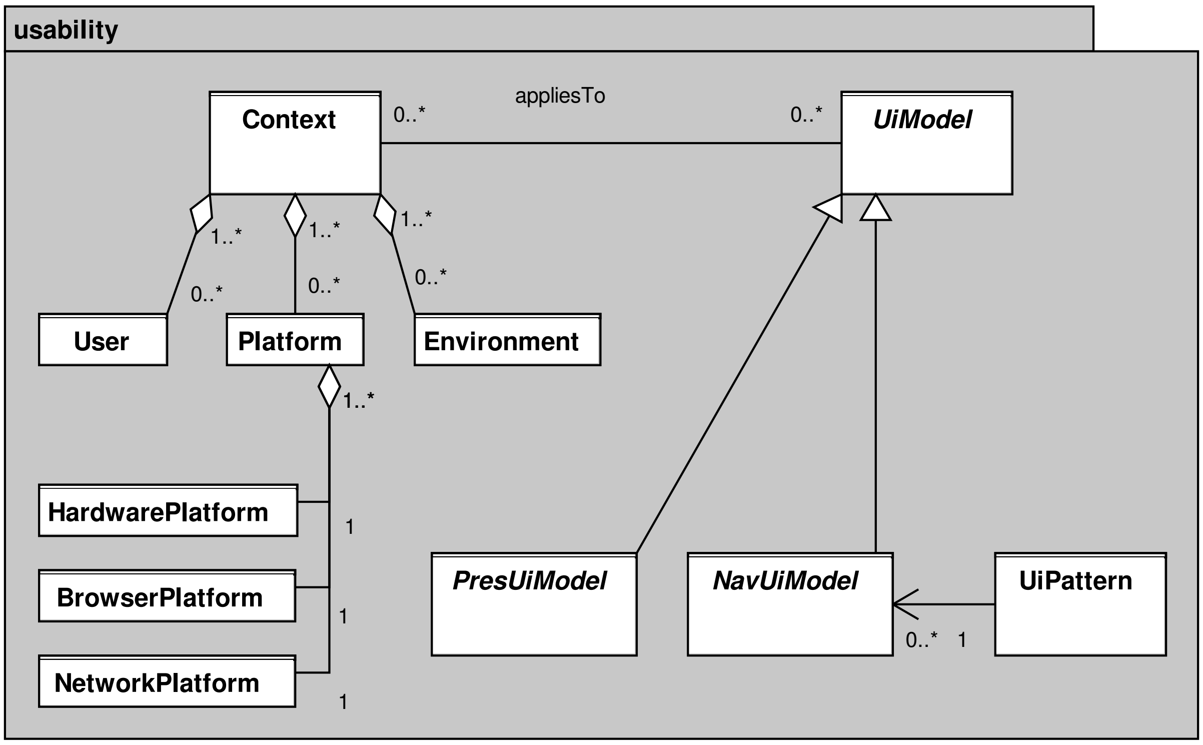


Figure 5.12: Overview of the metaclasses in the usability package.

them, such as the ones related to "information scent" (section 2.3.5, [Card01InfoScent]). Example invariant:
context p : NavUiModelClass **inv**: self.goalAchievedTime({"buy CD"}, c) < 600.0

abortRate(g : set of strings) For the current page, returns the measured fraction of users (in the range $(0;1)$) who aborted their current task or goal at this point, either by indicating so explicitly during a user test, or by leaving the site altogether. If g is empty, the value applies to any goal, otherwise it is restricted to users who are known to have one of the listed goals.

5.3 Integration With Existing Web Engineering Models

Summary: It is possible to incorporate the proposed model extensions into different web engineering solutions' models, either using «merge» or model transformations. This makes it easier to create usability tool support which can work with different types of models. Special steps are needed for XML-format presentation models.

The previous section has shown how to add information about the context, presentational and navigational aspects to an existing set of web engineering models. This was done by combining newly introduced metamodels (from the usability package) with existing metamodels from

UML-based web engineering (UWE). This section more thoroughly describes the changes required at the metamodel level to cleanly integrate the proposed model extensions into existing models. Figure 5.12 shows an overview of the classes in the metamodel. Together, they form the usability package.

The goal behind the clear separation of the model extensions is to design the extensions to be "portable" to other web engineering environments and models. This includes making the extensions available to all the web engineering solutions mentioned in section 3.2, such as UWE, WebML, OO-H and others. Another important aim is portable tool support: It should be avoided that any model-based tool must be adapted for each web engineering solution to allow the tool to read that web engineering solution's models. The presence of the common usability metamodel is a way to achieve this, as it enables a tool to keep track which class in the existing web engineering model represents which abstract concept – for example, any presentational metaclass will be derived from PresUiModel. Furthermore, most of the properties which are of interest to a tool with usability/accessibility support are specified in one of the metaclasses.

Integration is only necessary once for a particular set of web engineering models. As outlined below, there are several ways to integrate the model extensions depending on the features of the web engineering model that is to be extended. Figure 5.13 shows how the usability model extensions are formally integrated into UWE models:

- Two «merge» operations are used to copy both UWE's metamodels and the usability model extensions into the same namespace, which is called usabilityUWE in the figure.
- Model elements are added which connect the web engineering model with the extensions. For example, UWE's NavigationalClass metaclass is made a child class of NavUiModel.
- Finally, some elements of the existing models may also have to be modified. A typical example is that in UWE's representation of a "guided tour" pattern, the type property, inherited from the UiPattern class, is set to the fixed value "guided-tour".

The UWE example in figure 5.13 is a slightly simplified version of the actual model. Apart from the fact that not all relevant classes and attributes are shown (e.g. other pattern classes in addition to GuidedTour), the UWE package structure is more sophisticated in reality, with a hierarchy of packages for different aspects of the modelling task [Knapp04ArgoUWE]. To avoid the need for the developer to duplicate information about patterns in the model, the information from the association between UWE's NavigationalClass and AccessElement is "copied" to NavUiModel and UiPattern by means of a union with a single subset for each association end.

A few prerequisites must be fulfilled for a similar procedure to succeed with another web engineering solution's models. First, obviously the notation that is used for all models in this chapter is the Unified Modeling Language – however, not all existing models use UML. For the web engineering methods which are most important in practice (WebML, UWE and OO-H), this is not an issue most of the time (see below for an exception) as they are UML-based or at least provide an alternative UML-based notation for all concepts. In general, a trend towards using

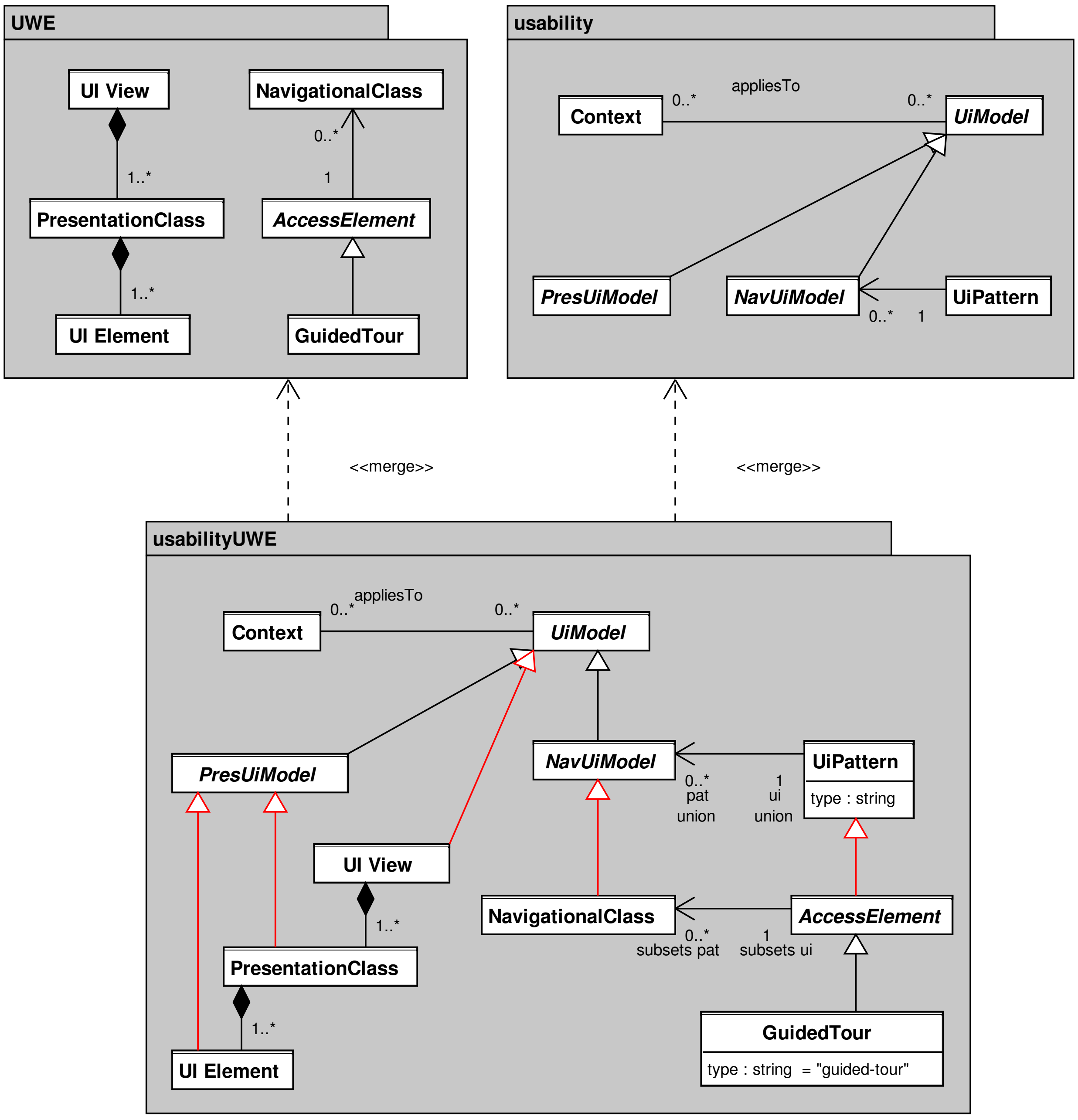


Figure 5.13: The «merge» operations copy existing web engineering models (top left) and the extension classes (top right) into a common package, usabilityUWE (bottom). In the next step, they are combined, primarily using inheritance (model elements shown in red).

UML is noticeable [Kappel04WE, section 3.10], so it is unlikely that future developments will be based on other modelling notations. If the notation is not UML-based, the model extensions cannot be applied – a UML profile must be created for the notation, a UML model must be produced using automatic model transformations, or a UML model which can express equivalent semantics must be created from scratch. In case this proves inconvenient or if the effort cannot be justified, one last resort exists: As shown below in section 5.4, the non-UML modelling solution could be used to output an implementation in which the information from the usability package is embedded, e.g. using a Microformats-like syntax.

Another prerequisite for a successful integration of the usability-related information is that the concepts of navigational and presentational models exist in the web engineering solution. In most cases, this is not a problem – the concepts are general enough to be applicable to most existing models. Sometimes, the distinction between navigation and presentation models is not very pronounced. For example, with UWE the presentation model is only a refinement of the navigation model. For these kinds of models, it is also possible that the class which represents a page inherits from both NavUiModel and PresUiModel.

Finally, the classes of the usability package contain a number of assumptions: HTML pages are represented with classes (rather then, say, associations), any user interaction pattern references several navigation classes using a directed association, no suitable user model is present so the one introduced in usability should be used, etc. A web engineering model can conceivably have a completely different view of these concepts, or it can already contain information about the context of use which should not be duplicated in another part of the models. If such a mismatch between models occurs, again automated model transformations are a promising solution: Either the “native” models are transformed just for the purpose of giving them to usability-related tools, or modelling takes place with an adapted version of the models, which is transformed into a form suitable for the web engineering solution’s code generation.

Integration With Non-UML Presentation Models

The WebML and OO-H presentation models do not fulfil the requirement that all models should be UML models. Instead, they are XML documents which are used as page templates [Ceri00WebML] [Cachero00PresOOH]. Similar to these two approaches, other current and future web engineering approaches might choose a related approach and interpret the page layout and presentation as part of the implementation. (In the form of UWEXML [Kraus02Generation], even UWE features a similar, XSLT-based template mechanism, but UWEXML templates are present in addition to UWE’s presentation models.)

For web engineering solutions whose presentation models are only page template documents, there are two possibilities to provide the information from the presentational metaclass PresUiModel to tools:

- Introduce a new presentation model. Figure 5.14 shows a possible way to do this: In the package which merges the existing models and the models proposed in this work, add a

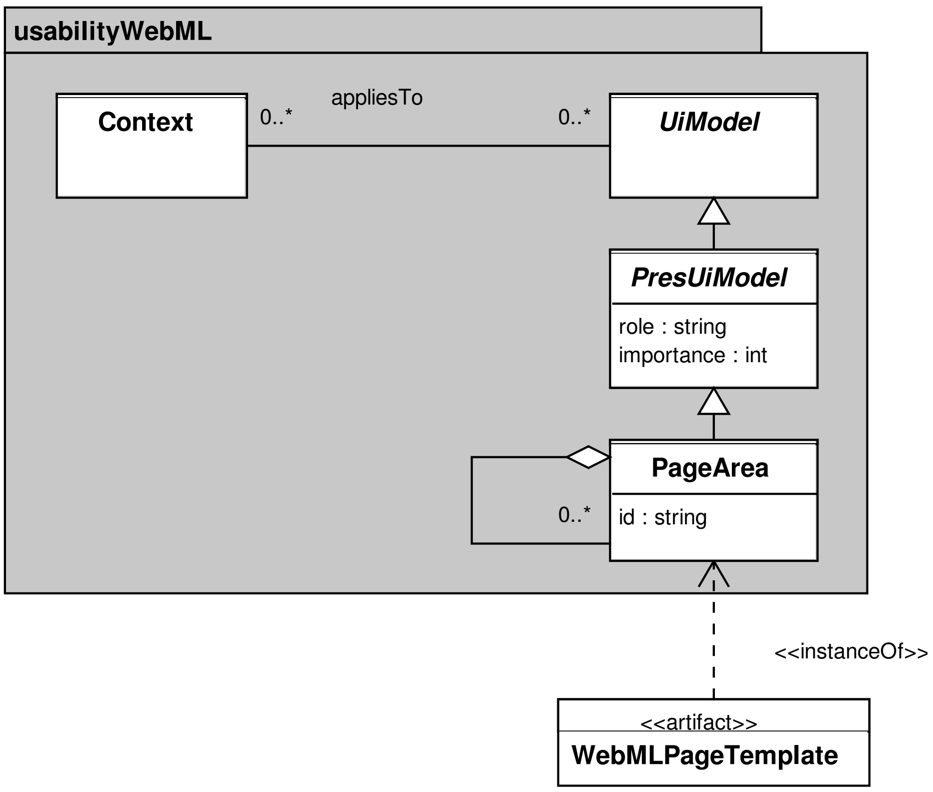


Figure 5.14: When the presentation model uses XML rather than UML, a new class PageArea is introduced. After further refinement, it becomes the XML page template.

new PresUiModel child class which does *not* have any corresponding class in the existing web engineering solution. In the figure, it is simply called PageArea. This class can represent a hierarchy of page areas, where the top-level PageArea represents an entire HTML page. The non-UML page template document can be imagined as an instance of this newly introduced simple presentation model.

- Do not include PresUiModel child class instances in the models at all, only embed information on page areas ("this page area is advertising") in the implementation. This means that the tools need to analyse not only the models, but also the final HTML code to access the relevant information. In this case, usability analysis which is based on the models *alone* (section 4.2.3) cannot determine e.g. whether the page layout is acceptable, but a tool which analyses both the models and an implementation (section 4.3.1) will not have this problem.

In the second case, the necessity for tools to access the HTML code is not a significant problem. In practice, many usability-related tools will already have to access the implementation directly for other reasons. As shown in the next section, embedding of model information has other advantages besides solving the problem that some web engineering solutions do not feature a concept that corresponds to `PresUiModel`. The section also explores different ways of adding model information to the implementation of a website.

5.4 Embedding Model Information in the Implementation

Summary: For some tool concepts, it is desirable to have access to the models of a web application together with the implementation. Many different possibilities exist to embed parts of the model in the implementation. The suggested XML-based approach is capable of representing the model extensions described in section 5.2.

Previous sections in this chapter have concentrated on the representation of context, presentational and navigational aspects in UML models. This section discusses an alternative to UML: The information from the model can be added to the implementation, either directly in its (X)HTML pages or referenced from the pages.

5.4.1 Motivation

There are several reasons why it may sometimes be convenient to embed the information in this way: As outlined above, no UML presentation model exists for some web engineering solutions, so it makes sense to add e.g. details about page areas directly in a page template. Moreover, for tools that analyse both the models and the implementation (sections 4.3.1, 4.5.1), it is sensible and desirable to make both models and implementation available in the same place, and to use the same web-based technologies for both.

Also, it is important to define a proposed standard way in which tools can gain access to the models that correspond to a particular implementation. This is necessary because unlike with other MDA tool concepts, some of the tools proposed in this thesis will not be run by the developer in his IDE, but by someone else, long after development is finished. If each tool required some configuration about where and how to find the models, not only would the effort of using the tool be increased, but there would also be practical problems in obtaining the models (e.g. because development was carried out by another company) and bringing them into the format expected by the tool.

A last reason in favour of adding the models to the implementation is that it should be possible for anonymous visitors to download model data from the website. This would allow anyone to use a tool to verify whether the site is accessible and usable, similar to the way that anyone can use a non-model based validator like TAW (page 34) on sites today. As a side effect, the public metadata of such “self-documenting” website implementations would be attractive for many other tool concepts which are not related to web quality, such as “semantic web” applications. Web engineering modelling environments should have facilities for automatic publishing of UI-related model information.

5.4.2 Alternatives for Representing Model Data

When embedding model data in the implementation of a web application, solutions must be found for different aspects, including the format used to represent information, where to store it and how to reference information from the HTML code of web pages.

Format for Model Information

As opposed to the viewable content of a website, models are machine-readable data. XML-based formats are a suitable way to represent them. If the models are diagrams, XMI (XML Metadata Interchange [OMG05XMI]) is a standardized representation for any model data which can be represented using MOF (Meta-Object Facility), including UML models. However, use of XMI results in fairly complex XML documents.

A simpler way to assign metadata to objects is RDF (Resource Description Framework, [W3C-RDF]), whose subject-predicate-object triples can express properties such as the language of a HTML document, but also more elaborate graph-structured information. RDF information can be embedded in a web page using RDFa [W3C-RDFa]. When doing so, a disadvantage is that only a single property/value pair can be assigned to any element on the page. If the page author already wants to semantically mark up the content of some elements with RDFa for another purpose, web engineering model information cannot be added for the same elements, it can only replace it.

Finally, it is also possible to use a basic XML namespace extension and to define new elements and attributes together with the semantics of their use.

Where to Store the Model Information

A model can either be embedded directly in an HTML document or it can be stored as a separate document. This separate document can be uploaded to the webspace alongside the content and then referenced from the HTML pages.

For embedding, one approach is to specify transformation rules which extract a model from the implementation's XHTML. With GRDDL [W3C-GRDDL], XSLT transformations create an RDF document from the XHTML page. This way, the RDF information can be embedded in almost any way in the document.

Some types of information are more suitable for embedding than others. For example, so-called Microformats (`http://microformats.org/`) could be used to embed the presentation model in the HTML or XHTML, using e.g. `<div class="navigation">` for the part of the markup which contains the navigation. For properties which apply to the document as a whole, the `<meta>` tag in the document head can be employed.

Finally, in some cases standards or recommendations already exist to represent model data in the document. In the case of the presentation model, the XHTML role attribute [W3C-XHTMLrole] can also be used to embed the `role` property of page areas, using markup such as `<div xhtml:role="navigation">`. For RDF data in general, triples can be represented in embedded form in an XHTML document using RDFa, which introduces new attributes for this purpose. The following markup (which must appear in an XHTML+RDFa document) is a typical example: `<div property="usability:role" content="navigation">`.

How to Reference Models

It seems appropriate for the HTML code to contain references to all models which were used directly or indirectly during its creation, as this will aid any tool to quickly access all relevant information for the HTML page at a certain URL. The models can be referenced via HTTP URLs in `<link>` tags inside the document head.

Apart from this, default implicit locations for models are imaginable. For example, for any web page URL, appending ".model" could be used as the default location for models associated with the URL. A directory-wide fallback location may also be useful, so that the models for the URL `http://example.org/dir/file.html` will be looked for in the following locations (in this order):

```
http://example.org/dir/file.html.model
http://example.org/dir/index.model
http://example.org/index.model.
```

If more accurate references to individual model elements are required – for example, references to single classes in a large file with model data – then the standard web-based mechanisms for identifying document parts can be used, such as fragment identifiers ("#abc" URL suffixes for any element with an "id="abc"" attribute), XPath [W3C-XPath] or XPointer references.

5.4.3 Proposed Method for Embedding Model Data

Below, one particular way of embedding the model data into a web application's implementation is described. A subset of it is supported by the tool Wusab from chapter 6. As outlined above, many ways exist to embed data. The proposed method was chosen as the simplest possible one while keeping the following two aims in mind: On one hand, the developer should be able to store any web engineering model in XMI format and reference it from an HTML page, which ensures that no model information is lost. On the other hand, properties for the model extensions related to usability (context, presentation and navigation model, section 5.2) should be embeddable directly in the document. This happens using new XML element attributes, which will make it easier for tools to extract the metadata. While an approach using e.g. GRDDL would have been possible, introducing new attributes serves the same purpose in a much simpler way. The use of a separate namespace ("we" for web engineering) ensures that the new attributes will not conflict with other attributes in use in the document. For brevity, only examples are shown, rather than a complete specification with an XML schema definition.

UML Models Involved in the Creation of a Page

Any UML models which were used for the creation of the web page should be referenced with a `<link rel="model"/>` tag, for example like this:

```
<link rel="model" id="we-model-navigation"
      href="http://example.org/navmodel.xmi"/>
```

By convention, the id attribute uses values with the prefixes we-model-presentation, we-model-navigation, we-model-uipattern, we-model-conceptual, we-model-use-case, we-model-context for the respective models, or the general we-model for other types of models. Multiple models can be referenced using multiple <link> tags. If there are several models of a kind, a suffix can be appended to make IDs unique, e.g. we-model-presentation-base. The id attribute is not mandatory, but it is necessary to refer to the model from the rest of the XHTML document later.

Referencing Model Classes From the Implementation

Tools which interpret the models need a way to associate the final implementation (XHTML pages or parts of pages) with concepts in the model. The approach for this is to reference the XMI file which contains the model, and then to identify the exact element within the XMI document from which an element in the implementation was created. Two new attributes in the "we" namespace are used for this, they can be added to any XHTML element: we:model is an XPath [W3C-XPath] expression which is executed on the current XHTML document and which yields the URL of the XMI model representation. we:path is another XPath expression which identifies an element inside that XMI document.

As an example, we assume there is a shopping cart on the web page. It corresponds to the ShoppingCart class in the navigation model `http://example.org/navmodel.xmi` from above. The opening <div> tag of the cart's XHTML representation could then read as follows:

```
<div we:model="//*[@id='we-navigation-model']@href"
     we:path="//UML:Class[@name='LoginPage']">
```

The expression in we:model can also directly contain the model URL. Because an XPath string literal is contained inside the attribute value, the string is double-quoted in this case, e.g. we:model="'`http://example.org/navmodel.xmi`'". If a model class corresponds to an entire XHTML page, the we:model and we:path attributes are added to the page's <html> element.

Context Information

The context information is especially relevant to tool support for issues related to usability and accessibility. For this reason, a simple XML-based description of the data is preferable to the more heavyweight XMI format. Apart from being referenced via the <link> tag of a document (see above), it is also possible to embed the context data directly in XHTML documents. A simple context model with two user groups and one platform is outlined in figure 5.15 – the properties of the different classes (figure 5.3 ff.) are directly turned into analogous XML constructs.

Presentation Model

Similar to the context model, an XML-based representation is also used for the presentation model. However, the role and importance values of the PresUiModel metaclass as well as the appliesTo association it inherits from UiModel (figure 5.8) are embedded in an existing XHTML

```
<model-context xmlns="http://example.org/...">
  <context id="GeneralPublic">
    <importance>1</importance>
    <userref href="#InternetVisitor"/>
    <userref href="#Employee"/>
    <platformref href="PC"/>
  </context>
  <user id="InternetVisitor">
    <maleRatio>50</maleRatio>
    <femaleRatio>50</femaleRatio>
    <unspecifiedSexRatio>0</unspecifiedSexRatio>
    <age>35.0</age>
    <ageDeviation>7.0</ageDeviation>
    ...
  </user>
  <user id="Employee">...</user>
  <platform id="PC">
    <deviceDescription>PC</deviceDescription>
    <supportedAbility>see-colour hear see-movement
    see-blinking input-text select point</supportedAbility>
    ...
  </platform>
  ...
</model-context>
```

Figure 5.15: Example for the XML representation of the context model from section 5.2. Compared to an XMI-based format, this format is more readable and shorter.

page rather than being isolated in a separate XML document or an embedded section in a document. The following new attributes (which are all defined inside the "we" XML namespace) can be present for an entire HTML page (if added to its <html> tag) or any element within the page, such as a <div> which contains the navigation menu:

we:role contains a space-separated list of role tags (see page 104), we:importance an integer value with the relative importance of the screen area relative to other areas on the page, and the value of we:appliesto is a space-separated list of URLs, each of which should include a fragment identifier which references the exact <context-model> element which applies to the page area. In most cases, we:appliesto will only be present in the <html> tag, i.e. a context will apply to an entire web page.

Navigation Model

The information in the usability classes related to navigation, NavUiModel and UiPattern (see section 5.2.6 above), is attached to individual pages in the implementation. For this reason, it is also suited for direct embedding in the pages. This is achieved by adding it to the html element in the form of attributes. In the case of the we:appliesto attribute, a conflict with the attribute of the same name from the presentation model is possible. However, for any correctly modelled application, the set of contexts will be identical for the presentation and navigation models. The we:purpose attribute contains the list of tags which describe the goals that should be achievable on the page, and we:pageimportance specifies an integer which gives the relative importance of the page compared to other pages on the website.

In contrast to the associated context and the purpose/importance of the page, navigation patterns include several pages, so the relevant information should be put in a separate XML model document. Referenced via a <link rel="model" id="we-model-uipattern" href="..."/> tag in the head of each document which is part of a pattern, the XML file describes the pattern and in turn references all its pages, for example as follows:

```
<model-uipattern xmlns="http://example.org/...">
  <uipattern>
    <type>guided-tour</type>
    <page href="http://example.org/page1"/>
    <page href="http://example.org/page2"/>
    ...
  </uipattern>
</model-uipattern>
```

Dynamically Generated Web Applications

In some situations such as with AJAX (Asynchronous JavaScript and XML, [Garret05AJAX]) applications, it is possible that several NavUiModel classes only result in the creation of a single HTML page. If that single page contains several different layers which are displayed alternately depending on the state of the application, embedding the model information as outlined above will no longer work. A case is even imaginable where all user interface elements are created at runtime with JavaScript. This does not make it possible to embed the model data in the final XHTML, because the XHTML document tree is not stored on the server, but only created when the browser executes the JavaScript.

The suggested workaround for this situation depends on the requirements for the web application: If the application is intended to be usable not only by sighted users who employ a JavaScript-capable device, but also by devices without JavaScript support, or by people who require non-visual rendering, then it will be necessary to provide a separate, non-AJAX fallback version of the website. The embedded model information can be added to the different HTML pages of this fallback version.

If, however, the website is only required to be usable by sighted users with browsers which are able to execute and render AJAX-based user interfaces, the MDA code generation process can create a set of static HTML documents which mimics the look and feel of the dynamic pages. These pages can contain the embedded model information for analysis by tools.

In either case, the embedded model will not include data about the special user interface conventions and techniques which are used in rich Internet applications. This will prevent any model-based tools from being able to recognize usability problems. As an example, it is feasible to implement an online shop in which drag & drop must be used to put items in the shopping cart. The resulting usability problems cannot be detected by the tool concepts introduced in this thesis: The users do not expect to use drag & drop on web pages, so they will have problems putting items in the cart. As detailed modelling concepts for dynamic web applications emerge [Bozzon06ConceptualRIA], the model concepts in this section will have to be adapted and extended. Meanwhile, any tool which offers usability support for RIAs can access the web engineering solution's UML models directly (if they are referenced via a `<link>` element) to extract any information that is not present in the XML-based representation.

Constraints

Any constraints which were added to pages can also be included in the implementation. This happens with the `we:constraint` attribute, which contains a string with an OCL constraint that should be met in the context of the element to which the attribute is added. Again, if an entire page should fulfil a constraint, the attribute should be added to the page's `<html>` element.

It may not make sense in all cases to embed OCL constraints in the HTML page: In order to interpret arbitrary constraints, a tool will have to access the UML model which is associated with the page. However, the constraint is already present in that UML model along with the rest of the model information. Still, certain tools may choose to support only a limited subset of constraints, and in doing so may avoid the need to access the UML model. In particular, the examples mentioned in section 5.2.6 could be implemented by a tool which simply assumes the presence and semantics of the relevant methods, such as `downloadTime()` for the time it takes to download the page.

5.5 Conclusion

This chapter describes a generic way to extend the models of existing web engineering solutions. The aim of the extensions is to allow the models to express all the information that is necessary to support the tool concepts from chapter 4. In the context of this work, the basic problem of existing models (as examined in section 3.2) is that certain information, which is important for usability and accessibility, is absent from the models. Above all, this includes a context model which describes the circumstances under which the web application is expected to be used. The proposed context model includes properties of the user (such as his knowledge, abilities and goals), the technical platform (hardware, browser and network access) used to access the application, and the environment of use. Minor extensions are also required for the navigation and presentation models of common web engineering solutions.

Context modelling has already been the topic of related work, but primarily outside the area of web engineering. The models from related efforts (in particular UsiXML) were adapted to the needs of usability tool support. Two requirements were central during this process: The models should be generic in the sense that it should be possible to add them as extensions to as many existing web engineering modelling solutions as possible. Also, they should not be too expressive and complicated, since this would make them too complex for humans to use efficiently, and for tools to interpret. The proposed model extensions fulfil these goals as much as possible. However, compromises are sometimes necessary for this. For instance, model transformations may be the only way to apply the extensions to some less well known web engineering solutions, and the relative simplicity of the models may require the model to be further extended by the developer for projects with highly specialized needs.

Finally, this chapter tries to support interoperability between modelling solutions and usability-related tools not only by providing a model extension common to all existing web engineering approaches, but also by defining a way to store model information alongside or embedded in the implementation. This way, any tool which is only given the URL of an implementation can have straightforward access to the models, regardless of the environment under which the models were created.

Chapter 6

Wusab, a Tool for Automatic Model-Based Usability Analysis

Together with the following chapter, this chapter provides more details about the tool concepts presented in chapter 4, providing prototype implementations and a comparison with related existent tools. In this chapter, the prototype “Wusab” is presented, a tool for the analysis of web engineering models together with the corresponding web pages. The tool can identify many usability and accessibility problems of a website automatically during several stages of development, and give feedback to the developer or site maintainer. It provides a basis and framework for the following tool concepts:

- Design phase, section 4.2.3: Continual analysis of web engineering models. The developer gets notified by the tool when he makes a problematic change to the models, which has the potential to eliminate some usability problems very early in the development process, when doing so is still cheap.

- Implementation phase, section 4.3.1: Analysis of models together with the HTML pages that implement the web application. This promises higher quality of results than with existing accessibility checkers, which only analyse the implementation.

- Maintenance phase, section 4.5.1: Analysis of models together with the implementation and usage statistics of the live website. The tool can identify changes happening after the launch of the site which might affect its usability, or verify whether constraints (e.g. regarding the site performance) are met.

For the Wusab prototype, only a few example tests were implemented, mostly for analysis of models together with the implementation (second point above). Existing usability and accessibility validators only take the HTML code of a page into account when they create a report of possible problems. In contrast to this, when a web engineering method has been used to create a website, additional information is available in the form of the navigation and presentation models which describe the site. Using this information, an automated validator can verify usability guidelines (such as “text on the web page should be easy to understand by the target audience”) with higher confidence. It can also perform automatic validation in cases where existent tools only output instructions for manual inspection by the developer. The flexible approach outlined in the previous chapter allows this model to be used by validators running inside an integrated development environment, but also at a later time, without access to the development environment. Access to the models is possible either directly (in the development IDE), via references to the UML models in XMI format which are inserted into the HTML pages, or by directly embedding some models in the implementation. The prototype implements the latter possibility. The Wusab prototype is described in [Atterer05ICWE] and [Atterer06JWE].

6.1 Related Work

Summary: Automated critique of models alone has been the focus of previous work. Also, many accessibility and usability validators for web pages exist. However, Wusab is the only tool which uses web engineering models to improve the quality of automated web page usability evaluation.

Automated Model Analysis and Critique

As outlined above, one possible variant of the idea of model-based usability analysis is that the tool only looks at the models and then warns about potential problems or provides suggestions on how to improve the models. While this approach has not yet been applied to web engineering models in the context of usability, similar tools exist for qualitative model analysis in other areas.

The ArgoUML modelling tool ([Robbins99Cognitive], `http://argouml.tigris.org/`) includes a facility for model critique: The so-called "design critics" are small programs that monitor changes made to the models. If they identify a potential problem, they add it to a list which is displayed in the user interface. Additionally, many critics also assist in correcting the deficit that they highlighted.

In UWE's adapted version of the program, ArgoUWE (section 3.2.1), the respective framework is used to suggest and correct problems in UWE models [Knapp04ArgoUWE]. The focus of implemented tests is on correctness and completeness of models (e.g. each process use case must show a corresponding process node), rather than on the quality of the final web pages.

The RIDE tool (rule-based inconsistency detection engine, [Liu02RuleBased]) follows the same idea, also with general inconsistency checking for UML models in mind. However, both ArgoUWE's critics and RIDE could probably be adapted to include checks for UML-based web engineering models, such as verifying that the presentation model contains all elements (search, help etc.) that are usually required for a website.

The idea of using a model to automatically identify usability problems during development is also described in [Kieras96GOMS, section 5] in the context of GOMS models (goals, operators, methods, selection rules) for desktop GUIs. These models describe the sequence of actions that a user is expected to take to achieve a certain goal. For example, to delete a file, it is necessary to select the file, open a menu and select the "Delete" entry. The author suggests a number of automatic tests which can be applied to more complex GOMS models, such as a way to estimate the time required by users to complete a certain goal. This includes a test for consistency (do related goals require similar actions?), a test for efficiency (can frequently used goals be reached quickly?) and a way to estimate the time required by users to complete a goal. As a GOMS model is a formal, exact description of the task, this kind of analysis is relatively easy. In comparison, web engineering models are typically more abstract, and many usability guidelines are more vague (in the sense that no algorithm and at most a heuristic can be formulated), making it harder to automate testing.

Usability and Accessibility Analysis of Web Pages

Several tools exist which perform an analysis of web pages with regard to accessibility problems. The analysis only works with the HTML of a web page or all the pages of a site, not with web engineering models. Typically, the tools are given the URL of a page to download and then automatically analyse the site before presenting the results in a report. Section 2.3.4 (page 31) has already given an overview of the different types of tools. This section attempts to cover most tools in this area. For each tool, the following information is included:

- *General information:* Whether the tool is academic or commercial, a non-functional prototype only, publically available, and whether it is e.g. a desktop GUI application or a web service.
- *Type of analysis:* How conclusions about usability are made; usually using heuristic algorithms for individual guidelines.
- *Web quality standard:* What rules are the basis of the analysis, or what "measurement" is applied to improve usability or accessibility.

	Analysis type	Web quality standard	Run-time extensibility	Input	Output	Interaction
LIFT	heuristics	accessibility	no	page	report	no
Bobby	heuristics	accessibility	no	page	report+ann.	no
WebXACT	heuristics	acc., privacy, content	no	site	reports	no
TAW	heuristics	accessibility	no	page	report+ann.	no
WAVE	heuristics	accessibility	no	page	annotations	no
WebTango	statistics	reference sites (a.+u.)	n/a	site	report?	no
Kwaresmi	heuristics	accessibility	yes	page	report	no
MAGENTA	heuristics	accessibility	yes	page	report	repair
EvalIris	heuristics	accessibility	yes	page?	report	no
ATRC	heuristics	accessibility	no	page	report+EARL	avoid false pos.
Wusab	heuristics	accessibility, usability	no	page, **models**	report	no

Table 6.1: Comparison of tools for automated web page accessibility/usability validation. In the available literature, only "Wusab" uses web engineering models during the validation.

- *Extensibility:* Whether and how the rulesets can be extended at runtime by the user.
- *Input:* The sources of information available to the tool – a single page (only one HTML page is analysed), many pages (an entire website or a part of it is examined), possibly others such as user input or models.
- *Output:* Format and content of the tool's report, possibly embedded in the input page or in a machine-readable format, possibly with suggestions on how to correct errors.
- *Interaction:* Whether the user can interact with the program while the analysis takes place. Either "none" (fully automatic analysis) or aspects like program-assisted repair of problems.

In the following list, it should be noted that none of the existing tools that were found includes analysis of web engineering models. For comparison, the Wusab prototype which does interpret model data is included at the end. Table 6.1 gives an overview of the properties of the different validators.

LIFT (`http://www.usablenet.com/`): Commercial, web-based service with a free demo version, repair functionality in additional desktop-based software, available until 2006,[1] good coverage of the WCAG and Section 508 guidelines.

- *Type of analysis:* Heuristics for guidelines
- *Web quality standard:* Accessibility guidelines: Section 508, WCAG 1
- *Extensibility:* Not at runtime
- *Input:* Single HTML page or multiple pages (which are analysed individually)
- *Output:* HTML-based report with suggestions on improvements
- *Interaction:* Marking false positives, interactive repair (via desktop-based tools)

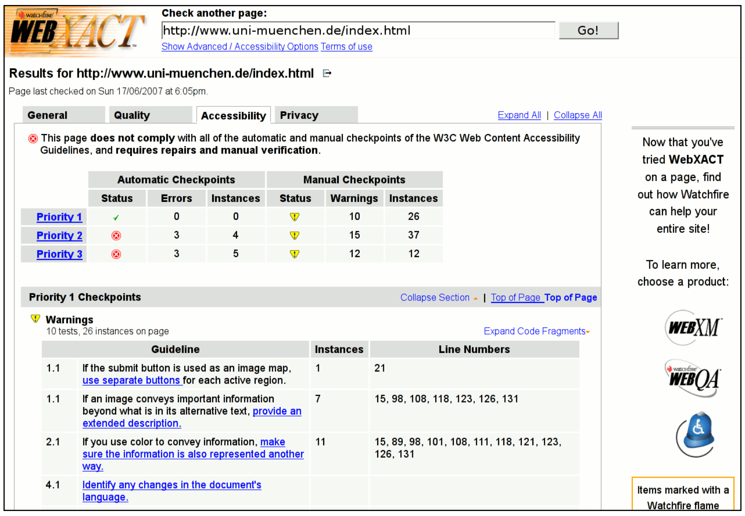
Bobby (`http://bobby.watchfire.com`): Commercial but free web-based service, available until April 2005 when it was made part of the commercial WebXACT.[2] One of the first validators to provide good coverage of the W3C WCAG.

- *Type of analysis:* Heuristics for guidelines
- *Web quality standard:* Accessibility guidelines: Section 508, WCAG 1
- *Extensibility:* Not at runtime
- *Input:* Single HTML page
- *Output:* HTML-based report, (in later versions) annotated version of original page
- *Interaction:* None

WebXACT (`http://webxact.watchfire.com`, see figure 6.2): Commercial service, with a basic free web-based version as well as a paid web-based functionality and desktop-based analysis. The number of tests is extensive, including not only accessibility, but also basic quality properties of the website content (e.g. spelling errors, click path depth, broken links), security and privacy issues.

[1]An HTML page with product information from 2005 is available via the Internet Archive:
`http://web.archive.org/web/20051226075307/www.usablenet.com/products_services/lift_machine/lift_machine.html`

[2]An HTML page with information from 2005 is available via the Internet Archive:
`http://web.archive.org/web/20050212011552/bobby.watchfire.com/bobby/html/en/index.jsp`



Check another page:

http://www.uni-muenchen.de/index.html

Results for http://www.uni-muenchen.de/index.html

Page last checked on Sun 17/06/2007 at 6:05pm.

This page **does not comply** with all of the automatic and manual checkpoints of the W3C Web Content Accessibility Guidelines, and **requires repairs and manual verification**.

	Automatic Checkpoints			Manual Checkpoints		
	Status	Errors	Instances	Status	Warnings	Instances
Priority 1	✓	0	0	⚠	10	26
Priority 2	⊗	3	4	⚠	15	37
Priority 3	⊗	3	5	⚠	12	12

Priority 1 Checkpoints Collapse Section | Top of Page Top of Page

Warnings
10 tests, 26 instances on page
Expand Code Fragments

Guideline		Instances	Line Numbers
1.1	If the submit button is used as an image map, use separate buttons for each active region.	1	21
1.1	If an image conveys important information beyond what is in its alternative text, provide an extended description.	7	15, 98, 108, 118, 123, 126, 131
2.1	If you use color to convey information, make sure the information is also represented another way.	11	15, 89, 98, 101, 108, 111, 118, 121, 123, 126, 131
4.1	Identify any changes in the document's language.		

Now that you've tried **WebXACT** on a page, find out how Watchfire can help your entire site!

To learn more, choose a product:

Items marked with a Watchfire flame

Figure 6.2: The WebXACT analysis tool examines a web page or entire website with regard to accessibility problems, but also other web quality issues and privacy regulations.

- *Type of analysis:* Heuristics for guidelines
- *Web quality standard:* Section 508, WCAG 1 for accessibility, additional heuristics
- *Extensibility:* Not at runtime
- *Input:* Multiple HTML pages
- *Output:* HTML-based report
- *Interaction:* None

TAW (test accesibilidad web, `http://www.tawdis.net`): Non-commercial tool, publically available as a web service and a stand-alone Java application, funded by the Spanish government, good coverage of the WCAG.

- *Type of analysis:* Heuristics for guidelines
- *Web quality standard:* Accessibility guidelines: WCAG 1
- *Extensibility:* Not at runtime
- *Input:* Single HTML page
- *Output:* Annotated version of original page followed by report (figure 2.12, page 34)
- *Interaction:* None

WAVE (`http://wave.webaim.org`, [Kasday00WAVE]): Non-commercial, publically available web-based tool with academic funding, good coverage of accessibility guidelines.

- *Type of analysis:* Heuristics for guidelines
- *Web quality standard:* Accessibility guidelines: WCAG 1, Section 508
- *Extensibility:* Not at runtime
- *Input:* Single HTML page
- *Output:* Annotated version of original page
- *Interaction:* None

WebTango [Ivory01Metrics], [Ivory01PhD]: Proof-of-concept academic prototype, not publically available. Special approach: Computes many design metrics (e.g. number of links, colours or text paragraphs) for websites, then allows the web developer to compare his own site's metrics with those of a number of sites which are known to have excellent usability.

- *Type of analysis:* Statistical correlation of the tested site's features with those of "known-good" sites
- *Web quality standard:* Many real-life sites, including nominees for the Webby Awards 2000 [Ivory01Metrics]
- *Extensibility:* Not at runtime (not applicable)
- *Input:* Many HTML pages of a website
- *Output:* Comparison of the tested site with others *(No details available)*
- *Interaction:* Probably none *(No details available)*

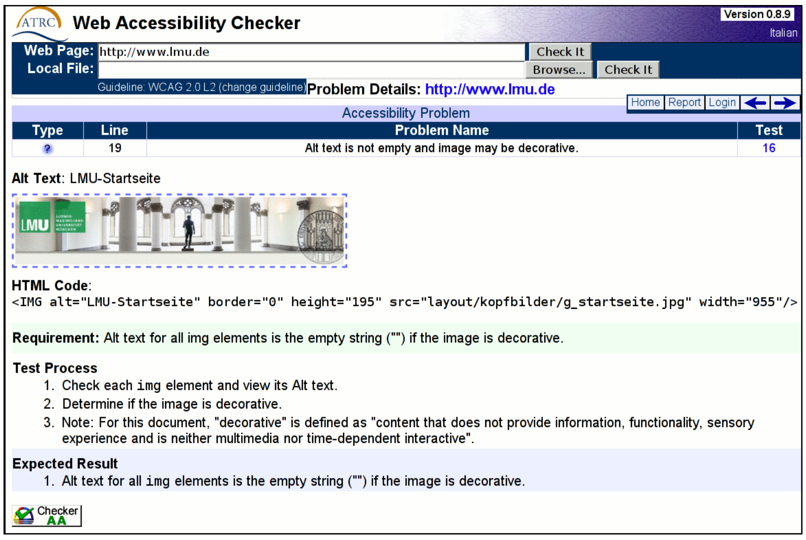
Kwaresmi [Beirekdar02Kwaresmi]: Proof-of-concept academic prototype, desktop GUI application, not publically available. Special focus is the high-level description of guidelines with a special language, their dynamic selection during evaluation. No significant number of tests is implemented.

- *Type of analysis:* Heuristics for guidelines
- *Web quality standard:* Intended for WCAG
- *Extensibility:* Guidelines are extensible and replaceable, special guideline definition language (GDL).
- *Input:* Single HTML page
- *Output:* Simple text-based report
- *Interaction:* None (except for adaptation of guidelines)

MAGENTA (`http://giove.isti.cnr.it/accessibility/magenta/`, successor of a similar tool called NAUTICUS): Non-commercial desktop GUI tool with academic funding, not publically available, fair number of tests for accessibility guidelines. The focus is on the usability of the tool as well as the analysis results, e.g. in the form of clear reports and interactive repair of problems.

- *Type of analysis:* Heuristics for guidelines
- *Web quality standard:* Accessibility guidelines: WCAG 1, Stanca act and guidelines for the visually impaired
- *Extensibility:* Guidelines are extensible and replaceable, special guideline abstraction language (GAL).
- *Input:* Single HTML page
- *Output:* Text-based report, integrated user interface for corrections
- *Interaction:* The program assists the user in the correction of any identified accessibility problems.

EvalIris ([Abascal03EvalIris], [Abascal04EvalIris]): Proof-of-concept academic prototype, web-based service, not publically available. Defines an XML schema for the description of guidelines.



Web Accessibility Checker

Version 0.8.9

Italian

Web Page: http://www.lmu.de Check It

Local File: Browse... Check It

Guideline: WCAG 2.0 L2 (change guideline)

Problem Details: http://www.lmu.de

Accessibility Problem

Type	Line	Problem Name	Test
?	19	Alt text is not empty and image may be decorative.	16

Alt Text: LMU-Startseite

HTML Code:

```
<IMG alt="LMU-Startseite" border="0" height="195" src="layout/kopfbilder/g_startseite.jpg" width="955"/>
```

Requirement: Alt text for all img elements is the empty string ("") if the image is decorative.

Test Process

1. Check each `img` element and view its Alt text.
2. Determine if the image is decorative.
3. Note: For this document, "decorative" is defined as "content that does not provide information, functionality, sensory experience and is neither multimedia nor time-dependent interactive".

Expected Result

1. Alt text for all `img` elements is the empty string ("") if the image is decorative.

Figure 6.3: Part of a report generated by ATRC. Because abstract information is not available, the user needs to check some guidelines manually, e.g. whether images are decorative.

- *Type of analysis:* Heuristics for guidelines
- *Web quality standard:* Intended for WCAG, Section 508
- *Extensibility:* XML-based description of guidelines
- *Input:* Probably single HTML page *(No details available)*
- *Output:* HTML-based report
- *Interaction:* None

ATRC Web Accessibility Checker (`http://checker.atrc.utoronto.ca`, figure 6.3): Non-commercial, publically available web-based tool with academic funding, very good coverage of accessibility guidelines. Successor to a desktop GUI-based tool called A-Prompt.

- *Type of analysis:* Heuristics for guidelines
- *Web quality standard:* Accessibility guidelines: WCAG 1, WCAG 2, BITV, Section 508, Stanca act
- *Extensibility:* Not at runtime, but uses modularised approach
- *Input:* Single HTML page
- *Output:* HTML-based report, or machine-readable report in EARL format [W3C-EARL]
- *Interaction:* Can ask the user questions about a specific suspected problem ("Does the anchor contain text that identif[ies] the link destination?") rather than outputting a warning for every instance of the problem, which can help to reduce the number of reported false positives.

Wusab ([Atterer05ICWE], [Atterer06JWE]): Proof-of-concept academic prototype, web-based service, not publically available.

- *Type of analysis:* Heuristics for guidelines

- *Web quality standard:* Intended for accessibility guidelines (e.g. WCAG) as well as usability guidelines (Web Style Guide and others)
- *Extensibility:* Not at runtime
- *Input:* Single HTML page, web engineering model
- *Output:* HTML-based report
- *Interaction:* None

In conclusion, automatic validation of accessibility issues in websites has been the topic of related work. This also resulted in the creation of a number of prototypes, of systems which are fit for productive use, and even systems which have been turned into commercial products. Depending on the tool, different aspects of the validation process have received more attention than others: Either a complete coverage of WCAG has been the tool's primary goal (e.g. with the ATRC checker), extensibility concepts were evaluated (e.g. Kwaresmi), or automatic repair of problems was the tool authors' main focus (e.g. MAGENTA).

Wusab represents a unique approach in the sense that it is the only web quality validation tool which is able to extract information from web engineering models. As shown in the next section, this approach has the potential to allow for much higher-quality and more accurate reports of possible problems with a website.

Unlike some of the above tools, the Wusab prototype does not attempt to provide a complete accessibility and usability validation solution which is immediately useful for development work on real-life web applications. Instead, it is a proof-of-concept tool which is intended to show a new approach for improving validation report quality. To this end, an implementation only exists for a few selected usability guidelines which are best suited for demonstrating the benefits of the new concept. As an area of future work, this implementation could be extended to include better coverage of guidelines and better integration with modelling environments.

6.2 Improving Automated Usability Evaluation Using Models

Summary: Usability guidelines are often formulated in an abstract way which makes automatic validation difficult. A number of examples shows that by including model information in the analysis, checks can potentially be more accurate and reliable, and can operate at a more abstract level.

6.2.1 Problem: Many Usability Guidelines are Abstract or Vague

Sources of guidelines like the W3C's Web Content Accessibility Guidelines, the Web Style Guide or Research-Based Web Design & Usability Guidelines (section 2.2) contain many different guidelines which describe desired properties of a a website or web application at various different levels of abstraction. How "subjectively" a guideline is formulated has a direct influence on how easy it is for a validation program to provide an implementation for it. As an example, we consider a number of guidelines (based on [OperaSmallDevices], [Lynch02WebStyleGuide], [NielsenAlertbox, 31 January 2005]). They are concerned with the text content of a particular web page and cover the entire range from "very concrete" to "very abstract":

1. *A page should not contain more than 10000 printed characters.*
   A guideline like this might be required if the target platform of a website includes older mobile phones which have tight constraints on main memory usage. Writing an algorithm to count the characters is straightforward.

2. *The text on the page should use correct spelling and grammar.*
   The validator can use a word list to check for incorrectly spelled words. With appropriate algorithms, many grammar errors can also be detected. All in all, the tests are fairly easy to implement. However, the following should be noted: Even though the guideline is still quite concrete, it is already no longer possible for a validation tool to say with certainty that a particular page fulfils the criterium. Instead, it can only prove that the page contains a mistake, and even this only with a high confidence rather than absolute certainty.

3. *The page content should be easy to scan.*
   At this point, a validator leaves the area of fixed rules altogether and is dependent on heuristics alone. Based on research results and expert opinion, an algorithm can be formulated which counts the length of sentences, the ratio of text paragraphs to bullet lists or headings etc., and outputs a probability (but not anything more definite) that the page conforms to or violates the guideline. Knowledge about the users can help to improve accuracy – for instance, teenagers read web pages differently than adults. Even in the case that the accuracy of the heuristic is high, an additional problem is that the guideline itself is not applicable all the time; a good reason may exist to ignore it for a particular page.

4. *The page content should be understandable.*
   Even if a tool has extensive knowledge about the users of the web application, it is difficult for it to make any guess with regard to how easy it will be for these users to understand the page content. A complete analysis of the semantics of text sentences, followed by an estimate of whether the user group will understand it, is not possible algorithmically today. At most, a test implemented in a validation tool can "scratch at the surface" of this type of analysis, e.g. by highlighting single complicated words in the text that the user group cannot possibly understand at their age or with their education.

5. *The text on the page should not be offensive to the user.*
   It is often unpredictable even for humans whether a person will take offense at a certain text or not. Similarly, a usability validation tool will have great problems even to make an educated guess in this respect. In practice, this type of guideline cannot be implemented most of the time, or the effort of its implementation would not justify the benefits. The difference to the previous point is that properties of a user *group* were sufficient above, whereas here knowledge about a *particular person* is required to estimate whether they will take offense.

The above set of examples is typical for many guidelines: Invariably, the more vague a guideline is (“vague” from the point of view of someone who needs to implement a heuristic algorithm for it), the more it involves the final users of the web application. They differ with regard to their particular background, experience and needs, but also with regard to the type of device they use. Based on the overall analysis of guidelines in section 2.2, it thus makes sense for a validator to include information about these factors in its analysis. In the case of Wusab, web engineering models were chosen as the source of this information.

Most existing accessibility and usability validators for web pages only cover at most the first two points above. Wusab attempts to go beyond this: The prototype implementation also includes tests which can be thought of as belonging to the third point, and some simple aspects of the fourth point.

For complete testing coverage of usability at all of the above levels, automatic analysis is not the right approach. Instead, usability tests with real users need to be conducted. Chapter 7 introduces tool support for this task.

6.2.2 Potential Benefits of Models as a Source of Meta Information

Section 2.3.4 (page 32) has already introduced the categorization of usability validator report quality into *completeness*, *correctness* and *specificity* by Brajnik [Brajnik04Comparing]. When trying to determine the benefits of model-based accessibility and usability validation compared to existent validators which only examine the running website, it is interesting to have a look at the potential effects of the meta information in the models on these three quality aspects. Any change in the usability report of a tool will always influence all three quality measures, so the following paragraphs compare certain properties of model-based checks to those of non-model-based ones, and discuss their effect on completeness, correctness and specificity.

Greater Accuracy of Checks

In many cases, the fact that models are available allows a better differentiation between “sub-cases” of certain guideline tests: Where non-model-based validators only contain one simple check, the model-based variant can distinguish several variants of the usability mistake.

A good example for a guideline which can be improved in this way has already been mentioned earlier: *Images should have alternative text, except when they are ornamental* [Lynch02WebStyleGuide, chapter 7]. Existing validators usually only contain a check which reports all cases where the alt attribute is missing or its text is empty. By looking at the presentation model, a model-based validator can correctly identify ornamental images, whose alternative text should be empty.

The greater accuracy of the evaluation algorithm increases the specificity of the validator, as a greater number of distinct accessibility issues can be detected with the above model-based test. Furthermore, the test reduces the number of false positives (non-model-based tools will incorrectly flag empty alt text of ornamental images as an error) as well as the number of false negatives (non-model-based tools will incorrectly assume that non-empty alt text of ornamental images is OK), so both the correctness and completeness of the model-based tool are improved.

More Checks Become Possible

With access to the models, it is feasible to implement validation algorithms for guidelines which are very difficult or impossible to validate for non-model-based programs. This increases the completeness of the model-based validator, as the number of false negatives in its report is reduced.

As an example, this guideline can be considered: *Each web page should contain items like a menu, link to the homepage, search facility, modification date…* [Lynch02WebStyleGuide, chapter 2]. With some effort, it may be possible for a non-model-based tool to interpret the different parts of the page markup and to come to a conclusion whether it is likely that the guideline was followed. However, depending on how the implementation provides the different required types of functionality, the tool's result may not be very reliable. In contrast, the same check is easy to implement for a validator that has access to the presentation model from section 5.2.5, as the role attribute of classes in the model directly describe the purpose of each part of the page.

The fact that it becomes possible to implement more tests with a model-based tool also has another effect: Existent non-model-based validators often output "default advice" in all the reports they generate. For instance, WebXACT always outputs the following: "Use the simplest and most straightforward language that is possible." While it is a good idea to educate the developer in the area of usability in this way (see section 4.6), these default messages can actually be interpreted as false positives if the problem they warn about is not really present on the validated website. A model-based heuristic can eliminate a message like the one above if the probability is high that a user of the specified age and education will be able to cope with the complexity of sentences on the page. Because this reduces the number of false positives, it increases the model-based validator's correctness.

Fewer Checks May be Necessary

Model-based validators do not only allow tests for more guidelines to be implemented, they also allow fewer tests to be run when a particular web page is evaluated. The reason is that non-model-based tools do not know anything about the user groups that a web application targets, so they must take a conservative approach and assume that users may be blind or otherwise disabled, or that they may use any device or browser in existence to access the page.

In contrast to this, a model-based validation tool can disable certain tests because they only apply to user groups which are not part of the site's target audience. For instance, if the web engineering models specify that the target audience solely consists of desktop users and not of users with mobile devices, it is not necessary for the page layout to adjust itself to small displays. As a consequence, warnings that the minimum page layout width is too large could be interpreted as false positives in this case. By reducing these false positives, a model-based tool can increase its correctness compared to tools which do not have access to models.

Higher Abstraction Level Possible for Tests

Web engineering models describe the website at a more abstract level than its implementation, revealing the web developer's intentions about the different parts of the user interface. This also allows a model-based tool to implement new and existing tests for guidelines with a higher level of abstraction than that of its non-model-based competitors. As already described above, the introduction of new abstract tests will increase the tools completeness, whereas a more abstract implementation of an existing guideline check is likely to increase the tests accuracy and thus the number of false positives or negatives.

An example for a test which happens at a more abstract level than that of non-model-based validators is validation of patterns: Based on the information on web user interface patterns in the navigation model (section 5.2.6), a tool can verify whether all patterns in the web application have been implemented in a way which reflects current best practice.

In conclusion, there is evidence that using models for accessibility and usability validation has advantages at a conceptual level. These advantages can potentially lead to better quality of the report that a model-based validator outputs, since the report includes fewer false positives and false negatives and because a greater variety of possible problems can be identified.

6.2.3 Examples for New and Improved Model-Based Usability Guideline Checks

In this section, a catalogue of examples for model-based accessibility and usability checks is presented. It includes the guidelines and corresponding algorithms or heuristics which have been cited as examples in previous sections. Only guidelines which can rely on or benefit from model input are shown – existing tests of non-model-based validators are excluded. Furthermore, the list below concentrates on examples where the additional meta information from the models is particularly useful, leading to guideline test implementations which promise to be noticeably more accurate and less error-prone than in the case that no model information is available.

This list of guidelines is by no means exhaustive – it is possible to come up with many more tests that are feasible: When examining the tests performed by state-of-the-art validators which are not model-based, many opportunities for improvements through model information become apparent. Further sources of inspiration are the existing guideline documents, especially when a look is taken at those guidelines that are not currently implemented by tools. Also, the properties which are stored in the models provide the basis for a number of additional tests. During the preparation of this section, it was possible to come up with several dozen ideas for model-based guidelines in a short time. However, it should be noted that implementing these tests is often not trivial, as the required heuristics will have to be fine-tuned using a number of real-world example websites to ensure that their accuracy is adequate.

In the following paragraphs, it is often assumed that the web page content is to be represented *visually*. If the user model (section 5.2.2) specifies that the content will be consumed exclusively with non-visual user agents, some of the guideline checks (in particular those related to presentation) should not be enabled. The wording of the guidelines is not taken from the sources verbatim, it only summarizes their ideas.

Presentational Aspects

Guideline: *The maximum recommended width of page content is x pixels.*
[Lynch02WebStyleGuide, chapter 4], [NielsenAlertbox, 31 July 2006]

If the content of the page is too wide, the rightmost part will be cut off on devices whose display resolution is too low. If the validator knows about the devices that will typically be used for viewing the site, it can select the right value for *x*, render the page and check whether its width fits.

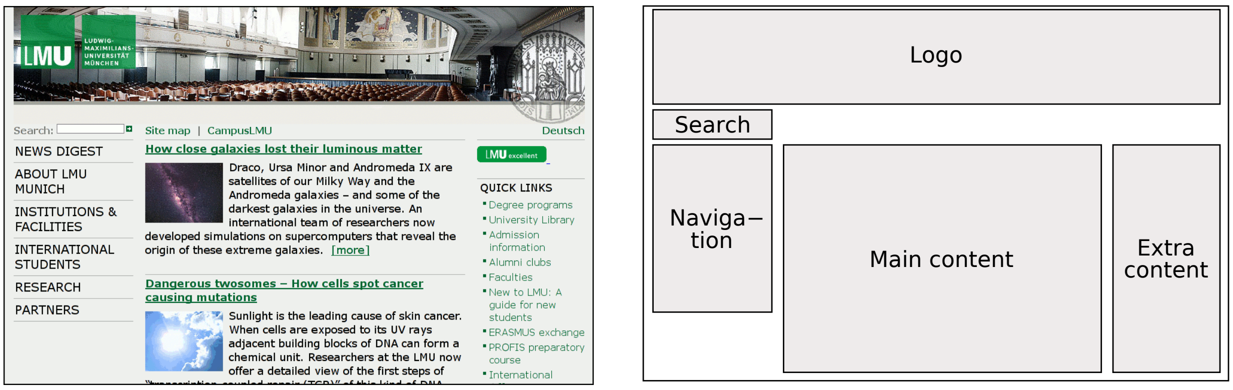
Due to the increasing variety especially of mobile devices, it is difficult to find a content width which is suitable for all screen sizes. A tactic used by a number of sites today is to have all content visible on screens that are 1024 pixels wide, and to have some content cut off on 800-pixel wide screens. This content is considered "less important" for the site – most of the time, it is advertising. A validator which knows about the content of the different parts of the page layout can, for example, turn an error "page does not fit on screen" into a warning "advertising will not fit on some screens".

Guideline: *Each web page should contain items like a menu, link to the homepage, search facility, modification date...* [Lynch02WebStyleGuide, chapter 2]

For a human, it is usually easy to distinguish different types of content on the page, and thus to determine whether they are present. An automatic validator can perform this task by using the presentational model extensions from section 5.2.5: In the role attribute of classes which are based on the PresUiModel abstract metaclass, values like "navigation-menu" or "search" indicate the presence of the respective functionality.

Guideline: *The layout of the page should follow the de-facto standard, e.g. logo in top-left corner inside header, navigation at left (to the left of the main content), footer at bottom...*
[NielsenAlertbox, Top Ten Mistakes in Web Design, Nr. 8], [Lynch02WebStyleGuide, ch. 4]

Again, using information from the presentational model, the validator can determine how the different content areas are arranged on the page (figure 6.4), and warn if the arrangement is problematic. For example, a warning message would be output if the model specifies that the main navigation menu is in the top right corner of each page, as this poses the risk of visitors not noticing the navigation. Additionally, there can also be a check whether the layout of pages is consistent across all pages of a website, with the possible exception of the homepage, which may have a different layout [Krug06Dont, chapter 7].


LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Search:

Site map | CampusLMU

Deutsch

NEWS DIGEST

ABOUT LMU MUNICH

INSTITUTIONS & FACILITIES

INTERNATIONAL STUDENTS

RESEARCH

PARTNERS

How close galaxies lost their luminous matter

Draco, Ursa Minor and Andromeda IX are satellites of our Milky Way and the Andromeda galaxies – and some of the darkest galaxies in the universe. An international team of researchers now developed simulations on supercomputers that reveal the origin of these extreme galaxies. [more]

Dangerous twosomes – How cells spot cancer causing mutations

Sunlight is the leading cause of skin cancer. When cells are exposed to its UV rays adjacent building blocks of DNA can form a chemical unit. Researchers at the LMU now offer a detailed view of the first steps of

LMU excellent

QUICK LINKS

- Degree programs
- University Library
- Admission information
- Alumni clubs
- Faculties
- New to LMU: A guide for new students
- ERASMUS exchange
- PROFIS preparatory course
- International

Figure 6.4: To validate a page (left), access to further information (right) is needed. Using the model, a validator can check whether the page layout follows the de-facto standard.

In order to determine the position of page areas on the HTML page, it is necessary for the presentation model information to be embedded in the HTML code of the web pages, as outlined in section 5.4.

Guideline: *The area occupied by different types of page content should be within certain bounds.* [NielsenAlertbox, 10 February 2003]

Related to the previous guideline, following a de-facto layout standard also means that the area on screen occupied by navigation, main content etc. is consistent with the majority of sites on the WWW. Based on Nielsen's statistics, one can propose a rule like "the main content must occupy at least 30% of the space inside the browser window". Using the presentation model, the validator can find the main content, render the page and measure its size.

The HardwarePlatform model allows this guideline to be implemented with more accuracy. First, it specifies the screen size of the device(s) on which the content will be rendered. Second, the form factor of the device is described with the deviceDescription attribute. Nielsen's figures only apply to desktop browsers. For devices with small screens such as mobile phones, layout conventions are considerably different (e.g. multi-column layouts are avoided), so the relevant threshold values need to be adjusted for them.

The prototype in section 6.3 implements the three guidelines above: It checks for the presence of certain required page areas, for the use of a de-facto standard layout, and for a reasonable subdivision of screen space.

Guideline: *The width of a web page's main content should adapt to the width of the browser window.* [NielsenAlertbox, 31 July 2006], [Lynch02WebStyleGuide, chapter 4]

In order to use the available screen space optimally, it is sometimes recommended that a liquid layout is used instead of a fixed-width layout. By rendering the page and measuring its content width, a validator cannot reliably determine whether the content adapts to the window width, due to the fact that the part of the layout whose width changes is not necessarily the main content –

for example, it could be an empty table column. The automatic check can be made more reliable if the validator knows (from an embedded presentation model) which part of the HTML contains the main content.

Navigational Aspects

Guideline: *Important navigation links should be placed prominently on the page. Navigation paths should be short.* [NielsenAlertbox, 11 July 2005], [Lynch02WebStyleGuide, chapter 3]

Existing usability validators can already analyse the navigational structure of the site and identify some problems, such as pages which are unlikely to be found because too many clicks are required to reach them from the homepage. Web engineering solutions' navigational models provide valuable additional information: They typically only specify the "most important" links offered by a page, not the numerous links added by the standard menus or the page footer. Furthermore, the importance attribute of the presentation model (PresUiModel, section 5.2.5) helps to distinguish between more and less important links, and the pageImportance attribute in the navigation model (NavUiModel, section 5.2.6) can be used to assign more importance to links which lead to important pages.

A validator can check whether these important links are placed and displayed prominently enough on the page to be found. For example, it is advantageous if the link (which can be a button, an image or a text link) is visible without scrolling on the user's device, if the text is meaningful (e.g. no empty image alt text, not "click here"), if the screen space occupied by the link is large enough and if the link can be followed even if JavaScript is disabled in the browser.

An alternative way of using models to improve navigational aspects is to have the developer specify constraints on the navigation graph using OCL constraints (page 109). For instance, a model can declare that a certain page should be reachable with three clicks from the homepage, that it should not be possible to reach a certain page unless a given other page has been visited before, and similar. The model can also specify that going through a sequence of pages with forms should take the average visitor two minutes, that the majority of visitors should take route A to reach a page rather than route B, or that the time the user has to wait for the display of a certain page should not exceed 10 seconds. However, a fully automatic validator like the one presented in section 6.3 is not adequate for testing whether these types of constraints are likely to be met in practice, as users' behaviour is influenced by too many factors when they use the site. Thus, user tests are still necessary to collect data (e.g. with UsaProxy, chapter 7), but the interpretation of log data from the tests can be automated.

Guideline: *Navigation patterns should be implemented in appropriate ways.*

Web engineering models allow the developer to specify navigation patterns which describe the intended type of interaction. For example, a "guided-tour" describes a sequence of actions which will typically be implemented as a series of pages connected with "previous" and "next" buttons, e.g. to fill out a number of forms. A validator can verify whether the pattern has been implemented in an intuitive way: Does the page contain an indication of progress, e.g. a digit for each step, with the current step highlighted? Are the "previous/next" buttons positioned consistently and labelled correctly on each page? Should a long page for one step be split into two shorter pages to prevent that the length of the individual pages varies too much? When the page is automatically created by a web engineering tool, many of these points can be done correctly (section 4.2.2), but the developer may have supplied a different layout or graphical design for the pattern or influenced the generation in other ways.

(Since it would be hard for a progress display to be recognized as such by analysing the HTML content, the model-based validator could simply require a page area with the custom purpose of "guided-tour-progress" to be present.)

The validator can also offer pattern-related critique on the model itself, without taking the implementation into account. In the case of the guided tour, it might output a warning if there are too many steps in the tour before the user can view the final page. The pattern might not be appropriate at all; if the model states that the target audience of the site consists of expert users who use the site every day (high interactionSpeed and low patience values in the user model, page 96), then it might be desirable to prefer efficiency over intuitiveness: The users may be faster when they fill out a form on a single page than when they have to click through several pages and skip over explanatory text for single form entries.

Image and Text Content

Guideline: *Images should have alt text, unless they are decorative.*
[Lynch02WebStyleGuide, chapter 4], [W3C-WAI, WCAG 2.0, guideline 1.1]

The alt attribute is required by the HTML standard, so it must always be present. Existing non-model-based automatic usability validators flag its absence as a syntax error, and some also output a warning if the attribute value is empty (alt=""). The most advanced ones always display a warning asking the user to check manually whether an image is ornamental (see figure 6.3). This is necessary because empty alternative text makes sense if the image in question is only decorative in nature, an invisible spacer image, a "mood image" or similar. If the validator knows from the presentation model of the page that certain images are decorative (because of a role attribute value of "ornamental", page 105), it can allow empty alternative text. For other images on the page, empty alt text generates a warning. Figure 4.3 on page 68 illustrates the presence of different types of images in a presentation model.

Guideline: *Images should not be over-used depending on audience, device and type of site.*
[Lynch02WebStyleGuide, chapter 3], [HHS06WebGuidelines, chapter 14]

The validator can measure the screen space occupied by images compared to that occupied by text, and warn if the ratio of the two exceeds a certain value. This value depends on the download bandwidth, the cost associated with downloading, and the amount of available screen space (platform model, section 5.2.3), but also the importance of the images compared to that of other page areas ("ornamental" role, importance attribute in the presentation model). A larger number of less important or ornamental pictures will be acceptable if the target audience of the site is children (age in the user model, section 5.2.2) or if the site is for entertainment rather than education, reference or similar (purpose in the navigation model, page 5.2.6).

Tools should provide concrete threshold values for the text/image ratio. These values can be based on expert opinion as described in [Ivory01Metrics]: A large collection of sites is rated by experts with regard to their usability, and the allowed range of ratio values is calculated for those sites whose properties match the site that is being developed.

Guideline: *Write texts in a style that is tailored towards the web.*
[NielsenAlertbox, 15 March 1997], [Lynch02WebStyleGuide, chapter 6], [HHS06WebGuidelines, chapter 15], [Krug06Dont, chapter 5]

In order for the visitor to understand the content of the page, it must be written in a style that is easy to parse on first sight, with bullet lists, a short summary of the content, etc. Furthermore, the vocabulary that is used should be known to the visitor, and the complexity of sentences should not be too high. The acceptable complexity of the text depends on several factors which need to be supplied to the validator in a model:

- The type of site, e.g. educational vs. entertainment (purpose in the navigation model), has an influence on the vocabulary and on how much effort readers will invest in understanding complicated sentences.
- The user's age affects the way pages are read, in particular when comparing children to adults.
- The user's education (e.g. "10 years of school", specified via the schoolYears attribute in the user model, section 5.2.2) is an important hint with regard to the allowed complexity of text.
- The target audience can be highly restricted (e.g. expert users for an intranet site, lawyers; goodKnowledge attribute in the user model), so jargon or other unusual words may be acceptable.

When evaluating the structure of the text, the validator can assume that children will more patiently read introductory text paragraphs [NielsenAlertbox, 14 April 2002], whereas teenagers and adults prefer a concise style, e.g. with bullets [NielsenAlertbox, 31 January 2005].

According to research results in the area of linguistics [DuBay04Readability], the readability of a paragraph of text can be validated automatically to a certain extent. The reading level is the number of school years necessary to understand the text. It is calculated by counting characters, words, syllables and sentences in the text, and entering these values in a formula such as SMOG [McLaughlin69SMOG]. If the result indicates that the main audience of a website will not understand significant portions of the text, an automated usability validator can output an error message.

A similar approach is possible with the individual words present in the text of the web page. Using a list of the most common words in the respective language, and possibly additional lists for specific user groups (e.g. the Academic Word List [Coxhead00AWL]), the validator can identify words on the web page which its visitors are unlikely to know. Because almost every website contains special words such as company names, it should be possible for the web developer to supply an additional dictionary of words which should also be assumed to be known to the users.

An exemplary implementation of several different readability and vocabulary checks is presented with the prototype in the next section.

6.3 Prototype Implementation

Summary: The Wusab prototype supports web-based ad-hoc creation of simple presentation models. It implements a number of guidelines related to page layout as well as image and text content. Information about encountered usability issues is presented in a HTML report.

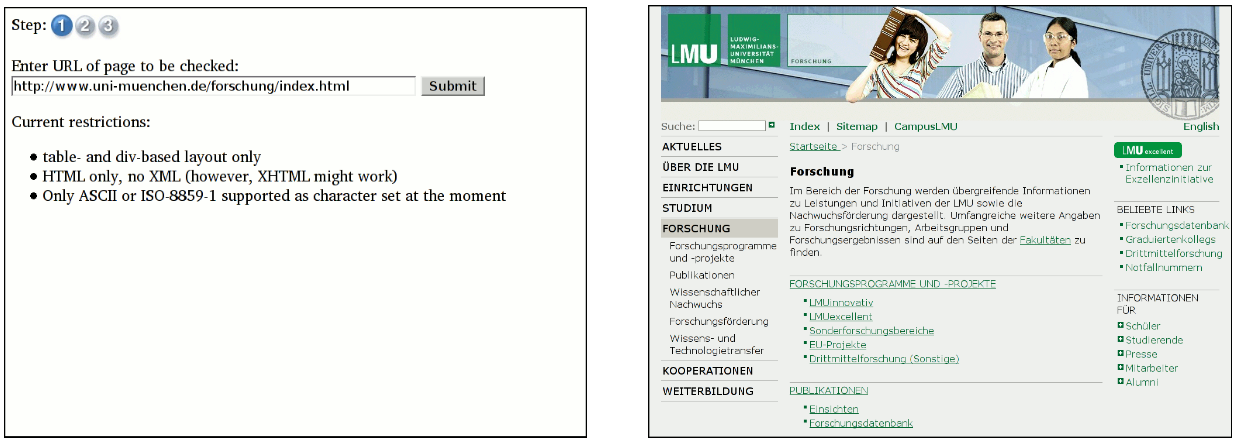
To demonstrate the feasibility of some of the ideas from section 6.2.3, the prototype of a model-based automated usability and accessibility validator was implemented. It is based on the tool first introduced in [Atterer05ICWE]. Like many of the other available validators, the program is realized as a web application. It is written in Java and runs inside the Apache Tomcat Java EE server. A web developer can enter the URL of a web page to validate, which will cause the validator to download and analyse its HTML code. The current implementation outputs an HTML page with a usability report – ideally, as outlined in section 4.2.3, validation should be integrated into a web engineering IDE so that a report (which could be generated in EARL [W3C-EARL] format by the validator) is only presented to the developer when problems are identified.

The implementation of the Wusab prototype was partially carried out by Andreas Singer and Ronald Ecker in their diploma and project theses [Singer06SemiAuto] [Ecker06SemiAuto].

In figure 6.5, the initial Wusab page is shown on the left, it asks the user to input the URL of the web page that is to be validated. On the right, the figure shows a screenshot of the web page whose URL is entered. It will be used for the examples in this section.

As discussed in section 5.4, several ways exist to attach the model information, e.g. with regard to the page layout, to the web page. At present, the prototype only supports extracting model information that is directly embedded in the page, it is not capable of accessing externally referenced XMI models.

In practice, existent web pages today do not contain any embedded model data, which will prevent the implemented model-based tests from working with these pages. To allow the tests to be run on sites without model data, the prototype supports ad-hoc creation of some basic model properties. As shown in figure 6.6, a web-based user interface has been implemented which supports the creation of a presentation model. The interface is subdivided vertically into three parts. The uppermost part displays a short introductory message with instructions (left), or the details that are stored in the embedded model about a particular page area (“Main content”, right). In the middle, the user can navigate to and select page areas by clicking on the square


Step: 1 2 3

Enter URL of page to be checked:
http://www.uni-muenchen.de/forschung/index.html

Current restrictions:

- table- and div-based layout only
- HTML only, no XML (however, XHTML might work)
- Only ASCII or ISO-8859-1 supported as character set at the moment

LMU
LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
FORSCHUNG
Suche:
Index | Sitemap | CampusLMU
English
AKTUELLES
ÜBER DIE LMU
EINRICHTUNGEN
STUDIUM
FORSCHUNG
Forschungsprogramme und -projekte
Publikationen
Wissenschaftlicher Nachwuchs
Forschungsförderung
Wissens- und Technologietransfer
KOOPERATIONEN
WEITERBILDUNG
Startseite > Forschung

Forschung

Im Bereich der Forschung werden übergreifende Informationen zu Leistungen und Initiativen der LMU sowie die Nachwuchsförderung dargestellt. Umfangreiche weitere Angaben zu Forschungsrichtungen, Arbeitsgruppen und Forschungsergebnissen sind auf den Seiten der Fakultäten zu finden.

FORSCHUNGSPROGRAMME UND -PROJEKTE

- LMUinnovativ
- LMUexcellent
- Sonderforschungsbereiche
- EU-Projekte
- Drittmittelforschung (Sonstige)

PUBLIKATIONEN

- Einsichten
- Forschungsdatenbank

Figure 6.5: On the Wusab page (left), the URL of a web page can be entered. The validator downloads and analyses the given page (right), together with any embedded model data.

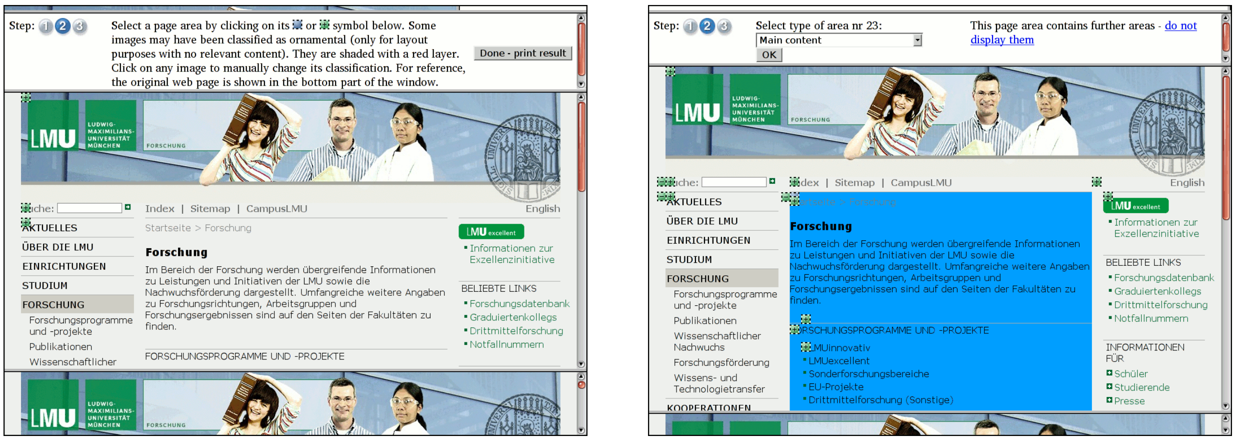
symbols: ⊞ and ⊟ display or hide the children of one area, ▣ selects an area. The current area is highlighted in blue – in the figure, it contains further children (represented by ⊞ symbols in the blue area). The bottom part of the screen shows the original, unaltered web page. This is sometimes useful because the visual changes applied by the tool to the middle part (addition of controls, disabling of all clickable links) may cause it to display incorrectly.

The prototype implements a number of usability tests which are described below. Compared to the tests supported by existing validators, these tests are either not possible at all without models (layout analysis) or provide greater accuracy (image alt text check).

Analysis of the Complexity of the Text on the Web Page

A readability grade (number of required years of school education) is calculated using a number of different algorithms, including Flesch-Kincaid, FORCAST and SMOG Grade Level [DuBay04Readability] [McLaughlin69SMOG]. These formulas combine variables whose values can be readily extracted from a text paragraph, such as the number of words, syllables and sentences. They are designed in such a way that the result is a number which indicates the approximate number of years of school education required to understand the text. Different formulas yield slightly different values, so it is a good idea to consider the average of several algorithms rather than just a single one.

The program is careful only to take real text into account, it ignores style information and scripts which are embedded in the page. Furthermore, it should be noted that the readability index values are only valid for paragraphs of text. For this reason, the validator only includes text content into its readability calculation if the text consists of complete sentences. It distinguishes between inline HTML elements (like `<a>`) and block-level elements (like `<p>`) to start new paragraphs. Additionally, it uses a simple heuristic (Java's `BreakIterator`) to differentiate


Step: 1 2 3

Select a page area by clicking on its or symbol below. Some images may have been classified as ornamental (only for layout purposes with no relevant content). They are shaded with a red layer. Click on any image to manually change its classification. For reference, the original web page is shown in the bottom part of the window.

Forschung

Im Bereich der Forschung werden übergreifende Informationen zu Leistungen und Initiativen der LMU sowie die Nachwuchsförderung dargestellt. Umfangreiche weitere Angaben zu Forschungsrichtungen, Arbeitsgruppen und Forschungsergebnissen sind auf den Seiten der Fakultäten zu finden.

Step: 1 2 3

Select type of area nr 23: Main content

This page area contains further areas - do not display them

Figure 6.6: By using the built-in web-based interface, users of the Wusab validation service can create a simple presentation model for pages in which no model is embedded.

between sentences on one hand, and non-sentence text in headings, bullet lists etc. on the other hand.

Analysis of the Vocabulary Used by the Web Page

Apart from analysing the overall sentence complexity, Wusab can also look at the individual words on the page. Each word that appears is looked up in several word lists: The most frequent 1000 English words, the next frequent 1000 words (both based on the General Service List, as adapted by Paul Nation[3]), and the Academic Word List [Coxhead00AWL]. Based on the results, the validator can determine the likelihood with which the text requires a college/university education, or a scientific background of the reader. User-supplied word lists which e.g. add the vocabulary in use in a particular company (section 5.2.2 on page 95) are not supported at present.

Analysis of Alternative Text for Images

As discussed in the previous section, alternative text in the alt attribute of image elements should be empty if the image is ornamental, otherwise it should be non-empty. The validator implements this check by looking at the data embedded in the HTML page, and outputting an appropriate warning if the guideline is violated.

Presence of Functionality

Wusab includes a straightforward test based on the presentation model. It ensures that page areas with certain functionality are present on the page. The validator requires exactly one main content area and one logo to be present on the page. Furthermore, there should be at most one search area and at least one navigation area.

[3] http://www.victoria.ac.nz/lals/staff/paul-nation/nation.aspx

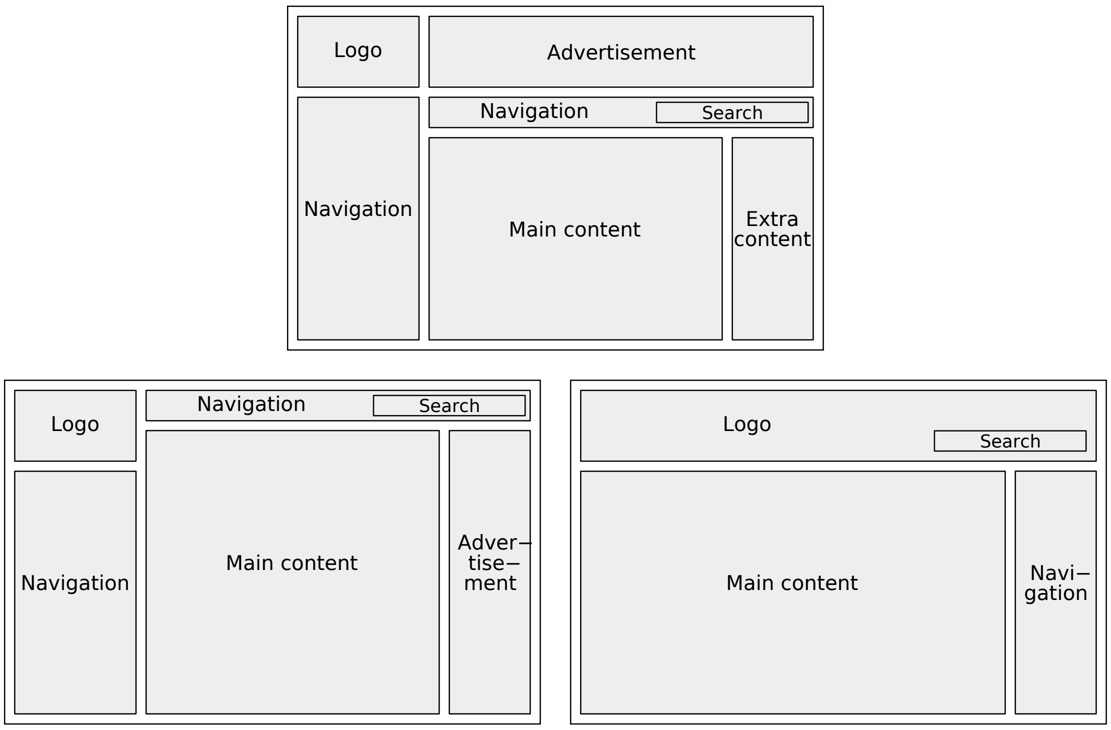


Figure 6.7: Three examples for page layouts which would be accepted by the Wusab prototype because they follow the de-facto standard conventions. [Ecker06SemiAuto]

Layout Analysis

The different parts of the layout should be arranged in a way which follows the de-facto standard. Nielsen has suggested that a certain aspect of web design should be called a standard if "80% or more of websites use the same design approach" [NielsenAlertbox, 13 September 2004]. In those cases where no single accepted way of arranging the content exists, Wusab permits all the different variants mentioned in the literature. Because the cited guidelines are often fairly vague, in some cases further restrictions were added based on observation of a number of popular websites. Due to the fact that the prototype implementation does not distinguish between different types of devices, the validation algorithm simply assumes a "desktop PC" device – the conventions for design differ significantly for other devices, in particular devices with small screens. The allowed locations of the individual page areas (see section 5.2.5 for a complete list) are as follows:

- **Logo:** Must be located in the upper left corner [NielsenAlertbox, 13 September 2004].
- **Navigation:** Navigation menus can either be arranged horizontally above the main content [Spool98Website] [Adkisson02DeFacto], or as a vertical column to its left [HHS06WebGuidelines, 7:5] or right [NielsenAlertbox, 14 November 1999]. In the latter case, no further page areas are allowed to its left or right, respectively.
- **Main content:** Users pay most attention to the part of the main content that appears "above

the fold", i.e. without scrolling [Krug06Dont, chapter 7] [Lynch02WebStyleGuide, chapter 3]. Additionally, the content should be in the middle. Thus, the current implementation requires that its left margin lies in the left half of the browser window and that its upper margin is no more than 300 pixels away from the top of the page.

- **Advertisements:** Users expect ads in the upper area of the page [Bernard01Developing], so it must be located above the main content. In practice, Wusab also allows it beside the main content to avoid too many false positives.
- **Search:** The site search facility should usually be located at the top centre or the top right [Bernard01Developing]. Wusab is much more lenient as the position is not uniform in practice: The search field can be inside any navigation area. In vertical navigation columns, it must be near the top of the area. If it is not in a navigation area, it must be positioned to the left, right or top of the main content.
- **Extra content:** The preferred location for this type of area is not discussed in the literature. The prototype allows it right-aligned or left-aligned as a sub-area of the main content, but only if no navigation area is already present with the same alignment.

Figure 6.7 shows three example layouts for which the positions of the individual page areas follow the rules above. Consequently, if given to Wusab for usability analysis, the layout of the pages would be accepted by the validator.

The layout check is implemented in part using client-side scripting. The reason for this is that analysis is supported even for very complex table-based and CSS-based layouts. Rather than providing a complete CSS parser and CSS/table rendering engine, which would require considerable effort to implement, Wusab inserts JavaScript of its own in the HTML code of the page and then displays it in a browser. The script executes inside the browser, records the pixel-based coordinates and sizes of the different page area rectangles, and resubmits this information to the validator running on the server. Using the rectangle coordinates, the rules above are much easier to implement. For the current implementation, the browser in which the JavaScript code executes is the one that is used to access the Wusab application. However, it would also be possible to use another browser which runs on the server, e.g. in its own virtual desktop session. With some existing websites, the page area rectangles overlap slightly. A certain amount of overlap is tolerated by the validator – the exact limits depend on the sizes of the involved areas.

Area Size Analysis

The relative size (in pixels) of page areas like the main content should be within certain limits. Based on the JavaScript-based page area size measurements, the validator also outputs warnings or errors if the area occupied on screen by some parts of the layout is too small or too large compared to that of other parts. The values are based on [NielsenAlertbox, 10 February 2003], but were adjusted to account for the slightly different way of measuring: Rather than counting pixels on screen, the calculations work with the dimensions of the scrollable page displayed by the browser. This area is usually less wide than the screen (because of the scrollbar, and possibly

Results of image alternative text analysis

RESULT	DESCRIPTION
ERROR	3 images on the analysed page have wrong ALT-text. If an ornamental image (a layout image with no relevant content) has a non-empty ALT-text, this will be confusing for people using screen readers or other assistive technology.
OK	8 images on the analysed page have correct ALT-text

Results of area size analysis

RESULT	DESCRIPTION
OK	Navigation areas are less than 20%
ERROR	Content area is less than 30%
OK	Advertising areas are less than 20%

Results of layout position analysis

RESULT	DESCRIPTION
OK	One navigation area is located to the left of the main content
OK	Logo is located on top of the vertical navigation
OK	No advertising found
OK	Found additional content on the right
OK	Placement of search area is OK

Results of scalability analysis

RESULT	DESCRIPTION
OK	Your website fits into a screen with a resolution of 1024 x 768 pixels
ERROR	Your website does not fit into a screen with a resolution of 800 x 600 pixels. Users will have to scroll horizontally!

Figure 6.8: Excerpt from the results for different guideline tests when running the prototype validator on the pages of Munich university.

because the window is not full-sized), but often much taller (if the page content does not entirely fit in the browser window size).

The main content should be the largest page element. Wusab outputs an error if it is below 30% of the entire page size, and still outputs a warning if it is below 50%. Navigation is the next most important page element. Still, its size should not be too large, it should lie between 20% and 30%. When users are confronted with advertising which is too large, this will have a negative impact on their opinion of the site. For this reason, the size of advertising areas should be below 20% of the page size.

For the prototype, the exact threshold values for area sizes can be changed in a configuration file on the validator server. If the validator were integrated in an IDE, an extended version of the mechanism for suppressing the display of false positives (see figure 4.2 on page 65) could instead be used to adjust settings of individual tests which are not appropriate for the application that is being developed.

Further Tests

The Wusab prototype includes checks for a number of other guidelines, such as an analysis of the contrast between text and backgrounds. It also determines whether a fluid page layout is used, i.e. a design which adjusts itself to the width of the browser window. These tests could take data from models into account to improve their accuracy. For instance, users with bad vision require designs with higher contrast, and a fluid page layout is of advantage on screens whose resolution varies, such as small handheld devices. However, the implementation does not access models for these checks at present, which will make the quality of the results comparable to that of other non-model-based validators.

Figure 6.8 shows an example for typical output generated by the tool. The depicted results are part of the usability report for pages from the university of Munich.

6.4 Discussion and Conclusion

Summary: Experience with the prototype suggests that validation quality can be improved if models are taken into account. However, an objective, fair comparison of the concept of model-based usability validation is hard: The scope of existent validators differs, and no models are available for existing websites.

This chapter has described the prototype of a model-based validator for web pages. It supports the validation of a number of example accessibility and usability guidelines based on model information which is embedded in the HTML code. The program is implemented as a server-side application.

It is evident from the descriptions in the previous section that many of the implemented tests only use heuristics rather than being able to give a 100% correct result. For example, it would be trivial to construct a page which has an excellent low readability grade, but which contains complete nonsense nevertheless. Still, considering the hard-to-automate nature of the problem, the results produced by the implementation are encouraging – tests of the prototype with existing websites suggest that the output of a model-based validator can indeed be more detailed and accurate than that of a validator which only processes the HTML data.

However, it is difficult to *objectively* compare the validation quality of the model-based Wusab prototype to that of the the existent non-model-based tools. Section 6.2.2 has already discussed the benefits of model-based validation at a conceptual level. A simple approach to a more practically oriented evaluation would be to evaluate several existing sites and to count the correctly identified issues and the false positives/negatives. Unfortunately, in the author's opinion, this would not result in a fair comparison.

On one hand, this is due to the fact that Wusab only supports a basic, limited set of model-based usability tests. The number of guidelines and the exact selection varies a lot for other existing validators. Rather than comparing the concepts, it would only be evaluated how complete the implementations are.

On the other hand, existing websites do not provide models which describe their presentation, navigation, their expected audience etc. For a comparison test, this information can be created and it can be assumed that it would be present in practice, but this involves a subjective interpretation of the implementation to create the model, so the result might not reflect the original developer's intentions. In particular, the platform model (user, device, environment) could only be guessed.

Finally, another aspect which makes comparison difficult is that the expressed aim of most existent validators is to evaluate the accessibility of websites only, and not the usability. It was already pointed out in section 2.1 that these two concepts do not exist independently of each other; bad accessibility usually affects usability. Consequently, even accessibility-only validators inherently also check for certain usability problems. However, it might not be entirely fair to conduct a test where the set of issues that should be recognized by the validator is extended to include usability problems, when the existent validators were never meant to recognize such problems.

Chapter 7

UsaProxy, a Tool for Website Usage Analysis and Testing

This chapter presents a tool called UsaProxy that is designed to support usability improvements during model-based web development. The central idea of the tool concept is to make user testing easier, by creating an accurate and detailed log of each test participant's actions on the website during the test, and by making the logging system straightforward to setup.

UsaProxy can be viewed as a complementary tool to the model-based automated page validator Wusab from the previous chapter: Whereas Wusab assists the web developer during the time of a development cycle when he is working on the design or implementation, UsaProxy helps during the user tests that follow. It improves the usability of the web application indirectly by making it less work-intensive for the developer to conduct user tests and to analyse their results. These user tests can also be remote tests (i.e. tests for which the test subjects do not come to a usability lab), which can further reduce the effort and costs of testing. The tool does not directly access the web engineering models itself, but its output can be used by a validator like Wusab to compare actual observed characteristics of the web application with the expected ones as specified in the model.

Web applications increasingly include pages which no longer employ the standard request/response paradigm to update the user interface when the user performs an action, but instead use JavaScript code to contact the server and then change page content without reloading it. In general, it is harder for the developer to obtain feedback about the usage of these AJAX applications than with conventional web applications: Since the server is only contacted to store or retrieve data and not for all user interface updates, data from logfiles becomes less useful for analysing the users' actions. Even for web applications which do not use JavaScript this way, the information in the server logs is often not detailed enough. For instance, it does not include the order in which the fields of a form were filled.

UsaProxy solves this problem with its detailed tracking of all user interaction with the web page. Implemented as an HTTP proxy, it uses AJAX technology itself to record all mouse movement and clicks, key presses and other activity on the page. Furthermore, every entry in UsaProxy's log file identifies the URL and the exact HTML element that was interacted with on the respective page. This makes it easy to extract facts like the exact time when a particular button was clicked. Finally, one interesting feature of the tracking solution is that no installation of software is necessary on the client machine. Instead, either the user's browser needs to be reconfigured (if interaction on third-party sites needs to be observed) or no changes at all are necessary (if the site visited by test users is under the web developer's control).

This chapter is based on several publications about UsaProxy [Atterer06UsaProxy] [Atterer07TrackingAJAX] [Atterer06LoggingAJAX].

7.1 Related Work

Summary: Many tools exist to automatically record user actions during testing. They depend either on client-side software installation or unreliable URL rewriting in HTML documents, or do not accurately identify clicked-on buttons etc. Logged data typically includes mouse and keyboard input, and sometimes direction of gaze.

The logging concept of UsaProxy builds upon and improves a number of related previous efforts in the area of user interaction logging. While all individual "building blocks" of the technology were demonstrated to be effective in related work, UsaProxy's combination of the following features is unique: It can be used without client-side software installation, the logging is very detailed (e.g. it records the mouse coordinates relative to the page instead of screen coordinates), and it identifies clicked-on buttons (and other elements that are interacted with) via their position and ID in the browser's DOM tree. Section 2.3.2 has already given a short overview of the possibilities to record participants' actions during user testing, ranging from server logs to audio and video recordings of the test subject.

Logging User Behaviour With an HTTP Proxy

The tool is implemented as an HTTP proxy which makes a small modification to the HTML pages that pass through it (see figure 7.1). This concept of using proxies for purposes other than caching was discussed in [Barret99Intermediaries]. UsaProxy takes advantage of it to insert JavaScript code for user tracking on the pages it delivers to the browser.

The WebQuilt system ([Hong01WebQuilt] and section 2.3.2) uses a special website to log the pages that the users visit on the web. However, the tool is not an HTTP proxy, but a normal server application. It modifies all URLs in the forwarded HTML pages to point back at the WebQuilt service. This has the advantage that the web browser does not need to be reconfigured to view a third-party website. However, there are also several disadvantages: The URL displayed in the browser is not that of the original site, which can confuse the test participants or cause them to accidentally turn off logging by entering a URL manually. Also, the URL rewriting will fail in many cases, most notably when URLs are generated dynamically using client-side scripting. Furthermore, it is difficult with this approach to correctly implement many aspects that advanced web applications utilise, such as AJAX support, CSS `@import` declarations, cookies,

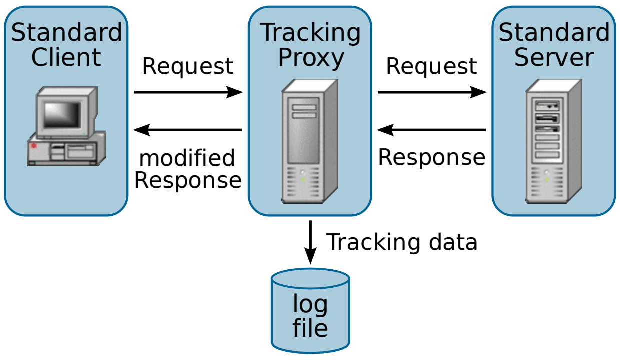


Figure 7.1: UsaProxy modifies all HTML pages that pass through it. The JavaScript code it adds can monitor mouse movements, keyboard presses and other types of interaction.

HTTP redirects etc. WebQuilt only appears to support basic HTML page rewriting. Finally, it does not employ any kind of client-side tracking, so the level of detail of its logs is comparable to that of web server logs.

Finally, SCONE/TEA [Obendorf04Scone] is another proxy-based solution that concentrates on full automation of the testing process, e.g. with an XML-based file format which allows the description of a user test and the tasks it consists of. The system is designed for installation on the client machine. It uses a combination of JavaScript and a Java applet to record information such as the exact links that are used to navigate from one page to another or the input made in forms, but apparently not mouse movements or individual key presses.

Client-Side Scripting for Mouse Tracking

Using JavaScript, it is not difficult to access information about the current position of the mouse pointer, the keys that the user presses while he uses the website, and a variety of other events, such as scrolling the page. The Cheese prototype [Mueller01Cheese] uses JavaScript to track mouse movement on an HTML page. It continuously records the current position together with a timestamp and sends it to the server for later analysis. The paper does not go into details about the exact approach, but it seems that only mouse coordinates are logged by the prototype, not clicks, scroll events or other types of interaction. Furthermore, the elements on the page (e.g. buttons that the mouse moves over) are not identified via their position in the DOM tree. The system is not fully automated – HTML pages were probably prepared manually for the user study by adding the JavaScript code for tracking. MouseTrack [Arroyo06MouseTrack] combines the approach from Cheese with URL rewriting to automatically add mouse movement tracking to visited pages.

The client-side part of the UsaProxy implementation uses a similar mechanism to collect data. In comparison to Cheese and MouseTrack, it supports more detailed tracking (for instance, of key presses). Unlike MouseTrack, its implementation as an HTTP proxy (rather than a server which performs URL rewriting) ensures that JavaScript is added automatically to any page that the user visits. In addition to this, the logging functionality is added in a way that was carefully designed to cause as few problems as possible with existing JavaScript code used by the web page.

WebVCR [Anupam00WebVCR] is a system which records user actions to allow bookmarking of web pages that require several steps to reach, including e.g. filling out forms. Two implementation variants are discussed, a server-based solution (with URL rewriting) and a client-based approach. Only the latter appears to have been implemented. It uses JavaScript together with a Java applet and LiveConnect to track user actions. The actions necessary to reach the next page are recorded for later playback. This does not include detailed mouse logging, but it would be possible to extend the system to record mouse movement, key presses etc. Unfortunately, the system's architecture relies on the fact that it is possible for one browser window to access another window and to steer what is being displayed in it. Since WebVCR's introduction in the year 2000, the security model for JavaScript in major browsers has been made more restrictive, so it seems unlikely that the system will work with current browsers – any attempts of the JavaScript to make cross-domain modifications in another window will fail if the window does not display a page on the same website. UsaProxy avoids this and related problems with security restrictions by making the browser execute the added logging code in the context of the site that each HTML page was requested from.

Client-Side Installation of Software

Recording of user actions is possible in a variety of ways if special software is installed on the client machine. While this avoids problems with the security restrictions imposed by browsers, it does not support UsaProxy's goal of making ad-hoc remote user tests possible: As shown later in this chapter, users are less likely to participate in a remote user test if they have to install software on their machine. Furthermore, if the logging code does not run inside the browser, it is more difficult for it to identify the exact button that was clicked on – information that can be very useful when analysing the log data of a user test. For example, these disadvantages apply to applications that record every change made on the user's screen, e.g. recordMyDesktop (`http://recordmydesktop.sourceforge.net`) or Morae Recorder (`http://www.techsmith.com/morae/record.asp`). The resulting video footage can help with later analysis, but it is not well suited for automated processing to find out the time needed to achieve goals.

Other client-side software can restrict the recording of information to the users' actions in the browser window instead of the entire desktop. In [Goecks00Unobtrusively], this is achieved by providing a special version of Internet Explorer which needs to be installed on the client machine. Due to the way the logging functionality is implemented, the log data is not very detailed; only the number of clicks on each page and the amount of scrolling and mouse movement is recorded. However, this level of detail is enough for the purposes of the paper, which is only interested in measuring the visitor's level of interest in each HTML page.

Another way of implementing client-side recording functionality is to provide a plug-in or extension for the user's browser. The extension is not subject to the same security restrictions as code running on pages loaded by the browser. At the same time, it has full access to the mouse coordinates as well as the DOM tree of the HTML document. Examples for this kind of logging solution include Webtour [Sastry99Webtour] for Netscape or Internet Explorer, TestGen4Web (`http://developer.spikesource.com/wiki/index.php?title=Projects:TestGen4Web`) for the Firefox browser, or the commercial plug-in of UserZoom (`http://www.userzoom.com`).

The two tools presented in [Reeder01WebEye] are intended for user studies in a laboratory. They combine client-side input device tracking with eyetracking. The first tool, called WebLogger, is installed on the computer that the test participants use. It runs an instance of the Internet Explorer browser and records user actions in the browser window. The logged data includes information on when a new page is loaded, scroll events and the gaze data from an eyetracking device that is connected to the computer. It is not mentioned whether key presses are also recorded. The second tool WebEyeMapper is designed to support later analysis of the log data. It can replay the browsing session and supports mapping of mouse coordinates to elements on the web pages, such as buttons or links.

Similar to the functionality of UsaProxy, the two tools allow for detailed logging of user actions, but extend it with recording of eyetracking data. Due to the necessity for special hardware, in-lab usability testing is the only intended scenario of use, and the logging system is restricted to a particular platform (Windows with Internet Explorer). While the eyetracking support is an advantage for laboratory tests, the platform and hardware restrictions make WebLogger and WebEyeMapper less attractive for other types of user testing. For example, unlike with UsaProxy it is not straightforward for arbitrary invited users from the Internet to participate in a test. Moreover, the additional post-processing which needs to be performed with WebEyeMapper prevents real-time access to the final log data and requires some additional work by the developer. Because WebEyeMapper needs to revisit all pages during the delayed analysis phase, websites with dynamically generated pages present a problem for the creation of an accurate log.

Comparison to Systems With Eyetracking

UsaProxy is not normally intended to perform logging of eye gaze data in addition to the user's mouse and keyboard input. An extension which allows for this is discussed in [Drewes07Detailed], but when the cost of eyetracking hardware cannot be justified or the software is used for remote usability testing, no gaze data can be collected. This could be interpreted as a disadvantage, but as previous work shows, it is possible to some extent to predict gaze direction based on mouse pointer data, or to cause the users to move the mouse to the screen area where they are looking.

The connection between the position of the mouse pointer and the gaze direction during web browsing was examined in [Chen01Correlation] in an experiment. The results allow the formulation of a number of rules for predicting where users look on a page. For example, when the mouse pointer was moved to a page area during the experiment, the probability that the same area was also looked at was 84%. Also, when the pointer moved from one non-empty page area to another in a saccade, the chance that the gaze was inside the involved regions was 70%. The distinction between empty and non-empty areas seems necessary, it is also supported by a study in [Mueller01Cheese] which revealed that the visitors of a website move the mouse pointer to an empty area to avoid clicking on a link. In general, the latter paper reports that the mouse pointer is often moved around the page as a reading aid.

In the absence of hardware eyetracking, an approach followed by some sources is to make the users move their mouse to the spot on the screen they are looking at. Simply telling them to do so is not very effective, as gaze movement often happens unconsciously, so it is attempted to prevent users from seeing text content properly unless they move the mouse over it. The authors of [Ullrich02PoorMans] prevent text from being read by adding JavaScript to the web pages of their application. It covers the text with a black bar, makes its colour very similar to that of the background, or reduces the font size. After the mouse pointer is moved over an area, a delay of 0.7 seconds is suggested before the text becomes visible, to avoid that the text of content becomes visible when the pointer moves over page areas that lie between the previous pointer position and the destination area. The "Enhanced Restricted Focus Viewer" [Tarasewich04DiscountEye] prevents text from being legible by blurring it. While the system is also intended for web pages, these pages need to be prepared for use by converting them to bitmaps and specifying clickable areas for links, buttons etc. For either of these two tools, it is not clear whether the changed user experience will not affect the users' reactions too much, e.g. because they are annoyed by the delay before text becomes visible.

7.2 Tracking User Actions on Web Pages

Summary: The logged data should include details on navigation between pages and actions within a page. It should be platform-independent, transparent and robust. UsaProxy achieves these goals by inserting its own JavaScript code on web pages in a minimally invasive way.

This section describes the aims of the logging solution in the form of a number of requirements it should fulfil, and then shows what approach was taken to address these requirements. At the technical level, the logging solution is based on several building blocks that are generally available for browsers and web-based applications, but used for a slightly different purpose here. After the more general overview of the system architecture in this section, section 7.4 below provides details on the implementation.

7.2.1 Requirements

After the analysis of the properties of related logging tools in section 7.1, the following requirements were determined as prerequisites for a web-based user interaction tracking system which combines detailed logging with uncomplicated setup and log analysis:

- User actions should be recorded in as detailed a way as allowed by the technical constraints. The recorded data should include the user's movement through the web application, both the list of URLs that is visited and in-page navigation (scrolling, mouse movements). Furthermore, any input provided to the application via clicks, key presses, drop-down menu selections etc. should be tracked.

- To avoid problems with dynamically generated HTML content, the HTML data delivered to the user's browser at any stage should also be recorded.
- Like the web applications with which it can be used, the logging system itself should be platform and operating system independent, and able to work with any standards-compliant technology both at the server and the client side.
- Operation should be transparent – the user experience should not be altered in any way while logging takes place, and from the server's or client's point of view, the logging system should behave like any other client or server, respectively.
- As few changes on the client machine as possible should be necessary. For instance, there should be no requirement for special hardware (e.g. for eyetracking) or for software installation.
- As few changes on the server machine as possible should be necessary. In particular, the server may belong to a third party, so at least one mode of operation should require no changes at all.
- The logging should also work with AJAX applications, due to the fact that AJAX is used in a steadily increasing number of web applications.
- The log file format should be suitable for automated parsing, and should contain enough information to allow automated analysis for typical usage scenarios. For example, the fact that the user has reached a goal by clicking on a certain button in an AJAX application should be easily recognizable in the log.

7.2.2 Approach

Depending on the type of user test, sometimes the necessary setup changes for logging functionality must be introduced at the client side, and sometimes at the server side – there is no single possibility which serves in all situations. Thus, the approach taken by UsaProxy is to allow the logging functionality to be inserted *either* at the client or the server side. It is up to the person conducting the test to decide for one variant on a case-by-case basis. The setup type does not have any effect on the accuracy or quality of the collected log data.

As mentioned above, an HTTP proxy is inserted between client and server. The concept of proxies is a part of the HTTP standard, and is supported by all current web browsers. Originally designed for caching of HTML pages to save bandwidth, it permits full access to the stream of HTTP requests and responses between client and server. This also means that the proxy can make arbitrary modifications to the data that passes between the two. Figure 7.1 (page 151) shows an overview of this architecture: Any request that the browser sends reaches the proxy. The proxy passes it on to the server, and subsequently returns the server's reply to the browser. In addition to this, the request headers and the server response are written to a log file. The system also has a "server mode" where it mimics a regular WWW server, passing on all incoming requests to the actual back-end server on which the website runs.

If the proxy only recorded the HTTP requests and responses, the accuracy of the tracking would not be significantly more detailed than that of server logs – it would only contain the visited URLs, data that is input into forms, and similar. To receive detailed information about the user's actions in the browser, e.g. any mouse movement, it is necessary to execute code directly on the client machine to record this data. In the form of their JavaScript support, current browsers offer a suitable technology for this purpose. The browser's security model only allows code to access mouse and keyboard information if the code was loaded as part of the currently displayed HTML page (or is referenced from it). The logging system can work around this restriction: Before it passes HTML content back to the browser, it makes a modification to it which causes special JavaScript code for logging to be loaded in the context of the page. To the client, it seems that the page content as sent by the server already contains the code, so it is executed in the context of the page without restrictions.

In practice, several pitfalls must be avoided to ensure the described process works under all circumstances. For instance, the log data must be transmitted to the proxy somehow, the logging JavaScript must not interfere with web applications, and slight differences and incompatibilities between the major browsers make the development of a working logging system a non-trivial task. Section 7.4 describes the details of the final implementation.

7.3 Making Use of the Collected Tracking Information

Summary: The tool concept is useful in many scenarios. When used for usability testing, it helps to reduce the effort of conducting a test because it allows remote testing and does not require installation of special hardware or software. Different types of user tests are supported, including inviting test participants over the Internet.

The data that is produced by a user logging application like UsaProxy is very detailed by default, e.g. "at this point in time, the mouse button was clicked over the link with the ID `tableofcontents`". Further processing is necessary to infer more abstract information about the task performed by the users, or even the users themselves. When a tool like UsaProxy is employed to track users while they interact with a web application, the following types of information can be distinguished:

- Concrete usage information: The data that is directly collected by tools describes the actions on the web page in detail, e.g. which parts of a form were filled out in what order.
- Abstract usage information: More abstract aspects of the application usage can be deduced from the logs. For example, if a form was completed very quickly by a user, it is likely that the user did not read the form completely and that he entered bogus data. Similarly,

if the log shows that the user repeatedly moved the mouse over the buttons for alternative actions, it is possible that he has difficulties making a decision.

- Concrete user information: Certain parts of the interaction log reveal details about the user's abilities and experience. For instance, it can show that the user is able to move the mouse to targets accurately, or that he types quickly.

- Abstract user information: Further general information about the user can be inferred independently of his use of web applications. This includes his general computer skills, his ability to spell correctly, or his interest in certain topics.

Resulting from these types of data, different use cases for logging software like UsaProxy can be formulated. The ability of UsaProxy to require either no server changes or no client changes makes it flexible enough for many tasks, including the following:

- Automatic generation of profiles for users;

- Debugging of web applications during development;

- Analysis of the usage of already running websites, for example for marketing or targeted advertising, but also to identify usability problems that users have with the site in practice;

- Self-adapting websites, i.e. websites which automatically adjust the content and interaction possibilities (welcome message, menu entries etc.) to the level of expertise and to the interests that the user is most likely to have;

- Website usability testing – this use case is described in more detail below;

- Shared browsing, i.e. two browsers which access the same proxy are connected to each other in such a way that any action in one browser is also performed identically in the other;

- Live customer support, allowing support staff to watch and assist a user who has problems using a feature of the site.

The last two points are interesting applications of the tracking technology which were further developed in [Atterer07AppSharing]: For shared browsing, an extended version of UsaProxy records the clicks, mouse movements and other interaction performed in each browser, and visualises it appropriately (e.g. with a second mouse pointer) in the other browser. Additionally, an AJAX-based chat facility that is added to displayed pages allows the users to communicate with each other directly.

In the live customer support scenario, visitors of a website who have problems using it can contact a support hotline by clicking on a special link on the website. Without browser reconfiguration, this causes UsaProxy to start its tracking, and enables a member of the support hotline to understand the problem by observing the users' actions. Via the text chat, the support staff can then ask for more information or give advice about possible ways to solve the problem.

Lowering the Cost of Usability Testing With UsaProxy

The logging approach which is introduced in this chapter has the potential to reduce the amount of work that is typically associated with a user test. It tries to solve the dilemma that a developer often faces when he wants to conduct user tests: On one hand, in order to gain useful insights about usability issues that are present in an application, it is necessary to conduct testing with a relatively large number of subjects and with detailed recording of usage data. On the other hand, the available development resources are limited in terms of money, time and manpower, which places a limit on the number of test subjects and the time available for the test.

The tool concept aims at automating those parts of the user test for which it makes sense, relieving the developer of many aspects which unnecessarily take up his time, and allowing him to concentrate on his observation of the test users. Preparing a "classical" usability test can require significant organization and preparation efforts:

- Before the test can take place, the technical infrastructure must be set up. Typically, this involves a machine on which tracking software is installed, e.g. for mouse/keyboard recording and screen capture. It may also include the addition of special hardware like an eye-tracking device or a video camera that observes the user. The setup must be tested to ensure that all components work as expected.
- Recruiting test participants can be time-consuming. In particular, it can be difficult to design a schedule if these participants have to come to the lab in person to take part in the test. If some test subjects are far removed geographically, transporting them to the testing hardware (or vice versa) can become expensive.
- When the test takes place, usually only one user at a time can use the system, either because setting up a second system would be too much effort or because the developer can only observe one test session at a time.
- After the test, the output generated by the different types of logging software and hardware (video, desktop actions, eyetracking etc.) must be collected, interpreted and combined into meaningful summaries. In practice, at this stage a lot of the data is ignored most of the time because it is not relevant. For example, video data is typically only consulted in a few special cases where further information is needed because the other types of data do not seem to make sense.

UsaProxy takes a different approach: Rather than collecting all types of data that can be recorded using current technology, it concentrates on the most important types of data and instead gets rid of many of the constraints that the above procedure places on the user testing process. This has consequences which affect the cost of testing:

No test lab is necessary – remote testing becomes possible: As shown above, much of the effort of a test is caused by the necessity to prepare a laboratory for use, to have the test subjects come to it, and to supervise the test. However, in many cases the results of a remote user test are comparable to those of a local one in terms of the number and severity of usability defects that are found [Brush04Comparison]. The test participant and the developer only need a voice connection (telephone or voice over IP) for communication, the user interaction can be recorded on the user's computer at home or work, and transmitted to the developer over the Internet. If there are concerns that some information will be lost by only performing remote usability tests (e.g. the users' initial reaction to the web application), it is possible only to invite some users to the lab while still relying on remote testing to find the majority of usability problems.

No special hardware or software is required: UsaProxy itself is the only software that is necessary to record the user's actions on web pages. On one hand, this means that it is not mandatory to buy further tools when preparing a user test. However, the more important aspect of this fact is probably that the risk of technical problems during software setup on remote machines is reduced, since only their browser settings may need to be adjusted. Often, the application's target audience (and thus the audience chosen during testing) includes users with average or below-average computer skills, for whom software installation may be a challenge. In addition to this, most web users (whether interested in technology or not) are suspicious of installing software on their computer and generally prefer not to do so (see also section 7.6). Finally, UsaProxy only records the actions that happen in the users' browser, not any other action they perform on their desktop. Compared to desktop recording solution, this approach improves the users' privacy – only the data that is strictly needed for the user test will be recorded and sent to the developer.

The data from special hardware, like a video stream of the user's face or the direction of his gaze, can provide additional clues about the users. However, as shown in section 7.1, mouse movement is related to gaze direction, and a voice connection is often sufficient to get an impression of the users' opinions and the problems they encounter, so additional hardware may not be very useful, especially when the effort of deploying it remotely is taken into account.

Scenarios for Usability Testing With UsaProxy

The preceding paragraphs have already outlined ideas on how to employ UsaProxy for usability evaluation. The following scenarios of use are imaginable for the tool:

"Classical" usability testing: The developer invites users to take part in a test and asks them to perform tasks on the web application, using a machine which has been set up in advance to perform logging via UsaProxy. The proxy itself can run on the same machine or it can be installed on another computer, e.g. the developer's workstation.

Tests of third-party sites: Since the HTTP proxy can add its tracking functionality to any HTML page before delivering it to the browser, it is possible to perform testing for sites which are not under the control of the developer. This can be useful if the aim is to compare different interactive sites, e.g. to evaluate the solution of a competitor, or if the task given to the user is of a general nature that may cause him to use search engines or other online services.

It should be noted that the current UsaProxy version cannot track user actions on sites that encrypt their content with HTTPS, which is frequently used for online payment, logins etc. However, temporary HTTPS sessions without tracking (such as the short session required to perform a login at Google Mail) can be mixed without problems with tracking of HTTP sessions. For this, the browser configuration should specify that no proxy is used for HTTPS traffic.

Usability tests for rich Internet applications: The number of web applications which use AJAX for their user interface is steadily increasing. UsaProxy is designed to allow tracking of clicks, mouse movement etc. even for these types of web applications, where the content displayed by the browser varies significantly during the time a single web page is displayed. Furthermore, the tool's own JavaScript-based implementation does not interfere with the JavaScript code already present on the website. In contrast to this, usage analysis based on simple server-side log files fails for rich Internet applications.

Remote usability testing: As described above, users can participate in a test from their office or home. If they need to be tracked on arbitrary websites, they need to reconfigure their browser to use UsaProxy as an HTTP proxy. However, if they are to be tracked on a website over which the developer has control, UsaProxy can be run in server mode, so no changes at all are required to the client setup. A voice connection between the user and the developer is recommended.

Parallel usability tests: It is feasible to automate remote tests with individual test participants in such a way that the developer does not need to be involved at all during the test: Each test subject can be given instructions on how to reconfigure their browser and on the tasks that should be achieved. The start and end time of each task can be extracted from the log, together with information on success or failure. By not requiring the developer to observe each test as it takes place, many users can take part in parallel, and they can take part at any time they like – a fact which is important if remote users live in a completely different time zone. Because analysis of the logs can also be automated, this type of test scales to high numbers of test participants. However, it should also be kept in mind that the test results will be less detailed if the developer is not able to discuss and understand each user's problems while the test takes place.

Inviting test participants over the Internet: Recruiting test users was mentioned above as one of the tasks that increase the effort associated with a user test. If the web application is already available publically, it is possible to invite arbitrary visitors to take part in a test. For example, during a public beta phase, a message on the site can ask for volunteers. If a user agrees to participate by clicking on a button, UsaProxy's logging is switched on and the user is presented e.g. with instructions on the tasks that should be tried out.

By using a button with "opt-in" functionality, privacy problems with the technology are avoided: Interaction with web pages should not be tracked at the level of detail permitted by UsaProxy without the users' knowledge about the tracking, and their explicit permission.

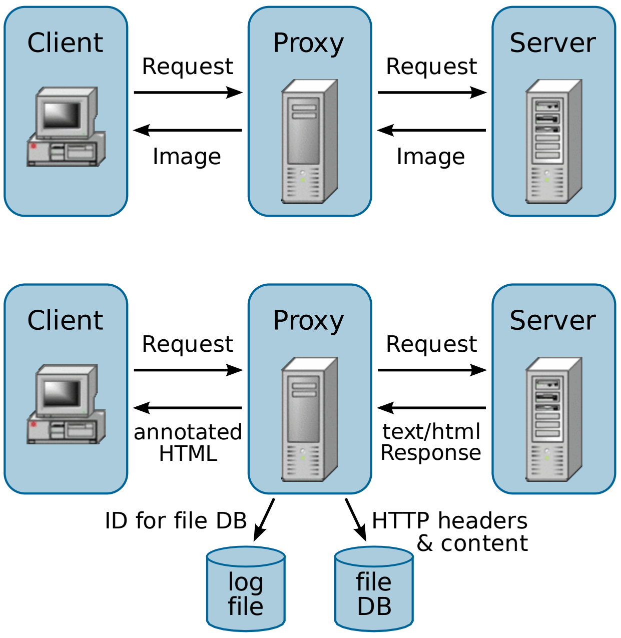


Figure 7.2: ***Top:*** **Server responses with images are returned unmodified to the browser.** ***Bottom:*** **For HTML content, JavaScript is inserted and the request and response are logged.**

7.4 Implementation

Summary: Numerous technical problems had to be resolved to allow transparent insertion of JavaScript on HTML pages, perform non-invasive logging of mouse movement and other types of interaction, work around browser security restrictions, and to permit flexible deployment in proxy, transparent proxy or server mode.

UsaProxy is implemented as an intermediary which can work in several modes of operation (HTTP proxy, transparent HTTP proxy, HTTP server mode with request forwarding). On one hand, the tool consists of the HTTP forwarding code, which is realized as a stand-alone Java program. On the other hand, JavaScript code executes on the client browser to record user actions. The JavaScript code sends the obtained data back to the proxy, allowing all information gathered by the two parts to be stored on the same machine the proxy runs on.

UsaProxy was implemented and later improved by Monika Wnuk as part of her project and diploma theses [Wnuk05UsabilityProxy] [Wnuk06Remote].

7.4.1 Forwarding and Modifying HTTP Requests and Responses

The purpose of UsaProxy is to serve as an intermediary to which browser HTTP requests can be directed in different ways, usually by reconfiguring a browser's HTTP proxy settings. When the tool is started, it listens on a network port that is configurable via a command-line option, and

```
<html>
  <head>
    <script id='proxyScript_UsaProxy' src='http://192.168.0.1/usaproxylolo/
      file/proxyscript.js?sd=5357&amp;ts=2008-01-12,17:10:53&amp;id='
      type='text/javascript'></script>
    <title>Homepage</title>
  </head>
  <body>
    <p>This is the homepage.</p>
  </body>
</html>
```

Figure 7.3: Before returning HTML content to the browser, UsaProxy inserts a `<script>` tag. Later, the browser's attempt to fetch the script content is also handled by UsaProxy.

waits for connections from browsers. Figure 7.2 shows the sequence of actions when such a connection occurs and the browser sends a request: Upon receiving the request, UsaProxy forwards it unchanged to the server the request was originally directed at. This behaviour differs from that of normal caching HTTP proxies, which e.g. add `If-Modified-Since` headers to determine whether they can serve the response from their cache rather than re-requesting it from the WWW server. UsaProxy is not a caching HTTP proxy. Even though it does record requests and responses, this only happens for logging purposes, and the recorded data is not used as a cache.

When the server's response to the request arrives, two cases can be distinguished: If the server's response indicates that the content is an image or other non-HTML data (top part of the figure), the data is returned to the browser without any further action, i.e. it is not stored on the proxy machine. This is necessary to reduce the storage requirements for the log data – because images and other non-HTML content are static most of the time, it will not affect the completeness of the logs in the vast majority of cases.

If the server's response consists of HTML code (its `Content-Type` is `text/html` or `application/xhtml+xml`), only a single modification is made to the content before it is passed on to the browser: UsaProxy inserts an HTML `<script>` tag immediately after the opening `<head>` tag of the HTML document. Figure 7.3 shows a real-life example for the inserted tag. Essentially, it has the form

`<script src='http://`*`a.b.c.d`*`/usaproxylolo/`*`path`*`?`*`args`*`'></script>`

where *a.b.c.d* is the IP address of the computer running UsaProxy, *path* is the path of a JavaScript file, and *args* consists of further arguments which are appended to the URL. When the browser makes another HTTP request to download the JavaScript code, it reaches UsaProxy like all other requests. Upon receiving it, the proxy recognizes its special form (the "usaproxylolo" path component in the URL) and does not forward it to any server, but directly returns its own JavaScript, disguised as a regular server response. As soon as the browser has received the file, the client-side tracking code is executed.

Some further changes to the HTTP data are necessary for the HTML modifications to work properly: Inserting the additional tag increases the size of the document, so the proxy adjusts the `Content-Length` HTTP header as appropriate before returning the modified document to the browser. Furthermore, the HTTP request is not really passed on to the server completely unmodified as claimed above: Some websites return HTML data in compressed form if the browser indicates that it supports compression. Inserting the `<script>` tag in compressed data, while possible, would be difficult, so UsaProxy simply disables compression by overriding any `Accept-Encoding` header sent by the browser, and always passing the header "`Accept-Encoding: identity`" to the server.

In figure 7.3, a number of arguments are appended to the script: "`sd`" stands for "server data", the value represents the ID under which the headers and HTML content are recorded by the proxy. Furthermore, the tracking JavaScript uses the same value in all log entries for a page, e.g. for each click or mouse movement. "`ts`" is a timestamp of the local time on the proxy machine – due to clock skew, the time can differ on the client machine, which can lead to incorrect timestamps in the log file if the time is not synchronized. "`id`" is only needed for shared browsing, and is empty in the example.

7.4.2 Operation Modes of the Proxy

UsaProxy features three different modes of operation, each of them with its individual advantages and disadvantages. An overview is given in figure 7.4 and below:

HTTP proxy mode: In the standard HTTP proxy mode, the browser on the client machine needs to be reconfigured. However, once this has been done, interaction tracking works on all sites that are visited, they do not need to be controlled by the developer. Furthermore, UsaProxy can run on any publically accessible machine, it does not need to be on the same local network as the client or server.

Transparent HTTP proxy mode: For this mode, a special network setup is required which intercepts all HTTP traffic exiting a local network and redirects it to a running UsaProxy instance. This is possible with access to the gateway router. Alternatively, a Linux machine with two network ports which runs UsaProxy can be physically inserted between the client and the Internet, and can filter HTTP traffic using its `ebtables` and `iptables` functionality. In this mode, all hosts whose traffic is filtered can be tracked on all sites without reconfiguring their browsers.

When invoking UsaProxy on the command line, the choice between normal and transparent proxy mode does not need to be specified; the program only knows one "proxy mode" and is able to distinguish automatically whether it is being used as a normal or transparent proxy.

Server mode: UsaProxy can also imitate an HTTP server directly. In server mode, it should be configured to wait for connections on the standard HTTP port 80 while running on the official IP of the website. Any incoming requests are forwarded to a back-end HTTP server

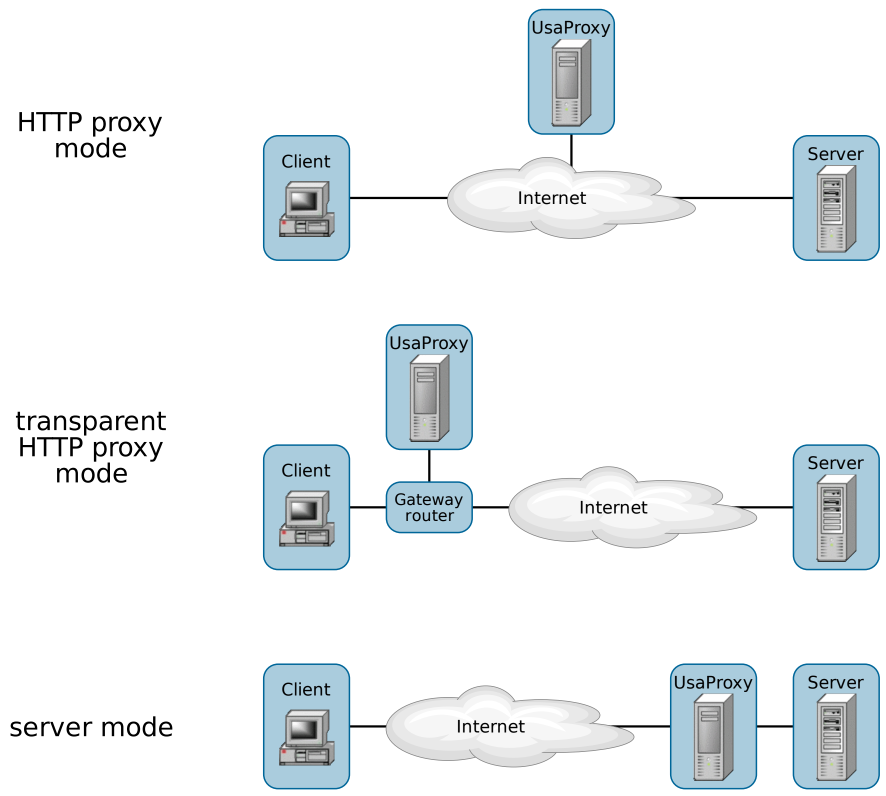


Figure 7.4: In HTTP proxy mode, the client browser needs to be reconfigured, but can then be tracked on any site. For transparent proxy mode, no browser reconfiguration is needed, but access to a router. Server mode is useful for tracking arbitrary visitors of a single site.

whose address and port is specified on the command line when the proxy is started. The real website is installed on this back-end server.

With server mode, it is possible to track all visitors of a particular website without requiring them to reconfigure their browsers. Moreover, the server setup of the website does not need to be touched, except for changing the server IP address or port number. However, the website must be controlled by the developer to be able to perform these setup changes.

The different operation modes have an effect on the hostname in the URL that is added to HTML documents inside the `<script src="...">` HTML tag by UsaProxy: Early versions of UsaProxy used a made-up hostname of “lo.lo” instead of the *a.b.c.d* IP address mentioned above [Atterer06UsaProxy]. This worked fine as long as the browser had been reconfigured to access the WWW via a proxy. However, in the case that UsaProxy was deployed as a transparent proxy, the browser would attempt to resolve this name to an IP address – the attempt would fail and the request for the script would not reach the proxy.

Another variant that was tried was to use the hostname of the website from which the HTML document was requested. For example, if the browser requested http://google.com, then the page content from the Google homepage would include a script URL that began with http://google.com/usaproxylolo/. Just like the solution using an IP address, this approach works in all three modes.

For activity tracking on websites alone, the use of either any resolvable hostname or an IP address is possible. The reason why an IP address is preferred is that a "global cookie" is required for the shared browsing scenario [Atterer07AppSharing], i.e. a cookie by which a browser can be uniquely identified regardless of the website on which the tracking JavaScript is loaded. The browser security model disallows access to cookies by arbitrary sites, but makes an exception for the site from which a script is loaded. When that site is always the same IP address *a.b.c.d*, a cookie stored for *a.b.c.d* is accessible on all web pages.

7.4.3 Client-Side JavaScript-Based Interaction Tracking

With the logged HTTP requests and responses, it is already possible to determine users' navigation behaviour on non-AJAX sites, including popular paths through the site, the most important entry and exit pages, or the average number of page views. UsaProxy's client-side JavaScript code augments this information with details about the user actions on each of the requested pages.

The primary concern which guided the development of UsaProxy's JavaScript code is the necessity to avoid any interference with other code that may be loaded by the web application itself. The first precaution of the code is to use a "_UsaProxy" suffix for all global JavaScript variables to reduce the chance of name conflicts.

Another problem is more challenging at first: User actions like mouse clicks are mapped to events like onclick. To receive the event, JavaScript code needs to register an event handler for it. Traditionally, AJAX applications achieve this by obtaining a reference for an object in the Document Object Model (DOM) tree which represents the element structure of the HTML document. Each object corresponds to an HTML element such as a button, and the code can subscribe to an event by overwriting certain methods of the object, for example the onclick method for mouse clicks.

Unfortunately, one implication of this event registration model is that only a single subscriber to any event is supported. If UsaProxy registered its own onclick and related methods for all objects in the DOM tree, any code of the web application which is loaded afterwards would overwrite that handler with its own method, and UsaProxy would be unable to track user behaviour for the respective element.

As a solution to this problem, two extended event registration models were added to browsers. They allow any number of interested parties to register for a particular event. Furthermore, they support the registration of an event handler not only for a single object, but also globally for all elements in the DOM tree or a subtree of it. The W3C version of the event registration uses the addEventListener() method and is supported by Netscape 6 and Safari/Konqueror, whereas Microsoft Internet Explorer only supports the alternative attachEvent() method. The Opera browser supports both variants. UsaProxy is intended to work with all the most popular browsers, it can use either method.

The UsaProxy logging code needs to be notified of many of the available events, and this needs to happen for all elements in the DOM tree. For this reason, it registers global event handlers which are later called in addition to any event handlers that the application may have registered. Via the object for which an event happens, access is possible to further useful data about the respective HTML element, such as its `id`, `href` or `src` attributes. Additionally, the object's position in the DOM tree can be determined. Via its event handlers, UsaProxy obtains the following information and causes it to be logged:

- Loading of new pages and (for some browsers only) closing of a window/tab with a page;
- Resizing of the browser window, with new window width and height in pixels;
- Whether the page or one of the elements on it gains or loses the input focus;
- Mouse movements, with pixel coordinates relative to the upper left corner of the page;
- Mouse clicks on elements, or hovering the mouse pointer over an element, with pixel coordinates.
- Scrolling inside the document, with vertical pixel offset from the top of the page;
- Keys that are pressed, including modifier keys like Shift.
- Changes made to the value of any field in a form, including radio buttons, drop-down menus, checkboxes and text fields.
- Text that is selected using the mouse or keyboard, either inside text fields or anywhere else on the page.

For mouse clicks and hovered-over elements, the element is identified in the log entry via any `id` `href` or `src` attributes it has. Additionally, for anchor tags the text of the anchor is logged. However, in some cases the element will not have been assigned an ID, so this information may not be enough to uniquely identify it in the document.

To account for this problem, the JavaScript code also creates a string which encodes the exact position of the element in the DOM tree: Starting from the root of the tree, each character specifies which *n*th child element must be selected to reach the node, where the character "`a`" stands for the first child, "`b`" for the second, etc. For example, the string "`aba`" indicates that the first element is selected (very likely `<html>`), then the second child (probably `<body>`), and then the first child element again (e.g. a `<h1>` heading at the start of the page). An extension of the syntax also allows this scheme to work for nodes with more than 26 children.

If the tracking code recorded every small change of the mouse position or the scroll offset, a significant amount of data would need to be sent back to the proxy. As the aim of the logging solution was to also support users with slow Internet connections, e.g. using a modem link, the amount of data is reduced by only recording changes of these values at most every 150 milliseconds.

Logging of Client-Side Interaction Data

The JavaScript part of UsaProxy's implementation collects log information in a string variable and sends it back to the proxy at regular intervals. This is required because there is no way for the

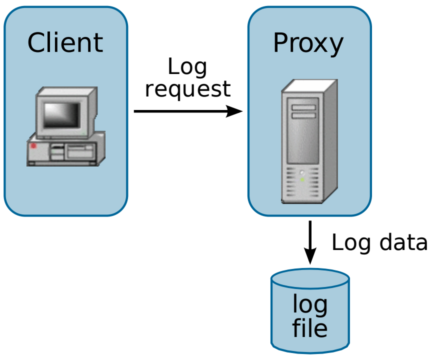


Figure 7.5: Log data is sent to the proxy by the JavaScript running on the client. The special form of the URL that is used for this causes the proxy not to forward it to any server.

script to store larger amounts of information on the client machine due to security restrictions. However, it is also a useful feature, since the proxy can conveniently store the logs for many test participants in a single, central place.

To deliver the log data, the same trick is used as for the initial download of the JavaScript code itself (see figure 7.5): A request to a special URL is made via an `XMLHttpRequest` invocation. The proxy recognizes this URL and does not forward it to a server, but instead writes a line to its log file. The line consists of the IP from which the log request originated, a timestamp and the submitted log data. The URL of the request now includes the string "`log`", followed by the text of the log line, i.e. `http://`*`a.b.c.d`*`/usaproxylolo/log?`*`log-data`*

The storage requirements for UsaProxy's log data are relatively modest even though the logging is far more detailed than that of server-side logs. In the example user studies that were conducted with the system and which involved viewing a number of pages on several websites, on average the amount of disk space that was needed for the recorded HTML documents was below 100 kBytes per user and minute, and the disk space for the log file was about 3 kBytes per user and minute.

7.5 Example Output

Figure 7.6 shows a short sample of a typical UsaProxy logfile. The log entries are a selection from a longer UsaProxy session and were chosen to include examples for many of the different possible log entry types.

As already explained above, the "`sd`" (session data) identifier in the lines of the log file is used to associate entries for HTTP traffic with entries created by the client-side JavaScript. Furthermore, a unique session ID "`sid`" is used to identify each browser instance. This way, several browsers accessing the same web application from the same IP can be distinguished from one another.

```
127.0.0.1 2008-01-14,00:05:40 httptraffic url=http://kiko.com/ sd=194
127.0.0.1 2008-01-14,00:05:40 sd=194 sid=lgHSEjeXpGYB event=load size=1272x800
127.0.0.1 2008-01-14,00:05:50 sd=194 sid=lgHSEjeXpGYB event=mousemove offset=358,62
  coord=624,267 dom=abababa img=kiko-example-calendar.jpg
127.0.0.1 2008-01-14,00:05:53 sd=194 sid=lgHSEjeXpGYB event=mousedown offset=44,18
  id=demo-link coord=638,133 dom=ababba link=http://kiko.com/account/demo text=Try
127.0.0.1 2008-01-14,00:05:59 httptraffic url=http://kiko.com/appt/application?user_id=229702
  sd=209
127.0.0.1 2008-01-14,00:06:09 sd=209 sid=lgHSEjeXpGYB event=mouseover
  id=calendar_display_wrapper_0 coord=1103,189 dom=abdbbb
127.0.0.1 2008-01-14,00:06:48 sd=209 sid=lgHSEjeXpGYB event=change type=text
  id=addSubgroupInput coord=121,314 dom=abeaaaaaedaa value=Testers
127.0.0.1 2008-01-14,00:06:48 sd=209 sid=lgHSEjeXpGYB event=blur id=addSubgroupInput
  coord=121,314 dom=abeaaaaaedaa
127.0.0.1 2008-01-14,00:08:29 sd=209 sid=lgHSEjeXpGYB event=keypress key=shift+U
127.0.0.1 2008-01-14,00:09:30 sd=209 sid=lgHSEjeXpGYB event=scroll y=012
```

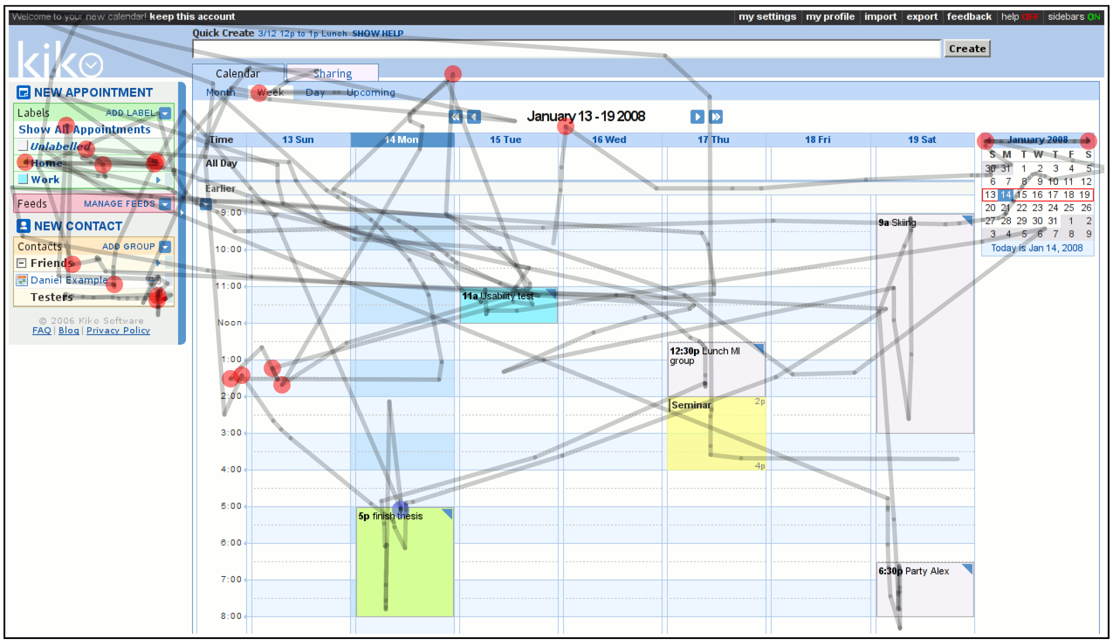
Figure 7.6: Sample of the log output generated by UsaProxy. httptraffic denotes HTTP requests passing through the proxy, the other lines are JavaScript-generated log events.

A simple visualisation of the log data from the same session is shown in figure 7.7. It was created by a small script which converts the log data into an SVG image. The background is based on a screenshot of the AJAX application that was visited. Overlayed on it is all data from the logfile which includes pixel-based mouse coordinates. Mouse movement is represented with grey lines, clicks with red circles. It should be noted that if the user had scrolled the page during the use of the web application, this would not have affected the accuracy of the visualisation, as UsaProxy records page-absolute mouse coordinates rather than screen coordinates.

The example in the figure is only one possible way the log data can be summarized. In fact, for large user studies with many users, other parts of the log which are not visualised in the figure will often be more useful. For example, this includes the information on *which* HTML elements were interacted with, and the timestamps of the interaction. They can be used for purposes such as measuring the time taken to achieve goals. In a similar vein, it is possible to calculate the total amount of time a user was reading the page, rather than the time it was open in a browser window. This can be achieved by assuming that scroll events or small mouse movements will occur frequently while the page is being read, and only accumulating the duration of such activity.

7.6 Evaluation

Summary: In a number of studies, different scenarios for usability tests were tried out with UsaProxy-based logging. In most cases, the required information could be extracted easily from the log. Because no client-side changes are necessary, Usa-Proxy also appears to be a good logging technology for remote tests.



Source website: `http://kiko.com`, accessed 13 January 2008

Figure 7.7: Visualisation of the log data for a session with an AJAX application. Information with pixel coordinates, like mouse movement and clicks, is overlayed on a screenshot.

In order to verify whether UsaProxy is suited for the logging of website usage, a number of small example user tests were conducted. Apart from demonstrating that the deployment modes of the system worked correctly, different test scenarios served to show that the logging functionality is flexible enough for user studies in which users visit a number of different websites, or in which they make use of advanced AJAX applications.

In the first user study [Atterer06UsaProxy], 12 test participants used a prepared computer whose web browser had been reconfigured to access the web via UsaProxy. They were experienced web users with a technical background (10 students of media informatics, 2 members of the media informatics staff). The second user study [Atterer07TrackingAJAX] was conducted with 7 participants with a similar background. Two different deployment methods were tried: On one hand, UsaProxy's transparent HTTP proxy mode was used – a machine with two Ethernet ports was inserted between the test subject's normal desktop computer and the Internet to filter all HTTP traffic. On the other hand, the test subjects were given instructions on how to reconfigure their browser. In both cases, no problems were encountered. With exact instructions on where and how the proxy setting could be found, all users were successful in changing it.

Tracking Users on Multiple Websites

As part of the first user study, the test subjects were given the following task:

Starting from the Wikipedia homepage, find the FAQ entry on how to insert an image on a Wikipedia page.

The task consisted first of finding the right page on the site (which is possible in a number of ways), and then scrolling to the correct location on the very long FAQ page. With server-based logs, the latter step cannot be verified, whereas the UsaProxy logs allowed for it.

One out of the 12 participants failed to complete the task. For the others, the average time taken to achieve the task could be determined, including scrolling to the correct page position. A look at the the navigation paths reveals that only 4 users took the optimal path. For those users who entered a search string in Wikipedia's internal search form, the string is also contained in the log. Finally, one user left the Wikipedia website altogether and relied on Google search instead. The proxy continued to track him when he left and re-entered the Wikipedia site, and also recorded his Google search query.

For the second study, a task involving several websites was intentionally chosen. A user test with such a task could be interesting e.g. to analyse the overall tactics used by web surfers when they look for information. The task was:

> *You want to buy a digital camera for at most $300. Search the WWW to compare camera models, then put your favourite camera into a shopping basket on Amazon.*

While the users needed to visit the Amazon website to complete the task, the instructions did not place restrictions on the websites to use for the first "information gathering" phase. Accordingly, many participants started with a general search engine query. In those cases where they opened several browser windows/tabs in quick succession, it was possible to determine the time spent scanning or reading each page by looking for mouse or scroll actions. A script was written to automatically extract for each user the total time spent on every visited site. UsaProxy was running as a transparent proxy for this task.

Recording Interaction With AJAX Applications

UsaProxy was also tested with complex AJAX applications. Particular areas of interest during these tests were whether the user experience was changed, and whether the proxy's JavaScript interfered with the correct execution of the AJAX application. In both cases, the logging system worked as expected; its use made no difference to the user experience, and all JavaScript functionality on the web application continued to work.

In the first study, the 12 subjects were asked to use the online calendar application Kiko, `http://kiko.com`. Figure 7.7 shows a screenshot of the current version of this AJAX application, the test was performed with an older version. The users were given the following task:

> *Set up the user group "Testers" and an appointment titled "usability test" which should take place on Tuesday from 11 to 12. The Kiko application will automatically create a demo user in that group, who should be the only participant of the appointment.*

The level of detail of the log was sufficient to determine the users' actions despite the fact that the calendar application only consisted of one HTML page and all functionality was implemented

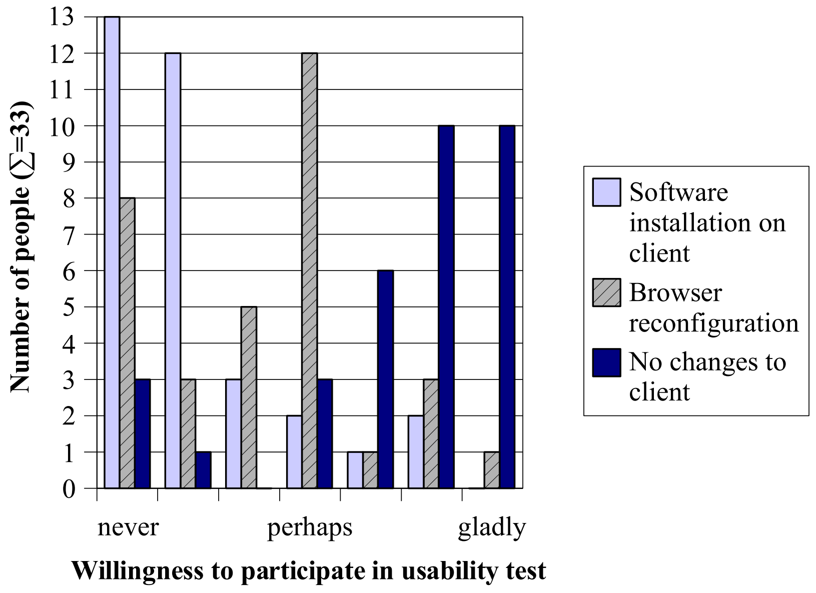


Figure 7.8: Results of a survey: The likelyhood of users to participate in remote website usability tests depends on the technology that is used to record mouse actions and key presses.

with JavaScript. For mouse hovering and clicks, the involved GUI elements can be identified easily either using their ID or (if no ID was assigned by the page author) via their position in the DOM tree. The logging works both for elements which are delivered as part of the HTML page and with dynamically generated elements. In the second study, a similar task was repeated with other complex AJAX applications, Google Calendar and Mail, with comparable results. Since the login procedure for Google involves the use of HTTPS, logging was only started after the users were logged into a special account which had been created for the test.

Finally, the second study also included a graphics-intensive AJAX application in the form of an online map service. The users were asked to interact with the map:

> *In the new* [i.e., AJAX] *version of* `stadtplandienst.de`*, identify those streets in Munich which form a ring about the city centre. Enter their names in the search field to have them highlighted.*

For this type of task, it becomes difficult to extract meaningful data from the interaction log, as the map application makes use of a number of anonymous tiles which contain graphics. Still, an analysis of the log data showed that it would be possible with some effort to e.g. obtain the coordinates on the map that a user clicked on. In practice, it will often be easier to resort to screen capturing instead of AJAX-based logging for an application like this.

Can We Recruit Test Participants on the WWW?

In previous sections, the possibility of inviting test users over the Internet has been mentioned as a possible application area for UsaProxy. For example, the regular users of a service could be asked to take part in the test of a new beta version of the web application. However, it is not clear whether average web users are willing to do so in practice. The assumed reason why users would shy away from participating in such a test was that they would not be inclined to install software on their machines to allow their actions to be tracked.

A small web-based questionnaire which was completed by 33 subjects supports this theory. While the number of samples is not high enough to allow for definite conclusions, it is sufficient to give a general idea about the possible effects of the tracking technology on participation rates in remote web-based user studies. Most of the subjects were aged between 25 and 32 and had a technology-related background. For each question in the questionnaire, they were able to choose between strong disagreement and strong agreement on a 7-point Likert scale.

The questionnaire began with a few general questions. First, the subjects were asked whether they would have privacy concerns if their mouse actions, key presses etc. were logged by a web page. About two thirds agreed with this statement (i.e. chose one of the 3 alternatives on the Likert scale which are nearest to “fully agree”). However, the next question restricted this scenario: If only a single website would employ logging temporarily during a web-based usability test, and a small reward would be offered in return (such as free access to an online service), then 23 of the 33 subjects indicated that they might participate in the usability test (3 neutral, 7 would rather not participate). Thus, users do not categorically reject the idea of web-based tests.

The next questions asked for the users’ acceptance of different logging technologies for a web-based usability test. They were asked whether they would participate in the test

1. if they had to install tracking software on their computer prior to the test,
2. if they had to reconfigure their browser before the test, and
3. if they did not have to change anything about their setup.

The results of these three questions are visualized in figure 7.8. They clearly show that for the sample of people who completed the questionnaire, the type of tracking technology has an influence on acceptance: When software needs to be installed, the median value on the Likert scale (from 1=“would never” to 7=“would gladly participate in user tests”) is 2, for browser reconfiguration it is 4 and for no changes to the client it is 6. These results suggest that JavaScript-based logging with a tool like UsaProxy is the preferable way to obtain usage information during a web-based remote usability test.

7.7 Conclusion

This chapter has described UsaProxy, a system for detailed logging of user actions on web pages. The technology indirectly helps to improve the usability of websites by reducing the effort of conducting a user test, possibly as a remote test with users that are recruited from the normal visitors of a website.

The log data generated by UsaProxy can form the basis for further analysis by automated tools: A model-based tool like Wusab can compare the data to information from web engineering models and report any inconsistencies. For example, the model can specify that all pages should load within 10 seconds, but the actual load times measured by UsaProxy could be higher for a significant portion of users. In a similar way, the analysis could highlight a situation where the number of clicks of the navigation paths from one page to another is higher in practice than what is suggested by the navigation model.

Other systems exist which allow user actions to be tracked on web pages. However, in comparison to these previous efforts, UsaProxy permits mouse movements, key presses and many other types of interaction to be logged even for highly dynamic AJAX applications. It supports a number of different scenarios, including one where the developer does not need to have access to the web server or to prepare its pages manually for a user test, and one where no changes to the client-side setup are necessary. Furthermore, the log data produced by the tool is detailed and easy to analyse in the sense that it does not only include pixel coordinates, but identifies individual buttons, links etc. the user interacted with on a page.

Its extensible architecture allows UsaProxy to be adapted for other projects which no longer concentrate on logging alone, but use the logged data for advanced interaction concepts. Beyond the area of usability testing, UsaProxy can be one of the building blocks of self-adapting websites, live customer support on a website, shared browsing sessions [Atterer07AppSharing], and collaborative annotation of web pages.

However, the JavaScript-based tracking technology also raises privacy concerns: With its use, the actions of regular visitors of a web page could theoretically be observed in great detail without their knowledge. When the logging approach is abused in this way, personal information about the users could easily be revealed by them unintentionally. For instance, mouse movement data can suggest that a user has bad motor skills, which could mean that he is elderly or disabled. Another example is the entry of confidential information: It is possible that a user starts typing a password to log into a website, but then notices that it is the password for a different site. Even if he deletes the password without submitting the login form, his key presses will already have been sent to the remote server.

Due to these dangers, it is strongly recommended that JavaScript-based logging is not employed without first explaining to the user how detailed the logs are, and not without their express permission. Even then, their consent should only be considered valid for a few hours or until the end of their browser session.

Chapter 8

Conclusions and Future Work

This chapter summarizes the advances made by this thesis in the area of web engineering. It lists the contributions made to solve the problems from chapter 1, and contains descriptions on how the claimed improvements were achieved. Finally, it outlines areas of future work.

8.1 General Conclusion

Summary: The thesis successfully combines insights from the areas of web engineering and human-computer interaction. It proposes changes to the development process as well as new tool concepts and respective changes to the models.

The thesis describes a catalogue of measures, particularly in the area of tool support, which can be applied to the development process of web applications in order to improve the accessibility and usability of the final application. Two areas of research are used as a source of inspiration: On one hand, the field of human-computer interaction focuses on the user interface of computer programs. Apart from promoting procedures like user testing, research in the field provides the basis for guideline documents which describe best practices for the design of (among others) web-based UIs. On the other hand, the model-driven development techniques found in the area of web engineering can result in more consistency, better code quality and higher developer productivity if applied correctly. So far, few attempts have been made to combine aspects of both approaches, but as shown by this thesis, each field can benefit from advances made in the other one.

Improvements to the Web Development Process and Tool Support

For the overall development process, HCI literature like [Constantine99Software] or [Rosson02UsabEng] provides method steps which ensure that the final application will be usable. Many of them, such as guideline reviews, paper prototyping or user testing involve significant amounts of manual work, and automation is not a topic in the HCI literature. Chapter 4 of this work highlights ways in which tool support for these usability-related tasks can be created, systematically covering all phases of development from requirements analysis to testing. The presented tool concepts promise to result both in better usability for the end users and less work for the developer. There is a conflict between the human-centered development concepts in HCI and the goal of tools to automate tasks without human intervention. The thesis resolves it by automating only those parts of the process where human involvement is not directly necessary.

Section 4.6 summarizes the central insights regarding automated usability tool support for model-based web development in a list of five major principles. In later chapters, more details and implementations are provided for two different tool concepts. Together, they cover all five principles, thus highlighting the different angles from which the topic of usability-related tool support can be approached.

Extending Web Engineering Models With Information About the Context of Use

As a prerequisite for many tool concepts, it is necessary to know the context in which the developed website is used. Based on previous work in this area (in particular [Limbourg04UsiXML]), the thesis identifies three major parts which form the context of use: The intended audience (e.g. teenagers), the technical platform that should be supported (e.g. mobile phones), and the environment in which the application is used (e.g. noisy). The models presented in chapter 5 provide means for the developer to specify the different properties of a context. The models of existing web engineering approaches do not include most of this information on how the audience perceives and operates the web application. One advantage of the new models is that they are presented as extensions to existing web engineering models, so their use (including being interpreted by automated tools) is not restricted to a particular web engineering solution.

Model information is a crucial element when improving the quality and accuracy of automated tools. For this reason, section 5.4 of the thesis also explores ways of making the information from the proposed model extensions available to tools in easy-to-access formats, embedded in the web application implementation.

The models also have a purpose beyond providing information to tools: When the developer creates them, it encourages him to think about the possible ways in which his application might be used by different users under various circumstances. This way, the thesis also accounts for the fact that tool support for usability improvements will be more effective if the developer is made aware of the importance of a well-designed user interface.

Prototype Implementations For Tool Concepts: Guideline Validation, User Testing

The two prototypes introduced in chapters 6 and 7 advance the state of the art both by demonstrating the general feasibility of usability tool support for web engineering, and by providing improvements in their specific application area compared to earlier, similar tools.

The tool Wusab (chapter 6) is an automated validator of usability and accessibility guidelines. It is intended to be run as an embedded component in an integrated web engineering development environment, e.g. alerting the developer the moment he is about to make a problematic change to the web application. Compared to existent tools which only analyse the final implementation of the application (e.g. [Kasday00WAVE], [Abascal04EvalIris]), the Wusab tool concept not only includes inspection of a site's HTML code, but also of the models which describe the website, and of logs that are produced during user testing or by the final, live website. This considerably larger amount of input permits the implementation of guideline checks which describe problems more accurately than non-model-based tools, and which are less prone to produce false positives or false negatives.

UsaProxy (chapter 7) is a tool for website interaction logging and usage analysis. It is capable of creating a detailed log of all actions performed by a user on a web page, including interaction with AJAX-based services. The tool is primarily intended to be used for user testing and to facilitate quick analysis of the test results, such as the time taken by participants to complete the task they are given. A number of related programs supports recording of the user's actions, ranging from solutions which require installation of software (e.g. [Reeder01WebEye]) to approaches which use URL rewriting to record visited URLs (e.g. [Hong01WebQuilt]). In comparison, UsaProxy has unique features which make it better suited for tasks like user testing. Apart from its flexible, minimally invasive architecture (with HTTP proxy, transparent proxy and server modes), the log data it produces is very detailed. For example, it includes information on the exact buttons or links that are used. This makes the log much easier to interpret than the mouse coordinates that are typically output by other solutions. The tool's flexibility results in a number of possible areas of use, including web application debugging and remote user testing.

8.2 Contributions of the Thesis

Summary: Chapter 1 has described the motivation for this work and the approach used to solve the problems that were identified. The following sections list these problems and the contributions made by the thesis to solve them.

Problems That Were Identified

- *Existing web usability/accessibility guidelines were not written with tool support in mind.*

  Most guideline documents which describe desirable properties of a usable website use wording which requires human judgement (section 2.2). It is hard to create algorithms which validate correctly whether such a guideline is met by a particular website. Existing validators mostly concentrate on accessibility and skip more advanced tests (section 2.3).

- *Existing web engineering development methods do not focus on usability.*

  While development methods which ensure good usability have been formulated in the field of human-computer interaction, there has only been a limited amount of interest to apply them to web engineering (section 2.3). It is possible to simply employ procedures like paper prototyping or user testing during web development, but this is only one step in the direction of integrating HCI practices in web engineering.

- *Current web engineering tools do not actively support the creation of usable websites.*

The tools of current model-based web engineering solutions allow the creation of usable sites, but they do not encourage it in any particular way. Instead, they concentrate on things like automatic generation of an implementation from models (section 3.2).

- *Existing web engineering models cannot express information relevant for usability.*

  The models of current web engineering approaches focus on modelling the web application itself. For most of the published approaches, they do not permit the detailed description of properties of the user or the way he accesses the web page. While such models exist (section 5.1), they have not been integrated with web engineering for the purpose of describing the intended audience and the situation in which it interacts with a site.

Contributions Which Solve These Problems

- *The thesis introduces concepts for web engineering tool support which aim at improving the usability of websites.*

  The work in chapter 4 goes beyond simply applying practices from the field of human-computer interaction to web engineering. Instead, the positive aspects of both fields are combined: HCI practices improve usability, but these improvements are mostly achieved through additional manual work. In contrast to this, the model-driven architecture that is prevalent in web engineering strongly relies on automated processes during development. The thesis successfully addresses this contradiction by introducing tool concepts for all phases of development that can help to improve usability. Five basic principles of usability tool support for model-based web development (section 4.6) summarize the thoughts behind the proposed tool concepts.

- *An extension for web engineering models is presented to allow the description of the users, platform and environment of web application use.*

  While approaches exist that model these factors, their application to the area of web engineering for the purpose of usability-related tool support is new. Information on how the audience perceives and interacts with the website, and on aspects such as the users' intentions, are a prerequisite for many of the tool concepts introduced by the thesis. The model extensions in chapter 5 are designed to support these concepts. The design also makes them universally applicable, allowing them to be integrated with the other models of existing web engineering approaches (section 5.3). Finally, the models also serve to make the developer consider the needs of the users of a web application. This is an important part of a user-oriented development process, as adequate usability cannot be ensured by tool support alone.

- *Embedding of model information in a web application's implementation is presented as a promising technique for new tool concepts.*

  The proposed procedure of embedding model information in the final HTML code (section 5.4) ensures that it will be available to tools at a later time. The thesis primarily advocates it to allow model-based usability validators (see next point below). Beyond the purposes of this thesis, the idea that the model data can be permanently and publically available to arbitrary site visitors after the end of development is promising in the context of the “semantic web”: Machine-readable metadata becomes available as a by-product of regular web development. (See also section 8.3 below.)

- *A novel tool concept is introduced, the model-based automated validation of usability guidelines for web pages.*

  If not only the HTML code of a website is available to an automated tool which creates a report of possible accessibility and usability problems, but also the model information, the accuracy and quality of the automated validation can be improved considerably (sections 4.3.1 and 6.4). This idea is further extended in section 4.5.1: In addition to the implementation and the models, usage data from user testing or the final, live website can also be made available to the validator. The latter aspect also makes it clear again that usability tool support should not be regarded as part of the analysis/design phase of a project, but that it needs to extend through the entire process (including the maintenance phase) to be truly effective.

- *For model-based usability validation of web pages, a number of example validation algorithms are presented (Wusab prototype)*

  Chapter 6 presents heuristics for the validation of a number of usability guidelines. Existing tools that only look at a website’s HTML code do not support these guidelines, but by applying the approach mentioned above and using models as input, their implementation becomes feasible.

- *The state of the art is advanced by the thesis in the area of automated web site usage tracking (UsaProxy prototype)*

  When recording and analysing the results of usability tests, the insufficient accuracy of logs output by current tools can cause additional manual work by the developer. The prototype presented in chapter 7 is an improvement in this area because of its support for identifying the exact page elements the user interacts with, and the tracking of users of AJAX applications. Due to its straightforward deployment, usability testing can even be conducted remotely, without setting up a dedicated usability lab.

8.3 Future Work

Summary: In a number of areas, this work leaves room for future improvements and advances. In the following, the most interesting ones are described.

The following proposed areas of future work are either concerned with questions which are beyond the scope of the thesis, or result from ideas encountered during the work on usability tool support which only touch upon the main thesis topic, and which were thus not pursued.

Automated Usability/Accessibility Analysis for AJAX Websites

In the case of the model-based validator Wusab (chapter 6), the evaluation of complex AJAX applications is not a goal of the tool concept – the validator restricts itself to web applications which employ at most a limited amount of JavaScript-based dynamic HTML effects.

This restriction is due to the fact that JavaScript offers a much wider range of possibilities for the implementation of user interfaces, making automatic analysis very difficult. For HTML, the set of interaction possibilities is fairly small. Essentially, the user traverses from one URL to another in different ways, such as clicking on links, pressing a button or providing form input. However, with JavaScript-based UIs, every mouse movement or other minor event could be significant input, and the response of the application to this input is also not constrained in any way.

A possible solution to this problem, and an area of future work, would be to rely on "AJAX user interface patterns" for client-side UI code. Rather than implementing the client-side user interface manually, these patterns (e.g. using drag & drop to move an item from one part of the user interface to another) would be included in the models for the user interface, and would describe the interaction in a more abstract way. Automated usability evaluation could then analyse them in a way that is similar to Wusab's approach for non-AJAX sites.

In the area of accessibility, the W3C specifications for Accessible Rich Internet Applications target the same problem – the ideas outlined in the project's roadmap [W3C-ARIA] could serve as a source of inspiration.

Methodology for the Comparison of Usability Validators

As shown in section 6.4, while the newly developed tool concept of model-based usability validation is promising, it is hard to come up with a fair, objective methodology to compare any two validator implementations (e.g. a model-based vs. a non-model-based one). Many factors influence how useful a validator is: It might require additional work (e.g. annotation of the HTML pages with metadata) to produce higher-quality results, or trade more accurate reports for a higher number of false positives. Additionally, any tests of validators with real-life web pages will yield different results depending on the selected sample of inputs. Finally, some proof-of-concept tools might be conceptually superior, but will only have implementations for a few selected guideline tests compared to the more complete test coverage of conventional tools.

It seems unlikely that the architecture and results of different validators can be directly compared using algorithmic means alone. The classification of evaluation methods by Ivory (section 2.3) provides the basis for the comparison of validator architectures. For other aspects of validators, such as ease-of-use or accuracy, procedures like a special type of user test might prove useful.

Applications for the Use of Model Data Embedded in HTML Pages

After introducing the idea of embedding model information directly in generated web pages in section 5.4, this thesis has hinted at the possibility of using this information for other purposes than model-based usability validation (chapter 6).

One example of such use of the published model information would be a browser extension that automatically downloads and interprets model data, and that uses it to provide additional navigation facilities, change the layout, or to make changes that improve usability for persons who are not among the expected target audience of the site. Alternatively, if more widespread use of embedded meta information is assumed, it becomes possible to create services which download and analyse model data from many different sites, in order to visualise it or to provide additional functionality which goes beyond that of the individual sites.

Role of Usability Models During Model Transformations

In chapter 4, this work mentions numerous examples for MDA-related tools whose use can lead to better usability of the developed website. In section 4.2.2, the idea of usable defaults for model transformations and code generation is discussed: If the created platform-specific models and implementation are "usable by default", it is also likely that the usability of the final web application will be improved, as the developer will often rely on the defaults supplied by his tools and only change them in a limited number of areas.

As soon as context information about the audience, technical platform and environment (section 5.2) is present in the models, it is a natural step to include it in the model transformation process to ensure that the "usable by default" requirement is more likely to be met. For example, the transformation could consult the user model and generate CSS declarations with high text-background contrast and large fonts if the expected users of the application are elderly.

This idea was not further pursued in favour of the tool concepts from chapters 6 and 7. However, future work in this area is possible. As outlined above, at the simplest level a transformation algorithm decides about the effects of properties in a usability-related model. A more advanced concept would also allow information to flow in the opposite direction: Whenever a change is made to a more specific model (such as a reduction of font size in the implementation's CSS), the more abstract model can be adjusted accordingly (the site is no longer suitable for elderly persons). In practice, such changes to the model should only be performed after alerting the developer and obtaining his permission. This way, the resulting tool concept would be an extension of Wusab from chapter 6. The extended tool would not only highlight inconsistencies, but also provide ways to interactively solve them.

Run-Time Content Adaptation For Improved Usability

The models from chapter 5 are introduced as a way for the developer to specify *static* expected properties of the context in which a web application is used. Approaches like UsiXML [Limbourg04UsiXML] have already provided ways to permit the specification of context changes during the time the application executes, together with the corresponding changes in the implementation. Primarily, this kind of adaptation is useful for the environment (which can e.g. be noisy only some of the time), but it can also make sense for the user (whose level of stress might vary) or the device (which might e.g. feature an attachable keyboard which is not present all the time). Currently, web applications typically do not feature this kind of fine-grained adaptation mechanism. Extending the existing models and tools to allow it would be an interesting challenge.

Bibliography

[Abascal03EvalIris] J. Abascal, M. Arrue, N. Garay, J. Tomás: EvalIris – A Web Service for Web Accessibility Evaluation. In *Proceedings of the 12th International World Wide Web Conference* (poster), Budapest, Hungary, 20–24 May 2003.

[Abascal04EvalIris] J. Abascal, M. Arrue, N. Garay, J. Tomás: The use of guidelines to automatically verify Web accessibility. In *Universal Access in the Information Society*, volume 3, issue 1, March 2004, pages 71–79, ISSN 1615-5289

[Adkisson02DeFacto] H. Adkisson: Identifying De-Facto Standards for E-Commerce Web Sites. Master's thesis, University of Washington, 2002

[Albrecht00SmexWeb] F. Albrecht, N. Koch, T. Tiller: SmexWeb: An Adaptive Web-based Hypermedia Teaching System. In *Journal of Interactive Learning Research*, Special Issue on Intelligent Systems/Tools in Training and Life-Long Learning, 11(3/4), 2000

[Anupam00WebVCR] V. Anupam, J. Freire, B. Kumar, D. Lieuwen: Automating Web Navigation with the WebVCR. In *Proceedings of the 9th International World Wide Web Conference WWW9*, Amsterdam, Netherlands, May 2000

[Arroyo06MouseTrack] E. Arroyo, T. Selker, W. Wei: Usability Tool for Analysis of Web Designs Using Mouse Tracks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2006 (CHI2006)* (work in progress), Montréal, Québec, Canada, April 2006

[Atterer05ICWE] R. Atterer, A. Schmidt: Adding Usability to Web Engineering Models and Tools. In *Proceedings of the 5th International Conference on Web Engineering ICWE 2005*, Sydney, Australia, July 2005, pages 36–41, Springer LNCS 3579

[Atterer05WebToolSupport] R. Atterer: Where Web Engineering Tool Support Ends: Building Usable Websites. In *Proceedings of the 20th Annual ACM Symposium on Applied Computing*, Santa Fe, New Mexico, USA, 12–17 March 2005

[Atterer06JWE] R. Atterer, A. Schmidt, H. Hußmann: Extending Web Engineering Models and Tools for Automatic Usability Validation. In *Journal of Web Engineering*, volume 5, number 1 (2006), pages 43–64

[Atterer06LoggingAJAX] R. Atterer: Logging Usage of AJAX Applications With the "UsaProxy" HTTP Proxy. In *Proceedings of the WWW 2006 Workshop on Logging Traces of Web Activity: The Mechanics of Data Collection*, Edinburgh, Scotland, May 2006

[Atterer06UsaProxy] R. Atterer, M. Wnuk, A. Schmidt: Knowing the User's Every Move – User Activity Tracking for Website Usability Evaluation and Implicit Interaction. In *Proceedings of the 15th International World Wide Web Conference (WWW2006)*, Edinburgh, Scotland, May 2006

[Atterer07AppSharing] R. Atterer, A. Schmidt, M. Wnuk: A Proxy-Based Infrastructure for Web Application Sharing and Remote Collaboration on Web Pages. In *Proceedings of the 11th IFIP TC13 International Conference on Human-Computer Interaction (INTERACT 2007)*, Rio de Janeiro, Brazil, September 2007

[Atterer07TrackingAJAX] R. Atterer, A. Schmidt: Tracking the Interaction of Users with AJAX Applications for Usability Testing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2007 (CHI2007)*, San Jose, CA, USA, April/May 2007, pages 1347–1350

[Bailey03Informal] B. Bailey, J. Konstan: Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2003 (CHI2003)*, Fort Lauderdale, Florida, USA, April 2003, pages 313–320

[Balbo95AutoEval] S. Balbo: Automatic evaluation of user interface usability: Dream or reality. In *Proceedings of the Queensland Computer-Human Interaction Symposium*, Queensland, Australia, August 1995
Referenced in [Ivory03WebEval, page 16]

[Balzert96SE] H. Balzert: Lehrbuch der Softwaretechnik: Software-Entwicklung. Spektrum Akademischer Verlag, Heidelberg, Berlin, Oxford, 1996, ISBN 3-8274-0042-2

[Baresi01W2000] L. Baresi, F. Garzotto, P. Paolini: Extending UML for Modeling Web Applications. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, Island of Maui, IEEE Computer Society, January 2001

[Barret99Intermediaries] R. Barrett, P. P. Maglio: Intermediaries: An approach to manipulating information streams. In *IBM Systems Journal archive*, volume 38, issue 4, December 1999, pages 629–641

[Baumeister05Adaptivity] H. Baumeister, A. Knapp, N. Koch, G. Zhang: Modelling Adaptivity with Aspects. In *Proceedings of the 5th International Conference on Web*

Engineering ICWE 2005, Sydney, Australia, July 2005, pages 406–416, Springer LNCS 3579

[Beirekdar02Kwaresmi] A. Beirekdar, J. Vanderdonckt, M. Noirhomme-Fraiture: A Framework and a Language for Usability Automatic Evaluation of Web Sites by Static Analysis of HTML Source Code. In *Proceedings of 4th International Conference on Computer-Aided Design of User Interfaces CADUI 2002*, Valenciennes, France, May 2002

[Benyon93UserModeling] D. Benyon, D. Murray: Applying user modelling to human-computer interaction design. In *Artificial Intelligence Review*, volume 7, numbers 3–4, August 1993, pages 199–225, ISSN 0269-2821, Springer Netherlands

[Bernard01Developing] M. Bernard: Developing Schemas for the Location of Common Web Objects. Usability News 3.1, 2001, `http://psychology.wichita.edu/surl/usabilitynews/3W/Web_object.htm`, accessed 2 January 2008

[BITV02] Bundesgesetzblatt: Verordnung zur Schaffung barrierefreier Informationstechnik nach dem Behindertengleichstellungsgesetz (Barrierefreie Informationstechnik-Verordnung – BITV). BGBl I Nr. 49, Bundesanzeiger-Verlag, Cologne, Germany, 23 July 2002

[Booch99UMLUserGuide] G. Booch, J. Rumbaugh, I. Jacobson: The Unified Modeling Language user guide. Addison-Wesley, Redwood City, CA, USA, 1999, ISBN 0-201-57168-4

[Bozzon06ConceptualRIA] A. Bozzon, S. Comai: Conceptual Modeling and Code Generation for Rich Internet Applications. In *Proceedings of the 6th International Conference on Web Engineering (ICWE 2006)*, Palo Alto, CA, USA, July 2006

[Brajnik00AutoWebUsab] G. Brajnik: Automatic web usability evaluation: what needs to be done? In *Proceedings of the 6th Human Factors and the Web Conference*, Austin, Texas, June 2000

[Brajnik04Comparing] G. Brajnik: Comparing accessibility evaluation tools: a method for tool effectiveness. In *Universal Access in the Information Society* journal, volume 3, numbers 3–4, Springer Verlag, October 2004

[Brettel97Color] H. Brettel, F. Vienot, J. D. Mollon: Computerized simulation of color appearance for dichromats. In *Journal of the Optical Society of America A*, volume 14, number 10, 1997, pages 2647–2655, ISSN 0740-3232

[Brush04Comparison] A. J. Bernheim Brush, M. Ames, J. Davis: A Comparison of Synchronous Remote and Local Usability Studies for an Expert Interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2004 (CHI2004)*, Vienna, Austria, April 2004, pages 1179–1182

[Cachero00PresOOH] C. Cachero, J. Gómez, O. Pastor: Object-Oriented Conceptual Modeling of Web Application Interfaces: the OO-HMethod Abstract Presentation Model. In *Proceedings of the 1st International Conference on Electronic Commerce and Web Technologies EC-Web 2000*, London, UK, September 2000, pages 206–215, Springer LNCS 1875

[Calvary03MultiTarget] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, J. Vanderdonckt: A Unifying Reference Framework for multi-target user interfaces. In *Interacting with Computers*, volume 15, issue 3, June 2003, pages 289–308

[Card01InfoScent] S. K. Card, P. Pirolli, M. Van Der Wege, J. B. Morrison, R. W. Reeder, P. K. Schraedley, J. Boshart: Information Scent as a Driver of Web Behavior Graphs: Results of a Protocol Analysis Method for Web Usability. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2001 (CHI2001)* (extended abstract), Seattle, Washington, USA, April 2001, pages 498–505

[Centeno06WebCompWCAG] V. L. Centeno, C. D. Kloos, M. Gaedke, M. Nussbaumer: Web Composition with WCAG in mind. In *Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A)*, held at the 14th International World Wide Web Conference (WWW14), Chiba, Japan, 2005, pages 38–45

[Ceri00WebML] S. Ceri, P. Fraternali, A. Bongio: Web Modeling Language (WebML): a modeling language for designing Web sites. In *Proceedings of the 9th International World-Wide Web Conference (WWW9)*, Amsterdam, The Netherlands, May 2000

[Ceri03WebML] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, M. Matera: Designing Data-Intensive Web Applications. Elsevier Science, San Francisco, CA, USA, 2003, ISBN 1-55860-843-5

[Ceri07MDDContext] S. Ceri, F. Daniel, M. Matera, F. M. Facca: Model-driven Development of Context-Aware Web Applications. In *ACM Transactions on Internet Technology (TOIT)*, volume 7, issue 1, February 2007

[Chen01Correlation] Mon-Chu Chen, John R. Anderson, Myeong-Ho Sohn: What can a mouse cursor tell us more? Correlation of eye/mouse movements on web browsing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2001 (CHI2001)* (extended abstract), Seattle, Washington, USA, April 2001, pages 281–282

[Chen04SOUPA] H. Chen, F. Perich, T. Finin, A. Joshi: SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In *Proceedings of the International*

Conference on Mobile and Ubiquitous Systems: Networking and Services. Boston, MA, USA, August 2004

[Chi00SiteScent] E. H. Chi, P. Pirolli, J. Pitkow: The Scent of a Site: A System for Analyzing and Predicting Information Scent, Usage, and Usability of a Web Site. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2000 (CHI2000)*, The Hague, The Netherlands, April 2000, pages 161–168

[Conallen99WebUML] J. Conallen: Modeling Web application architectures with UML. In *Communications of the ACM*, volume 42, issue 10, October 1999, pages 63–70

[Constantine99Software] L. Constantine, L. Lockwood: Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design. ACM Press/Addison-Wesley, New York, NY, USA, 1999

[Correani04Nauticus] F. Correani, B. Leporini, F. Paternò: Supporting Web Usability for Vision Impaired Users. In *User-Centered Interaction Paradigms for Universal Access in the Information Society: 8th ERCIM Workshop on User Interfaces for All*, Vienna, Austria, June 2004, Springer LNCS 3196, pages 242–253

[Coxhead00AWL] A. Coxhead: A New Academic Word List. In *TESOL Quarterly*, volume 34, number 2, summer 2000, pages 213–238

[Deshpande02WE] Y. Deshpande, S. Murugesan, A. Ginige, S. Hansen, D. Schwabe, M. Gaedke, B. White: Web Engineering. In *Journal of Web Engineering*, volume 1, number 1 (2002), pages 3–17

[DeTroyer00WSDM] O. De Troyer, T. Decruyenaere: Conceptual Modelling of Web Sites for End-Users. In *WWW Journal*, volume 3, issue 1, Baltzer Science Publishers, 2000.

[Dix04HCI] A. Dix, J. Finlay, G. D. Abowd, R. Beale: Human-Computer Interaction, 3rd edition. Pearson Education/Prentice Hall, Harlow, UK, 2004

[Drewes07Detailed] H. Drewes, R. Atterer, A. Schmidt: Detailed Monitoring of User's Gaze and Interaction to Improve Future E-Learning. In *Proceedings of the 12th International Conference on Human-Computer Interaction (HCII '07)*, Beijing, China, Springer LNCS 4555, July 2007

[DuBay04Readability] W. H. DuBay: The Principles of Readability, Impact Information, 2004, `http://www.impact-information.com/impactinfo/readability02.pdf`, accessed 31 December 2007

[Dumke03WE] R. Dumke, M. Lother, C. Wille, F. Zbrog: Web Engineering. Pearson Studium München, 2003, ISBN 3-8273-7080-9

[Ecker06SemiAuto] R. Ecker: Semi-automatische Usability-Analyse des Layouts von Webseiten. Project thesis, Media Informatics Group, University of Munich, November 2006

[Eimeren06OnlineStudy] B. van Eimeren, B. Frees: Schnelle Zugänge, neue Anwendungen, neue Nutzer? ARD/ZDF-Online-Studie 2006. In *Media Perspektiven* 8/2006, Frankfurt/Main, Germany

[Florescu98DB] D. Florescu, A. Levy, A. Mendelzon: Database techniques for the World-Wide Web: a survey. In *ACM SIGMOD Record*, volume 27, issue 3, September 1998, pages 59–74, ISSN 0163-5808

[Fraternali98HDMlite] P. Fraternali, P. Paolini: A Conceptual Model and a Tool Environment for Developing More Scalable, Dynamic, and Customizable Web Applications. In *Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology (EDBT98)*, Valencia, Spain, Springer LNCS 1377, March 1998

[Gaedke00WE] M. Gaedke, G. Graef: Development and Evolution of Web-Applications using the WebComposition Process Model. International Workshop on Web Engineering at the 9th International World-Wide Web Conference (WWW9), Amsterdam, The Netherlands, May 15, 2000.

[Gaedke05WSLS] M. Gaedke, M. Nussbaumer, J. Meinecke: WSLS: An Agile System Facilitating the Production of Service?Oriented Web Applications, Engineering Advanced Web Applications. S. C. M. Matera, Rinton Press, pages 26?-37, 2005, ISBN 1?58949?046?0

[Garret05AJAX] J. J. Garrett: Ajax: A New Approach to Web Applications, `http://www.adaptivepath.com/ideas/essays/archives/000385.php`, accessed 12 February 2008

[Garzotto95HDM] F. Garzotto, L. Mainetti, P. Paolini: Hypermedia design, analysis, and evaluation issues. In *Communications of the ACM*, volume 38, issue 8, August 1995, pages 74–86

[Gellersen97WebComposition] H.-W. Gellersen, R. Wicke, M. Gaedke: WebComposition: An Object-Oriented Support System for the Web Engineering Lifecycle. In *Computer Networks and ISDN Systems*, volume 29(8-13, Special Issue on the 6th International World-Wide Web Conference), 1997, pages 1429–1437

[Goecks00Unobtrusively] J. Goecks, J. Shavlik: Learning Users' Interests by Unobtrusively Observing Their Normal Behavior. In *Proceedings of the 5th International Conference on Intelligent User Interfaces*, New Orleans, Louisiana, USA, 2000, pages 129–132

[Gomez01Conceptual] J. Gómez, C. Cachero, O. Pastor: Conceptual modeling of device-independent Web applications. In *IEEE Multimedia*, volume 8, issue 2, April 2001, ISSN 1070-986X

[Graham03Wu] I. Graham: A Pattern Language for Web Usability. Addison-Wesley, 2003, ISBN 978-0201788884

[Heer05Prefuse] J. Heer, S. K. Card, J. A. Landay: prefuse: a toolkit for interactive information visualization. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2005 (CHI2005)* Portland, Oregon, USA, April 2005, pages 421–430

[Heinsen03UsabPrakt] S. Heinsen, P. Vogt: Usability praktisch umsetzen. Handbuch für Software, Web, Mobile Devices und andere interaktive Produkte. Hanser Verlag, Munich, Germany, 2003, ISBN 3-446-22272-3

[Hennicker01Modeling] R. Hennicker, N. Koch: Modeling the User Interface of Web Applications with UML. In *Practical UML-Based Rigorous Development Methods – Countering or Integrating the eXtremists*, Workshop of the pUML-Group at the UML 2001 conference, Gesellschaft für Informatik, Köllen Druck+Verlag, October 2001, pages 158–173

[HHS06WebGuidelines] U.S. Department of Health and Human Services: Research-Based Web Design & Usability Guidelines, 2006 ed., electronic version (usability.gov), Washington DC, USA, ISBN 0160762707

[Hong01WebQuilt] J. I. Hong, J. Heer, S. Waterson, J. A. Landay: WebQuilt: A Proxy-based Approach to Remote Web Usability Testing. In *ACM Transactions on Information Systems (TOIS)*, volume 19, issue 3, July 2001, pages 263–285, ISSN 1046-8188

[ICWE07Proc] L. Baresi, P. Fraternali, G.-J. Houben (eds.): Proceedings of the 7th International Conference on Web Engineering ICWE 2007, Como, Italy, Springer LNCS 4607, July 2007

[Isakowitz95RMM] T. Isakowitz, E. A. Stohr, P. Balasubramanian: RMM: A Methodology for Structured Hypermedia Design. In *Communications of the ACM*, volume 38, issue 8, August 1995, pages 34–44

[ISO9241] International Standards Organization: ISO 9241: Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability (1998) / Ergonomics of human-system interaction – Part 110: Dialogue principles (2006)

[Ivory01Automate] M. Y. Ivory, M. A. Hearst: The state of the art in automating usability evaluation of user interfaces. In *ACM Computing Surveys*, volume 33, issue 4, December 2001, pages 470–516

[Ivory01Metrics] M. Y. Ivory, R. R. Sinha, M. A. Hearst: Empirically Validated Web Page Design Metrics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2001 (CHI2001)*, Seattle, Washington, USA, April 2001, pages 53–60

[Ivory01PhD] M. Y. Ivory: An Empirical Foundation for Automated Web Interface Evaluation. PhD thesis, University of California at Berkeley, 2001

[Ivory03WebEval] M. Y. Ivory: Automated Web Site Evaluation: Researchers' and Practitioners' Perspectives. Kluwer Academic Publishers, Dordrecht, Netherlands, 2003, ISBN 1402016727

[Jacobson99UnifiedProcess] I. Jacobson, G. Booch, J. Rumbaugh: The unified software development process, Addison-Wesley, Boston, MA, USA, 1999, ISBN 0-201-57169-2

[Kappel04WE] G. Kappel, B. Pröll, S. Reich, W. Retschitzegger (eds.): Web Engineering. Systematische Entwicklung von Web-Anwendungen. dpunkt-Verlag, Heidelberg, Germany, 2004, ISBN 3-89864-234-8

[Kasday00WAVE] L. R. Kasday: A tool to evaluate universal Web accessibility. In *Proceedings of the 2000 conference on Universal Usability*, Arlington, Virginia, USA, November 2000, pages 161–162

[Kieras96GOMS] D. E. Kieras: A Guide to GOMS Model Usability Evaluation Using NGOMSL. In M. Helander, T. Landauer (eds.): The handbook of human-computer interaction. Amsterdam, Netherlands, 1996

[Kitajima00CoLiDeS] M. Kitajima, M. H. Blackmon, P. G. Polson: A comprehension-based model of Web navigation and its application to Web usability analysis. In *People and Computers XIV – Usability or Else! Proceedings of HCI 2000*, University of Sunderland, St. Peter's Campus, UK, September 2000, pages 357–373

[Knapp04ArgoUWE] A. Knapp, N. Koch, G. Zhang, H.-M. Hassler: Modeling Business Processes in Web Applications with ArgoUWE. In *Proceedings of the 7th International Conference on the Unified Modeling Language (UML 2004).* Springer LNCS 3273, October 2004

[Koch00Adaptive] N. Koch: Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process. PhD thesis, FAST Reihe Softwaretechnik, UNI-DRUCK Verlag, December 2000

[Koch02Expressive] N. Koch, A. Kraus: The Expressive Power of UML-based Web Engineering. Second International Workshop on Web-oriented Software Technology (IWWOST 2002), May 2002

[Koch06Transformations] N. Koch: Transformations Techniques in the Model-Driven Development Process of UWE. In *Proceedings of the 2nd Model-Driven Web Engineering Workshop (MDWE 2006)*, held at the 6th International Conference on Web Engineering (ICWE 2006), Palo Alto, CA, USA, July 2006

[Kraus02Generation] A. Kraus, N. Koch: Generation of Web Applications from UML Design Models using an XML Publishing Framework. In *Proceedings of the Integrated Design and Process Technology Conference (IDPT 2002)*, Pasadena, California, USA, June 2002

[Krug06Dont] S. Krug: Don't Make Me Think – A Common Sense Approach to Web Usability, 2nd edition, New Riders, Berkeley, CA, USA, 2006, ISBN 0-321-34475-8

[Lewis93TaskCentered] C. Lewis, J. Rieman: Task-Centered User Interface Design – a Practical Introduction, 1994, online publication, `ftp://ftp.cs.colorado.edu/pub/cs/cs/distribs/clewis/HCI-Design-Book/`

[Limbourg04UsiXML] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, V. Lopez-Jaquero: UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proceedings of the 9th IFIP Working Conference on Engineering for Human-Computer Interaction*, Hamburg, Germany, July 2004

[Lin00DENIM] J. Lin, M. Newman, J. Hong, J. Landay: DENIM: finding a tighter fit between tools and practice for Web site design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2000 (CHI2000)*, The Hague, The Netherlands, April 2000, pages 510–517

[Liu02RuleBased] W. Liu, S. Easterbrook, J. Mylopoulos: Rule-Based Detection of Inconsistency in UML Models. Workshop on Consistency Problems in UML-Based Software Development, at the Fifth International Conference on the Unified Modeling Language, Dresden, Germany, 2002, pages 106–123

[Lynch02WebStyleGuide] P. J. Lynch, S. Horton: Web Style Guide: Basic Design Principles for Creating Web Sites, 2nd edition, Yale University Press, March 2002

[Marincu04Comparative] C. Marincu, B. McMullin: A Comparative Assessment of Web Accessibility and Technical Standards Conformance in Four EU States. In *First Monday*, volume 9, number 7, April 2004, ISSN 1396-0466

[McLaughlin69SMOG] G. H. McLaughlin: SMOG Grading – a New Readability Formula. In *Journal of Reading*, volume 12, May 1969, pages 639–646

[Metz06CMS] C. Metz: Unterstützung von Content-Management-Prozessen und Web-Engineering-Methoden durch Web 2.0-Technologien. Diploma thesis, Media Informatics Group, University of Munich, November 2006

[Mueller01Cheese] F. Mueller, A. Lockerd: Cheese: Tracking Mouse Movement Activity on Websites, a Tool for User Modeling. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2001 (CHI2001)* (extended abstract), Seattle, Washington, USA, April 2001, pages 279–280

[NielsenAlertbox] J. Nielsen: Alertbox: Current issues in Web Usability, ISSN 1548-5552, `http://www.useit.com/alertbox/`, accessed 13 February 2007

[Nussbaumer08WSLS] M. Nußbaumer: Entwicklung und Evolution dienstorientierter Anwendungen im Web Engineering. Dissertation, Universitätsverlag Karlsruhe, 2008, ISBN 978-3-86644-208-5

[Obendorf04Scone] H. Obendorf, H. Weinreich, T. Haß: Automatic Support for Web User Studies with SCONE and TEA. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2004 (CHI2004)*, Vienna, Austria, April 2004, pages 1135–1138

[OMG03MDA] Object Management Group (OMG): MDA Guide Version 1.0.1. omg/2003-06-01, June 2003, `http://www.omg.org/docs/omg/03-06-01.pdf`

[OMG05XMI] Object Management Group (OMG): MOF 2.0 / XML Metadata Interchange (XMI) V2.1 specification, September 2005, `http://www.omg.org/technology/documents/formal/xmi.htm`

[OMG06OCL] Object Management Group (OMG): Object Constraint Language Specification, version 2.0 formal/2006-05-01, May 2006, `http://www.omg.org/docs/formal/06-05-01.pdf`

[OperaSmallDevices] Opera Software: Making Small Devices Look Great, `http://dev.opera.com/articles/view/making-small-devices-look-great/` accessed 28 June 2007

[Pastor97OOMethod] O. Pastor, E. Insfrán, V. Pelechano, J. Romero, J. Merseguer: OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods. In *Proceedings of the 9th International Conference on Advanced Information Systems Engineering, CAiSE 1997*, Barcelona, Spain, June 1997, Springer LNCS 1250, pages 145–158

[Reeder01WebEye] R. W. Reeder, P. Pirolli, S. K. Card: WebEyeMapper and WebLogger: Tools for Analyzing Eye Tracking Data Collected in Web-use Studies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2001 (CHI2001)* (demonstration), Seattle, Washington, USA, April 2001, pages 19–20

[Robbins99Cognitive] J. E. Robbins: Cognitive Support Features for Software Development Tools. PhD thesis, University of California at Irvine, 1999

[Robertson06RE] S. Robertson, J. Robertson: Mastering the Requirements Process, 2nd edition. Addison-Wesley Professional, Boston, USA, 2006, ISBN 9780321419491

[Rosson02UsabEng] M. B. Rosson, J. M. Carroll: Usability Engineering – Scenario-Based Development of Human-Computer Interaction. Morgan Kaufmann Publishers, San Francisco, USA, 2002, ISBN 1-55860-712-9

[Sastry99Webtour] C. R. Sastry, D. P. Lewis, A. Pizano: Webtour: a system to record and playback dynamic multimedia annotations on web document content. In *Proceedings of the seventh ACM international conference on Multimedia (Part 2)*, Orlando, Florida, USA, 1999, pages 175–178

[Schwabe98OOHDM] D. Schwabe, G. Rossi: An object oriented approach to Web-based applications design. In *Theory and Practice of Object Systems*, volume 4, issue 4, October 1998, pages 207–225, ISSN 1074-3227

[Singer06SemiAuto] A. Singer: Semi-automatische Accessibility-Analyse von Webseiten. Diploma thesis, Media Informatics Group, University of Munich, November 2006

[Spool98Website] J. M. Spool, T. Scanlon, C. Snyder, W. Schroeder, T. DeAngelo: Web site usability: a designer's guide. Morgan Kaufmann, San Francisco, CA, USA, 1998, ISBN 1-55860-569-X

[StancaAct04] Italian Government: Law n. 4, January 9, 2004 – Provisions to support the access to information technologies for the disabled ("The Stanca Act"). Official Gazette, January 2004. Unofficial English translation on `http://www.pubbliaccesso.gov.it/`, accessed 20 June 2007

[Tarasewich04DiscountEye] P. Tarasewich, S. Fillion: Discount Eye Tracking: The Enhanced Restricted Focus Viewer. In *Proceedings of the Americas Conference on Information Systems AMCIS 2004*, New York, NY, USA, August 2004

[Ullrich02PoorMans] C. Ullrich, E. Melis: The Poor Man's Eyetracker Tool of ActiveMath. In *Proceedings of the World Conference on E-Learning in Corporate Government Healthcare and Higher Education eLearn-2002*, Montreal, Canada, 2002, pages 2313–2316

[USSection508] U.S. Government, Architectural and Transportation Barriers Compliance Board: 36 CFR Part 1194, Electronic and Information Technology Accessibility Standards, Federal Register volume 65, number 246, 2000

[W3C-ARIA] World Wide Web Commitee (W3C), R. Schwerdtfeger (ed.): Roadmap for Accessible Rich Internet Applications (WAI-ARIA Roadmap), `http://www.w3.org/TR/wai-aria-roadmap/`, W3C Working Draft 4 February 2008

[W3C-ATAG2] World Wide Web Commitee (W3C), J. Treviranus, J. Richards, M. May (eds.): Authoring Tool Accessibility Guidelines (ATAG) 2.0, `http://www.w3.org/TR/ATAG20/`, W3C Working Draft, December 2006

[W3C-CSS2] World Wide Web Commitee (W3C), B. Bos, T. Çelik, I. Hickson, H. Wium Lie (eds.): Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, `http://www.w3.org/TR/2007/CR-CSS21-20070719`, W3C Candidate Recommendation, July 2007

[W3C-EARL] World Wide Web Commitee (W3C), Shadi Abou-Zahra (ed.): Evaluation and Report Language (EARL) 1.0 Schema, `http://www.w3.org/TR/EARL10/`, W3C Working Draft, March 2007

[W3C-GRDDL] World Wide Web Commitee (W3C), Dan Connolly (ed.): Gleaning Resource Descriptions from Dialects of Languages (GRDDL), `http://www.w3.org/TR/2007/REC-grddl-20070911/`, W3C Recommendation, September 2007

[W3C-RDF] World Wide Web Commitee (W3C): Resource Description Framework (RDF): Resource Description Framework (RDF) – Specifications, W3C Recommendations, `http://www.w3.org/RDF/`, accessed 21 October 2007

[W3C-RDFa] World Wide Web Commitee (W3C), B. Adida, M. Birbeck (eds.): RDFa Primer. Embedding Structured Data in Web Pages, `http://www.w3.org/TR/2007/WD-xhtml-rdfa-primer-20071026/`, W3C Working Draft, October 2007

[W3C-WAI] World Wide Web Commitee (W3C): Web Accessibility Initiative (WAI), `http://www.w3.org/WAI/`, accessed 23 May 2007

[W3C-WCAG1] World Wide Web Commitee (W3C), W. Chisholm, G. Vanderheiden, I. Jacobs (eds.): Web Content Accessibility Guidelines (WCAG) 1.0, `http://www.w3.org/TR/WCAG10/`, W3C recommendation, May 1999

[W3C-WCAG2] World Wide Web Commitee (W3C): Web Content Accessibility Guidelines 2.0, `http://www.w3.org/WAI/intro/wcag20`, accessed 23 May 2007

[W3C-WCAG2req] World Wide Web Commitee (W3C): Requirements for WCAG 2.0, `http://www.w3.org/TR/wcag2-req/`, W3C Working Group Note, April 2006

[W3C-XHTMLrole] World Wide Web Commitee (W3C), M. Birbeck, S. McCarron, S. Pemberton, T. V. Raman, R. Schwerdtfeger (eds.): XHTML Role Attribute Module, A module to support role classification of elements, `http://www.w3.org/TR/2007/WD-xhtml-role-20071004/`, W3C Working Draft, October 2007

[W3C-XPath] World Wide Web Commitee (W3C), J. Clark, S. DeRose (eds.): XML Path Language (XPath), Version 1.0, http://www.w3.org/TR/1999/REC-xpath-19991116, W3C Recommendation, November 1999

[Warmer98OCL] J. Warmer, A. Kleppe: The Object Constraint Language: Precise Modeling with UML. Addison Wesley Longman, Inc., Boston, MA, USA, 1998, ISBN 0-201-37940-6

[WatchfireBobby] Watchfire: Bobby accessibility validator tool, http://www.watchfire.com/products/webxm/bobby.aspx, accessed 16 July 2007

[WatchfireWebXACT] Watchfire: WebXACT online accessibility validator, http://webxact.watchfire.com, accessed 16 July 2007

[Wnuk05UsabilityProxy] M. Wnuk: "Usability-Proxy" für Webseiten. Project thesis, Media Informatics Group, University of Munich, August 2005

[Wnuk06Remote] M. Wnuk: Remote Web-based Monitoring and Shared Browsing. Diploma thesis, Media Informatics Group, University of Munich, December 2006

List of Figures

2.1 W3C Web Accessibility Initiative – overview 9
2.2 WCAG 2.0 and accompanying documents 15
2.3 Research-Based Web Design & Usability Guidelines – example guideline 18
2.4 Classification of usability evaluation methods 23
2.5 Improving usability during content creation: Bluefish HTML editor, Joomla CMS 25
2.6 Automated user tests with SCONE/TEA 27
2.7 Dataflow diagram of WebQuilt log visualisation 29
2.8 WebQuilt visualisation of user actions during a user test 29
2.9 Visualisation of mouse and gaze data 30
2.10 Usability validator classification: Completeness and correctness 32
2.11 The W3C validator 33
2.12 Annotation of input pages by the TAW accessibility checker 34
2.13 Example output of the Colorblind Web Page Filter 37

3.1 Model transformations with UML-Based web engineering (UWE) 43
3.2 UWE navigational structure model 44
3.3 UWE presentation model 45
3.4 ArgoUWE, a modelling tool for UWE 46
3.5 WebML hypertext model 48
3.6 WebML tool support: WebRatio user interface editor 49
3.7 OO-H Navigation Access Diagram (NAD) 51
3.8 OO-H Abstract Presentation Diagram (APD) 52
3.9 VisualWADE, the modelling and code generation tool of OO-H 53

4.1 Sketch of user interface for “virtual paper prototyping” 62
4.2 Sketch: Automatic usability analysis of models in an IDE 65
4.3 Model-based usability validation: Do images need alt text? 68
4.4 Simple modelling support in a prototype web CMS: Presentation model 70
4.5 Simple modelling support in a prototype web CMS: Menu editor 71
4.6 Simple modelling support in a prototype web CMS: Conceptual model 72
4.7 Simple modelling support in a prototype web CMS: Overall UI integration 73
4.8 Proxy architecture for a web-based logging system for user tests 76
4.9 Example for a web-based user feedback system – Google Maps 81
4.10 Visualisation for the changes made to content by a user (Wikipedia) 82

5.1 Example for a WebML context model . . . 86
5.2 UsiXML context model . . . 88
5.3 Model overview: usability package for attaching context to a user interface model 92
5.4 Example: Attaching contexts for three user groups to UI models . . . 94
5.5 User model specifying knowledge, abilities and goals . . . 96
5.6 Platform model specifying hardware, browser and network properties . . . 100
5.7 Environment model describing any influence of external factors on application use 102
5.8 UiModel and PresUiModel abstract metaclasses . . . 105
5.9 UiModel and NavUiModel abstract metaclasses . . . 108
5.10 Example class based on NavUiModel, with methods for OCL constraints . . . 110
5.11 Example: Specifying usability requirements using OCL constraints . . . 111
5.12 Overview of metaclasses in the usability package . . . 112
5.13 A «merge» copies UWE models and model extensions into usabilityUWE . . . 114
5.14 Integration of non-UML presentation models . . . 116
5.15 Example: XML representation of the proposed context model . . . 121

6.1 Comparison of tools for accessibility and usability validation . . . 127
6.2 Analysis results of the WebXACT accessibility checker . . . 129
6.3 Sample report of usability problems as found by the ATRC checker . . . 131
6.4 Model-based analysis of the layout of web pages . . . 138
6.5 Wusab start page – entering the URL of a web page . . . 143
6.6 Ad-hoc web-based creation of a presentation model for layout with Wusab . . . 144
6.7 Layout analysis – examples for acceptable layouts . . . 145
6.8 Excerpt from the results produced by Wusab for a page from Munich university . 147

7.1 UsaProxy adds JavaScript code to HTML pages that pass through it . . . 151
7.2 Images are not modified. For HTML, the HTTP request and response are logged 161
7.3 Example for the <script> tag that the proxy inserts on HTML pages . . . 162
7.4 UsaProxy modes: Proxy, transparent proxy, server mode . . . 164
7.5 The client-side JavaScript code sends an HTTP request with log data to the proxy 167
7.6 Sample of the log output generated by UsaProxy . . . 168
7.7 Visualisation of the log data for a session with an AJAX application . . . 169
7.8 Survey results: Influence of logging technology on acceptance of remote user tests 171

Acknowledgements

I want to thank my family, friends and colleagues for putting up with me during the longer-than-planned period of reduced social activity implied by the writing of this thesis, and their encouragement at those times when completing the task seemed to be a distant and uncertain event.

I want to acknowledge my advisor, Prof. Heinrich Hußmann, for giving me the freedom to work on the aspects that seemed most promising to me while at the same time giving feedback and steering me in the right direction, and for the luxury of being allowed to work almost exclusively on the thesis in the last year. Furthermore, I would also like to thank my secondary advisor, Prof. Martin Gaedke, for his insightful comments and suggestions on the content of the thesis. Despite not being my supervisor, Prof. Albrecht Schmidt deserves special mention and a big thank you for constantly making me work on new aspects of my research, write papers and start new projects, and for always being available to discuss new ideas and concepts.

I would also like to thank all my other colleagues at the Media Informatics Group, among them Andreas Pleuß for many discussions and valuable feedback regarding UML modelling, Alex De Luca for web technology and accessibility topics, and Raphael Wimmer for taking my mind off boring research issues during his regular trips to the coffee machine.

A number of students have contributed to this work via their project and diploma theses, sometimes even co-authoring conference papers and investing much more work than would have been required. In particular, thanks to Monika Wnuk for exceptional work on UsaProxy, but also to Andreas Singer, Ronald Ecker, Bernhard Engstler and Christoph Metz!

Like any other scientific work, this thesis would not have been possible without previous work by other researchers. While looking at related work, the discouraging impression was often that all my nice ideas had already been covered in some mid-1990s CHI paper. However, in reality building upon the work of others has allowed me to go further than I would ever have been capable of going on my own. In particular, I thank Melody Ivory and Nora Koch for their work.

For most of the time while working towards my PhD, I was funded by the BMBF (Bundesministerium für Bildung und Forschung) in the context of the “intermedia” project. I want to thank my computer science, business administration and communication science colleagues in this project for interesting cross-discipline work and for allowing me to apply my research to the overall aims of the project.