Journal of Web Engineering, Vol. 5, No. 1 (2006) 043–064

EXTENDING WEB ENGINEERING MODELS AND TOOLS FOR AUTOMATIC USABILITY VALIDATION

RICHARD ATTERER
Media Informatics Group, University of Munich
Amalienstr. 17, 80333 Munich, Germany
richard.atterer@ifi.lmu.de

ALBRECHT SCHMIDT
Embedded Interaction Research Group, University of Munich
Amalienstr. 17, 80333 Munich, Germany
albrecht.schmidt@acm.org

HEINRICH HUSSMANN
Media Informatics Group, University of Munich
Amalienstr. 17, 80333 Munich, Germany
Heinrich.Hussmann@ifi.lmu.de

Received April 30, 2005
Revised February 2, 2006

In this paper, we present ideas of how to improve the quality of automated web usability validators. This can be achieved by taking advantage of the models of established Web Engineering solutions.

We begin by analysing two of the currently available Web Engineering solutions (UWE and OO-H) with regard to the question whether any websites created with them have a high usability. Additionally, it is examined whether the respective models can express usability aspects. In a small case study, an example website is created by converting a model to an implementation manually. Special attention is paid to usability issues regarding both the generated pages and the development process.

Subsequently, we take a look at existing implementations of usability validators, noting how the quality of their results is often not optimal. This is due to the fact that not enough abstract information is available. In the next step, we identify existing Web Engineering model properties which can be used to improve the checks, and propose further extensions to models.

Keywords: Web Engineering, usability, modeling, validation, comparison study

Communicated by: S Comai

1 Introduction

Creating an easy-to-use user interface is just one of the many challenges that the developers of a web application face during their work. Due to the many factors which influence the user experience of the application (ranging from differences in the user's input and output devices to latency issues and the request/response paradigm of the HTTP protocol), an application with the same functionality typically has a quite different interface when implemented as a web application instead of a GUI application for a desktop computer.

Current Web Engineering approaches such as UWE (UML-based Web Engineering, [15]) and OO-H (Object-oriented Hypermedia, [9]) try to offer a complete solution to generate web pages (or assist in their generation), typically providing models for the application logic, navigation structure and the final presentation of the web pages. However, they do not currently offer integrated tool support for the automated or semi-automated usability validation of the websites they create.

In this paper, we examine the potential of model-based usability support. In preparation for this task, we examine the state of the art both in the area of Web Engineering and in the area of automated usability validation.

For Web Engineering, the focus is on the navigation and presentation aspects, in particular the question whether the output of the existent tools results in websites with high usability. Building upon previous work in this area [3], the following is examined for the two Web Engineering solutions OO-H and UWE:

- Does the *method* involve aspects which have the goal of improving usability?
- Do the navigation and presentation *models* allow expressing usability constraints?
- Do the *tools* have support for what the methods/models provide in terms of usability?

In the area of automated usability validation, we analyse existing solutions, paying special attention to their limits and shortcomings:

- Which tests of the existing tools only output general messages? (Like: "please check whether the page [meets a certain criterion]"). Could these tests be improved?
- Looking at commonly used sets of usability guidelines (e.g. the Web Style Guide [23]), which guidelines can be implemented for automatic tests using information from the models?
- Which guidelines could be implemented, but require extensions to the models?

This paper is structured as follows: Section 2 introduces the example setting of business processes in a travel agency and describes the experience of creating a website prototype supporting these business processes, in particular the issues relevant to usability. Subsequently, section 3 is concerned with the existing solutions UWE and OO-H and the usability support they provide. Section 4 discusses perceived shortcomings with the current methods and tools.

Section 5 analyses the features and shortcomings of existing automated usability validators, it includes an overview of available tools and a discussion of how the lack of access to a model makes them less powerful. Next, section 6 gives a list of tests which become possible if a model is present, and section 7 suggests a number of items which should be included in models to allow further improvement of automated tests. Section 8 presents some conclusions and an overview of future work.

2 Case Study

The entire process of modelling and implementing a website was performed with a small example. All steps were deliberately performed manually in order to get a feeling for all the issues related to creating a working, usable website from nothing more than an idea of the business processes it has to support. A prototype was implemented in PHP.

2.1 Example Setting: Travel Agency

For the subsequent work, the example that was chosen is a travel agency. This choice is particularly interesting for usability research because of the following properties:

- Several quite *different activities* need to be combined in an intuitive way in the same user interface: Searching for information (on travels, flights, car rental etc.), collecting "interesting" items and booking/reserving/cancelling items.
- The same interface (with only minor variations) should be usable by *different audiences*, i.e. both by customers booking flights from home and by travel agents serving customers in the agency.
- The setting mostly fits into the standard, widely-used application type of an "online shop", so any conclusions drawn will be applicable to a large number of existing web applications.

2.2 Modelling of the Business Processes

Going from the mere idea of the business to be modelled to the finished website involved the following steps: Modelling the application, creating a page design and graphical design, and converting the model elements into elements on the web pages.

In [21], the authors describe a systematic approach for modelling business processes as well as analysing and optimising them. This approach was followed to create UML activity diagrams for the example by identifying the business goals and active business partners, determining those business partners' business use cases, describing the different use cases with a few sentences, and finally by modelling the business processes in detail. The following paragraphs briefly describe the results of the different steps from [21] for the example "travel agency" scenario. The method introduced by Oestereich et al. includes further steps which aim at optimising the business processes, which are not of interest for this paper.

Determining the focus of modelling As the first step, the company to be modelled and its aims were determined. At this level of abstraction, the aim is simply "to earn money". Also, an overview of the different organisational units and the relationships between them (e.g. which units take orders from others) was created. In the case of the travel agency, there might be marketing and sales departments. However, the subsequent steps concentrated on the sales of travels due to the fact that it is more interesting in the context of this paper, as it is the department where direct interaction between a customer and a travel agent takes place.

Modelling of organisational units The organisational units which were obtained in the previous step were modelled using a UML class diagram. Because of the simple example, this step is trivial, and the resulting diagram is very simple.

Identifying persons taking part in the business processes From the point of view of the company, there are two types of customers: People wanting to book a travel for themselves,

and organizers of travels which rely on the agency to book flights, hotels etc. for a larger group of people. To keep the example simple, only the former are considered.

Identifying and describing business processes The following business processes were identified, and described informally with a few sentences:

- Search for items (hotels, flights, complete travels, or "extras" like a rented car)
- Reserve an item, e.g. put a flight on hold
- Book an item
- Enter or update customer data

Modelling business processes At this point, Oestereich et al. suggest creating a UML model for each business process. However, following this suggestion was not optimal for the purposes of Web Engineering: After creating the individual models, the subsequent attempt to convert them to a navigation structure for the website turned up a question – how should the different processes be "connected" in the user interface, and where should navigation from one process to another be possible?

For example, a typical scenario is that a customer has inquired for information about a flight, and now wants to book it. In this case, navigation from "search for item" to "book an item" should be possible, and it should be possible without the need to re-enter the flight number.

In response to this problem, a unified UML activity diagram for all related business processes was created. A simplified version of it is shown in figure 1. Because the example is not very complex, the model is still easy to understand. For real-world business processes, it will often be necessary to create one more abstract UML activity diagram which only shows the relations between individual business processes (e.g. which processes precede or are contained in other processes), and to model these individual business processes in separate diagrams. The more abstract model provides important information when creating a website for the business processes, as it documents typical usage scenarios.

2.3 Creating the Website

At this point, a number of notes regarding the usability of the process can already be made. Ideally, to allow automatic processing by tools, we would like to be able to integrate the following points into the diagram in a format other than UML comments:

- Searching, reserving and booking are too intertwined to be separated from each other: During a typical customer visit in a travel agency, the customer will alternately express his wishes, ask for temporary reservations (put a seat on a plane on hold) and ask for more details or alternatives to the reserved items. The web pages should support such quick switches between activities.
- Searching is the most important activity: It is desirable to have the search facility available on all pages for quick access, and to have a way to remember earlier queries.

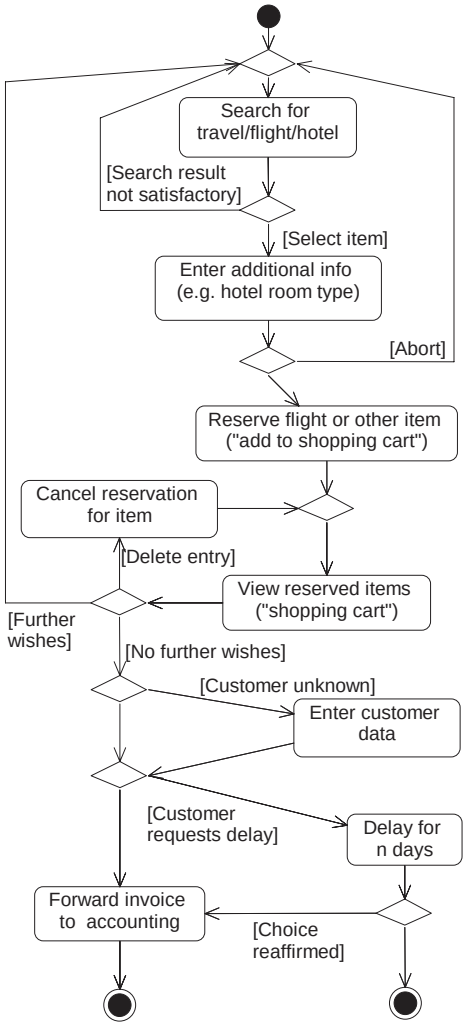

Fig. 1: **Activity diagram for a business process: Searching for and booking travels in a travel agency**

- The whole business process may take several days to complete. As a result, the user interface needs to be able to track many concurrent instances of the process, similar to a trouble ticket system.
- Unnecessary manual entry of customer data should be avoided, it should be possible to search for existing customers by customer number or name.

As we will see later, some of these issues can be expressed by Web Engineering solutions, though not necessarily at this level of abstraction.

Page design Once the business process was defined, an overall page design was created for the web application, and the required page elements were fit into it. Whereas the earlier

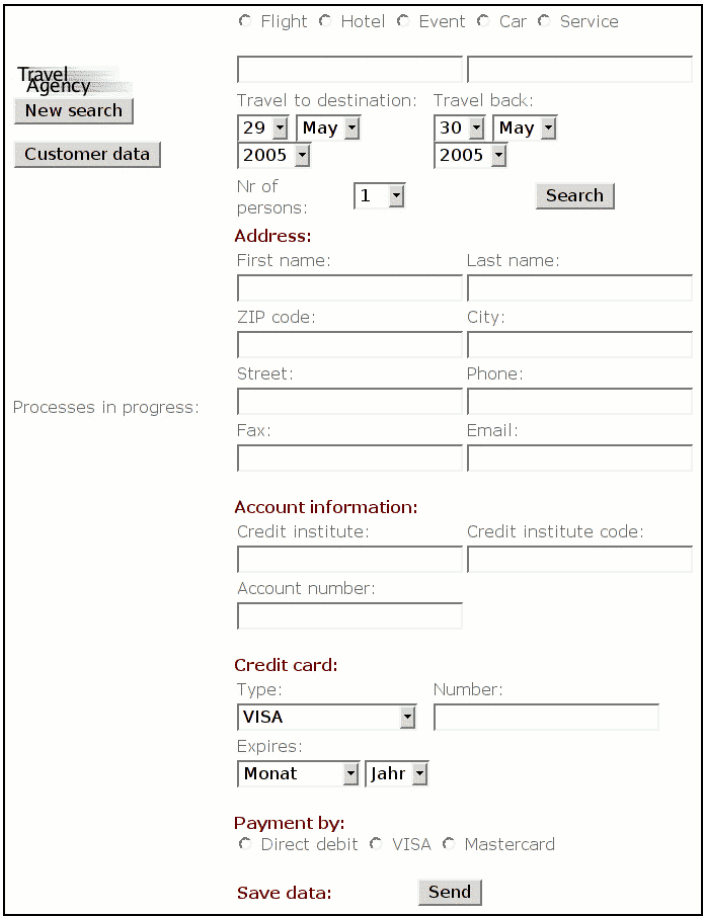

Fig. 2: **Screenshot of the prototype website. The simple business process "new customer", which is part of "enter customer data", can be represented by a single HTML page.**

modelling work followed a top-down approach (from an abstract business idea to relatively concrete activity diagrams), this can be viewed as a bottom-up approach.

One reason for this approach is that the low-level page design is actually very standardized for most web applications: Among others, Nielsen [19, Nr 7] [20] has conducted research which shows that a standard page design with navigation at the left, advertisement at the top (if necessary) and other similar items is desirable, because it reduces the time needed by (notoriously impatient) users to learn how the site works.

In the example, a simple navigation area on the left contains links to support the basic functionality, which is to start actions corresponding to the different business processes (customer database, information and booking etc.). As noted above, the system needs to keep track of a number of separate instances of business processes which run in parallel; they are also listed in the navigation area. See figure 3 for a screenshot of the prototype.

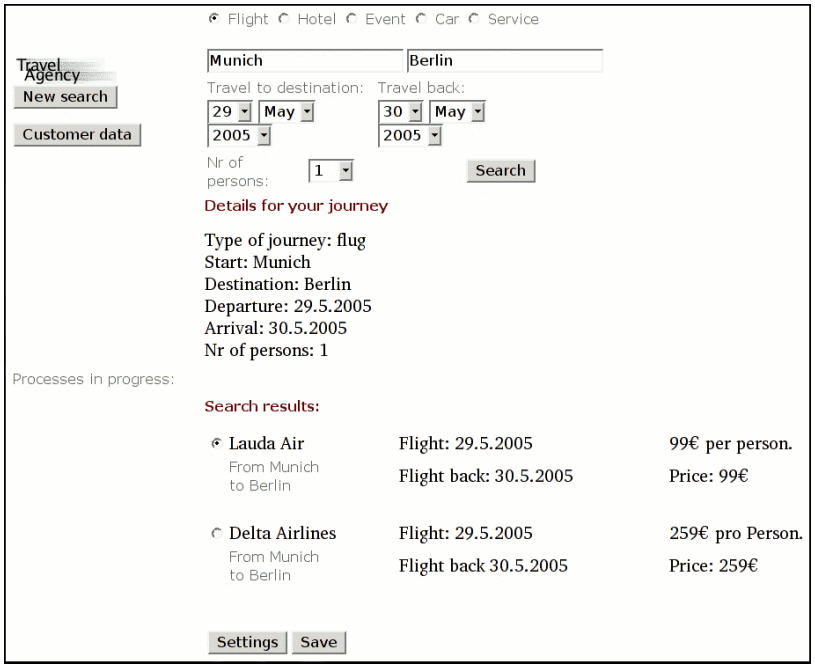

Fig. 3: **Screenshot of the prototype website displaying search results. From the navigation in the top left corner, new business processes can be started. The list of active processes (each holding one or more reserved items) is displayed below it, unless it is empty (as in this case).**

Several typical web application patterns [9, section 3.2] were discovered with the example application: If larger amounts of information, such as customer data, are to be entered by the user, a *guided tour* (pages with "previous"/"next" buttons) is appropriate. A *shopping cart* is also present – the cart is called "reserved items" on the web pages to stress the fact that e.g. a seat on a plane is temporarily "locked" while it is in the set of reserved items. Standard *indexes* are used to create lists of links to search results or active business processes.

In addition to these, a new concept was identified which we will call *inlining*, an allusion to inlined C/C++ functions: Since searching is a central activity for the example, having the user follow links to the search form is not ideal. Instead, a small version of the search form is displayed in all the places where a link to the form would otherwise have been placed. With the prototype website, there is no distinction between the normal and small versions of the search form – the form from figure 3 simply appears at the top of all pages where the search facility should be available. Figure 2 shows the web page for the sub-process "new customer".

Graphical design The graphical design was not a primary concern when implementing the prototype for the example website – the resulting pages are simple and only use a minimum amount of graphic elements.

However, in general the graphical design is an important aspect which can significantly influence the overall usability of the final website. The web application developer should take

care to add graphics which are not only visually pleasing, but which also do not surprise or mislead the user (e.g. graphics which are not recognisable as links). Instead, the graphical design should stress the intended use of the different elements on the page.

Conversion of the model The model in figure 1 is slightly too abstract to be converted into web pages. For Web Engineering solutions with tool support, the information for automatically generating the link structure and page content is available in the navigation and presentation models, but no such models were created for the prototype. Instead, the activity model served as a basis for the ad-hoc manual creation of "appropriate" links and page content.

In practice, this worked very well – in particular, since only one model is present, you do not need to switch between the "application logic", "navigation" and "presentation" views of the web application, and the different diagrams for these views do not need to be kept in sync. Also, it is usually easy for a human to come up with appropriate page content for a given model; as long as no automatic creation of pages takes place, additional models may just increase the amount of modelling (i.e. the amount of work) without significant benefits.

Consequently, the use of fewer different types of diagrams can be interpreted as improving the usability *of the development process*. It may be advisable to restrict the use of additional diagrams to particularly challenging areas of the application. This would be analogous to current practice in classical Software Engineering, where class diagrams are the predominant form of diagram, and e.g. object diagrams are used to model a small number of selected details about the system.

Further work will be necessary to determine whether the positive aspects of advanced tool support (with automatic page generation) and of having fewer types of diagrams can be combined.

2.4 Lessons Learned

Usability is multi-facetted, it is influenced by aspects at every abstraction level, from the business process model to graphics design. In order to be suitable for creating usable websites, a Web Engineering solution needs to take all these aspects into account, which is far from trivial.

Usability is difficult to get right and easy to get wrong. Since most people designing web applications are not experts in this field, it would be nice to have more than just usability guidelines like [6] and [19] – such guidelines need to be integrated into the design process, and ideally even the Web Engineering tools.

3 A Look at Existing Solutions

In this section, we will have a look at the navigation and presentation aspects of OO-H (`http://gplsi.dlsi.ua.es/iwad/ooh_project/`) and UWE (`http://www.pst.ifi.lmu.de/projekte/uwe/`), and the respective Web Engineering tools, VisualWADE and ArgoUWE, to check how well the creation of usable websites is supported. As mentioned in the introduction, the method, the models and the tools will be examined.

3.1 OO-H

The Object-oriented Hypermedia Method is described in [9], more details on presentational aspects are available in [5]. Creating a web application using OO-H involves creating standard class diagrams and, based on these, *navigational access diagrams* for each type of user. Using these navigational diagrams, a default web interface, intended for quick prototyping, can be generated.

For more sophisticated web pages, a default *abstract presentation diagram* is derived from the navigational diagram, and subsequently refined by the web application developer. Essentially, the presentation diagram represents a template mechanism.

The VisualWADE tool offers very good support for all steps of the development process, including automatic generation of prototypes.

This approach is very general and gives freedom to the developer to add steps like usability testing with prototypes.

One item that is directly relevant to usability is OO-H's decision to promote the generation of different navigational diagrams for different types of users. It is not clear whether this is supposed to include not only descriptions of the same process from different perspectives ("customer books seat on plane" vs. "airline executive accepts booking"), but also by different audiences ("customer books" vs. "travel agent books"). The latter adaptation of content must be performed with great care – in particular, websites which present different content after asking the visitor what audience group he belongs to (e.g. for B2B, "home office" and "small company") can be frustrating to use.

The method's use of certain patterns is laudable. The available patterns deal with some problems related to usability (such as the user's tendency to "get lost" on sites with a large number of pages) and try to provide standardised ways to deal with the problems (e.g. the *Location Pattern*, which adds headers and footers with navigation information to pages).

The use of patterns is a step in the right direction, but in the authors' opinion, the available patterns are very concrete, and the whole presentational diagram could easily be replaced with a more traditional template solution such as SSI (server-side includes).

Because automatic creation of prototypes is possible, the models given as input to the tools must describe the web application in appropriate detail. This means that a significant amount of modelling is necessary to get relatively simple results – this makes the development process somewhat tedious and time-consuming.

3.2 UWE

The approach of UML-based Web Engineering, described in [15], is comparable to that of OO-H. After creating the conceptual model, which describes the application logic in the same way as in classical Software Engineering, *navigation* and *presentation models* are constructed. These models are subsequently converted into an XML format using the UWEXML preprocessor. Next, the UWEXML generator semi-automatically produces XML templates, presentation stylesheets etc. The developer can manually refine the output to suit their needs. In the presentation model, patterns like a *guided tour* (pages with "previous" and "next" buttons) or indexes (lists of links) can be utilised – similar to OO-H, the presentation diagram is essentially a template mechanism with support for patterns.

In [10], the method is described with a focus on the user interface and extended with the aim of improving usability. Storyboarding and the creation of pure HTML prototypes are introduced to help the developer with the design of the presentation model.

The tool support for UWE (ArgoUWE [14] for modelling, UWEXML for processing the models) allows the semi-automatic creation of websites. The tools promote the use of frames to subdivide pages into subpages – this is a feature which should not be overused due to the fact that frames are generally frowned upon by usability experts.

As evidenced by the thoughts on storyboarding, the authors of UWE have put some effort into creating a development method which ensures that the resulting websites have high usability. Nevertheless, the abstraction level of any "usability modelling" is comparable to that of OO-H: The available patterns are very concrete, and the presentation diagram is separate from the navigation diagram – a fact which may make it more difficult to create intuitive websites, as the two models are strongly related to each other, and changes to the one will rarely go without changes in the other.

4 Critique

The two Web Engineering solutions are similar in their approach, so any criticism regarding usability support is equally applicable to both.

The methods and tools show promise, but there is still room for improvement regarding the "usability of the development process". The use of the presentation models is not fully justified: They could be replaced with simpler template solutions without ill effects, primarily due to the fact that the existing user interface patterns supported by them are so straightforward that they can be expressed in simple template languages. Also, creating the presentation model results in work for the developer that does not pay off; HTML or XML page templates will typically *still* need to be created in addition to the model. Finally, the editors for the presentation model are not nearly as powerful as established web design software like Dreamweaver. They only support a subset of what modern web pages (using e.g. JavaScript) can do, and thus limit the developer in his possibilities.

With UWE and OO-H, there are separate models for navigation and presentation. As mentioned earlier, having just one model which includes both presentational and navigational aspects may be more desirable from a usability point of view, due to the fact that navigation and presentation are closely coupled: The one deals with the operation of the website at the page level, the other within a page.

Regarding the navigation models, another issue worth noting is that the methods focus on links necessary for the application, not so much on "general" links like a site navigation menu, "related links" etc. The importance of intuitive general navigation structures should not be underestimated, because for many sites, users browse a site primarily because they search for information, and not because they want to actually use the site's central web application.

Neither UWE nor OO-H provides a way to model usability guidelines at an abstract level, e.g. "searching is the most important activity", "a purchase in the online shop should be possible in five minutes" etc. Whether such modelling is possible at all remains an open issue.

5 Existing Approaches to Automated Usability Validation

In the previous sections, the methods, models and tools in the field of Web Engineering were analysed with regard to their suitability for usability validation. We now turn towards the field of usability research to have a look at the existing automated usability or accessibility validators. In section 5.1, we examine how powerful the current solutions are by looking at a selection of available validation tools. Subsequently, section 5.2 highlights the limits inherent in validating HTML pages without having a model which describes certain properties of the pages.

To get an overview of the properties that are desirable for an easy-to-use website, we looked at a variety of guidelines from different sources, including the following:

- World Wide Web Commitee (W3C): Web Accessibility Initiative (WAI) and related documents [25], [24] – for concrete page design aspects (colours, font sizes etc.)
- Yale Web Style Guide [23] – for both concrete design advice and more general guidelines regarding e.g. testing
- Jakob Nielsen's alertbox series [16] – for additional significant rules
- siteusability.com [22] – for guidelines resulting from the list of common usability mistakes
- About Face 2.0 [7] – for further user interface design guidelines

Only few of these guidelines are intended to be implemented by automated tools – instead, most assume a human who is able to judge properties of the web pages. An example from the Web Style Guide is the advice to divide information into chunks of "appropriate" size, dependent on the nature of the content, before turning each chunk into a web page.

As we will see in the following sections, there is a significant gap between the above guidelines for "human usability validators" and the typical sets of guidelines that automated validation tools support. The intent of our work is to make this gap narrower.

5.1 Existing Usability Validators

This section gives an overview of the currently available validators. Apart from the actual functionality (in terms of the set of guidelines that is validated), we also take a look at the ideas behind some of the tools, for example interactive repair of errors or extending the set of rules to be verified. An extensive discussion of the different possibilities of automated validation can be found in [12].

We have looked at the following usability and accessibility validators:

- A-Prompt (`http://aprompt.snow.utoronto.ca`)
- Bobby (`http://bobby.watchfire.com`)
- EvalIris [1]
- Kwaresmi [4] (`http://www.isys.ucl.ac.be/bchi/research/Kwaresmi.htm`)
- LIFT (`http://www.usablenet.com`)
- NAUTICUS (`http://giove.cnuce.cnr.it/nauticus/nauticus.html`)
- TAW (`http://www.tawdis.net`)
- WAVE (`http://wave.webaim.org`)
- WebTango [13]

None of these tools works with a presentational or navigational model taken from a Web Engineering solution like UWE [14] or OO-H [5]. Furthermore, none allows interactive "reverse-engineering" of models from existing web pages, or annotating them with abstract information. Looking at the output of the tools, it becomes clear that the lack of additional, more abstract information about the pages is a problem: Many tools output messages which tell the user to perform manual checks for some of the page content.

The following paragraphs attempt to categorise the available tools into several groups. Very roughly, they also represent the development in the area of web page validators over time: In the beginning, the validator was simply seen as a program to implement a given set of tests. Later on, attention was paid to issues like making the validator itself more flexible and easy to use.

5.1.1 Hard-coded tests

A number of validators use a set of built-in tests which is not extensible without altering the program source code. Validation happens in a non-interactive way, the tool output consists of error/warning messages and the line numbers in the HTML code they apply to.

This group of tools is currently most advanced with regard to the variety of guidelines that have been implemented as tests. A possible reason for this is that they have been under development for a longer time than other tools, and that these other tools concentrate on introducing new concepts rather than boasting an extensive number of tests.

An example for such a tool which has matured into a commercial product is LIFT (`http://www.usablenet.com`). Given the URL of a web page, it can examine the page as well as pages it links to. It outputs usability and accessibility tips which are generated by a large number of tests. Due to lack of abstract information about the pages being analysed, it cannot always determine whether there is a problem with a page, which results in output messages like "Please check if the table is used to present data and in such a case provide header information."

5.1.2 Annotation of input web page

In an attempt to make operation of the validators easier, some tools output a version of the analysed web page which has been annotated with little icons to highlight potential problems. One of the most popular accessibility tools in this category is the *Bobby* service (`http://bobby.watchfire.com`), whose tests verify whether pages comply with the Web Content Accessibility Guidelines (WCAG 1.0) or the U.S. government's section 508 guidelines. For problems which cannot be verified automatically, Bobby's output always includes a number of general guidelines which the user must verify manually, like: "Are there navigation bars for easy access to the navigation structure?"

TAW (test accesibilidad web, `http://www.tawdis.net`) is similar to Bobby in operation and presentation of results. It also outputs messages which tell the developer to check some aspects manually, e.g. "Si la imagen contiene información importante, use el atributo longdesc." (If the image contains important information, use the longdesc attribute).

WAVE (`http://wave.webaim.org`) annotates the supplied web page with a variety of icons. For some real-world pages, this can be confusing. The tool also suffers from the problem that no model is available for pages. For example, it will always warn if an image's alt text is empty, even if that image is ornamental, so that an empty string is appropriate.

5.1.3 Interactive Repair of Problems

An interesting approach taken by some tools is to help the user to repair any problems that the tool detects, e.g. by suggesting several alternative solutions for each problem. This way, the development of accessible and user-friendly web pages can be made a lot easier, especially for unexperienced users who cannot correctly interpret messages like "consider using longdesc for this image".

Examples for tools which support interactive repair include A-Prompt (`http://aprompt.snow.utoronto.ca`), a Windows GUI application concentrating on the section 508 guidelines, and NAUTICUS (`http://giove.cnuce.cnr.it/nauticus/nauticus.html`) which aims at making web pages accessible for the blind.

5.1.4 Statistics-Based Algorithms

One perceived problem with the tools above is that it is very difficult to determine exactly what tests are important to determine whether a web page is user-friendly. Thus, rather than working with a fixed set of tests to validate the input web pages, some tools take the approach of comparing properties of the website to be tested with the same properties of a number of reference websites. Those reference websites have been rated by experts regarding their accessibility and usability, so the tool is able to calculate a rating for the test website. For example, WebTango [13] bases its ratings on the winners of the Webby Awards.

5.1.5 Extensible Rulesets

Another issue with some tools is that they are difficult to extend with new tests. Being able to extend a tool easily and without modifying its source code can be beneficial: User testing can uncover problems with the analysed web pages which should from now on be checked automatically. Furthermore, the web designer performing the tests may not be capable of programming in a language like Java or C. Thus, validators have emerged which enable the user to specify additional rules in a notation tailored towards verification of HTML pages.

With Kwaresmi [4] (`http://www.isys.ucl.ac.be/bchi/research/Kwaresmi.htm`), guidelines are specified in a special language called GDL (guideline definition language). EvalIris [1] is a web-based accessibility evaluator which can be used either directly by a user or as a Web Service, and which can analyse single web pages or entire websites. The authors have defined an XML-Schema which allows for the description of guidelines in XML.

5.2 Limits When Validating Based on Implementation Only

There exists a large number of usability and accessibility guidelines which is validated by current tools just by analysing the HTML pages, CSS (cascading style sheets) and other content that can be retrieved from a website.

However, the implementation of checks for these guidelines often suffers from the problem that no model is available, i.e. no abstract description of certain properties of the web page (or its parts). This way, the validator either fails to find certain usability problems in the pages or it outputs too many general warning messages.

The rest of this section discusses the implementation of checks for a selection of guidelines, and the problems that arise if no model is available.

5.2.1 Design

The following aspects should be checked by tools to ensure that the design of the web pages does not have a negative effect on their usability:

- *Colours and fonts:* It is straightforward to check given HTML code for high colour contrast [11] and the use of a limited number of different font faces, but it is not possible to do this reliably for images which contain a rendered version of some text, unless a model provides information regarding the text contained in the image.

- *Animated design elements:* Blinking text or images should be identified by tools – for some users, they are irritating and annoying, for others (e.g. epileptics) they can even be dangerous. Detection of blinking text, blinking animated graphics and blinking embedded content like Flash animations presents an increasingly difficult, but not insurmountable challenge to a validation tool. On the other hand, it is impossible to let the tool allow more animation effects for advertising, disallow them for the navigation area, or similar, unless additional semantic information about the page is available.

- *Liquid layout:* The web page layout should adjust to the current browser window size [19, Nr 2]. While this can be verified without the help of a model, a non-model-based tool must rely on heuristics [8] to determine that the *main content* of the page adjusts its width when the window size changes, and not e.g. an empty table column.

5.2.2 Functionality

A variety of checks can be performed to determine whether a web page will function as intended with different users, devices or rendering software:

- *Dependency on scripting support:* To some extent, it is possible to determine automatically whether a web page will no longer work if scripting, e.g. JavaScript, is not supported by the user agent. The decision as to whether or where scripting is allowed should be left to the developer, who should be able to specify his wishes when he designs the page, either by putting them in a model or the Web Engineering tool's project preferences.

- *Dependency on graphics support:* Similar to the point above, it is possible to ascertain to some extent whether a page will still work without graphics support in the user agent.

- *Working links:* It is easy to flag broken links. However, only a tool solution which is integrated into a Web Engineering environment can help with more advanced issues related to valid links, e.g. to ensure that after an update of the site structure, old URLs (old bookmarks or external "deep links" to the site) continue to work.

- *Accessibility by search engines:* It is possible for a tool to provide hints as to how to make a website easy to index by search engines. On the other hand, without a navigation model it is difficult to find out whether large parts of a site are hidden from search engines, e.g. because they are only reachable through a form that has to be filled by a human user.

5.2.3 Content

The usability of web sites is not just influenced by the special features of the medium used, one also needs to take care that the sites' content follows certain principles.

- *Spelling and grammar:* The orthographical correctness of the text can be checked fairly easily. In some cases, the check's quality could be improved if the tool is given extra information, such as the type of text ("contains medical terms") or information about the expected audience.
- *Concise and scannable style:* Content on web sites is usually only scanned by visitors in search for information. To support this style of consumption, a tool could measure the number of words per sentence or per paragraph, the frequency of headings and the amount of highlighted words (emphasis or links).

6 Modelling of Usability Aspects of Web Applications

In section 5.1, we have described the ways in which current tools assist in the generation of easy-to-use websites, under the assumption that only the HTML implementation is available for automatic processing. As we have seen, in many cases this leads to heuristical approaches: By some means or other, the tool has to deduce certain abstract properties (e.g. "is this the main page content?") before applying rules to the pages. Despite advances e.g. in the area of automatically determining which part of a page is the main content [8], this is not ideal – the heuristics can provide incorrect results, and a tool which makes too many mistakes is a nuisance rather than a help.

The quality of automated usability tool support can be increased significantly by taking advantage of the models which are available in current Web Engineering solutions. For instance, UWE and OO-H both feature navigational models which provide details on the ways the site is intended to be traversed, and presentational models which define abstract properties of the page layout – for example, they allow us to assign meaning to parts of the page layout, like "this is an advertisement". Building upon the insights gained in section 5.2, this section gives examples of improvements which are possible if the tool has access to a model.

In this section, we will use the "travel agency" example from section 2 to illustrate some of the discussed extensions to models, and to describe validator checks which take advantage of these extensions. However, it should be noted that the model extensions should only be regarded as examples of how to integrate usability-related information into existing Web Engineering environments – in this work, we concentrate on *what* needs to be added to models to make model-based usability validation possible, and only to a lesser degree on *how* to add it. Additional research is needed to find the right way to represent this information.

A number of ways are imaginable for the representation of usability information during the development of applications with Web Engineering methods:

- The existing presentation and navigation models can be extended to allow the modelling of information which is useful for automatic validation.
- A new "usability model" can be introduced. It can contain information which is not suitable for inclusion into existing models, and can contain references to other models to annotate them from a usability point of view.

- A new language for the description of usability information could be of advantage. For example, a text-based description language could be embedded directly in the HTML output of the website implementation, or an XML-based language could use XPath expressions to refer to parts of HTML pages.
- On the other hand, if the level of abstraction of the information is not high enough, it may make sense only to store this information in the "project preferences" of a Web Engineering toolkit, and not in the models.

In practice, a mixture of these possible approaches is necessary: The task of making a website usable needs to fit into established development processes. Furthermore, the usability-related work should not come with more burdens than benefits for the developer. Finally, the chosen solution for the representation of usability information should allow for easy-to-use and powerful tool support. As mentioned, this will be the subject of further work.

6.1 Presentational Aspects

Presentation models allow us to assign meaning to parts of the page layout. Depending on the level of detail with which the model allows attaching further information to page areas, the quality of checks can be improved, for example in the following areas:

Standard page layout With a model which describes the different page areas, we can check whether the page design follows one out of a number of de-facto standards, for example "three columns with header, site name at top, navigation at left, advertisement at right". It is very important not to use an unusual overall layout, as this requires visitors to learn how to use the site, and hinders them in their primary goal of finding the information they are looking for.

Related to this, it is possible to check whether the layout of content is consistent across all

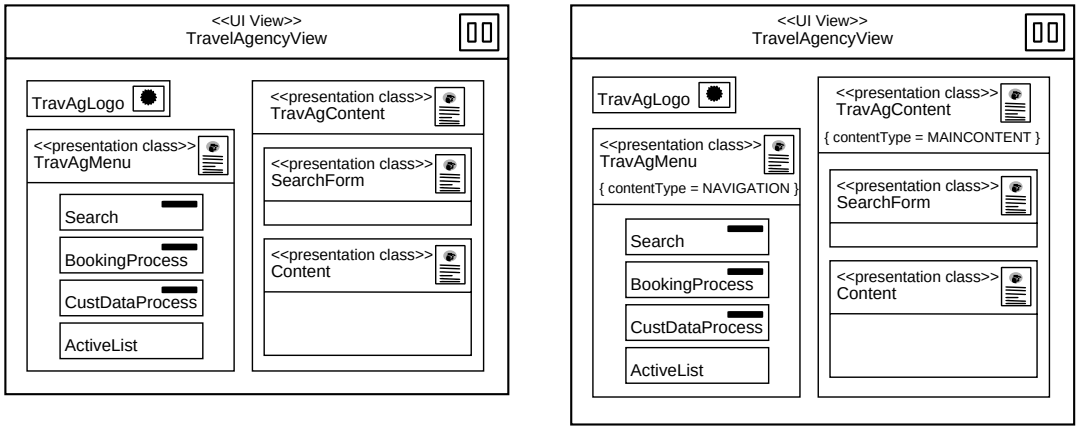

Fig. 4: **Left: UWE presentation model for the example website from section 2. Right: A possible extension allows a number of automated usability tests; the "contentType" property describes the type of information that is presented.**

Fig. 5: **Left: The site logo is declared as an image which is not ornamental in nature. Right: A banner image is declared as advertising.**

pages – for example, it is not advisable to change the screen location of the main navigation menu for parts of the site.

Figure 4 shows a possible way of extending an existing UWE presentation model [10]. For usability validation, the new "contentType" property provides information about the different areas that make up the page. With the example, only two areas have been annotated: The left part of the screen is declared to contain the site navigation, and the right part is declared to contain the main content.

Graphics As mentioned earlier, using an appropriate model makes it possible to determine whether an image contains text, and to perform appropriate tests. We can distinguish between ornamental graphics which do not need *alt* text, graphical menu entries etc. which need as *alt* text the text of the menu entry, and pictures/charts which need an *alt* text with a short description, or possibly even an additional *longdesc* description.

In our example, we assume that the travel agency's logo should be associated with the agency's name, so it could be declared as non-ornamental graphics as shown in figure 5 (left). When a website is later generated from the model, the tool can ensure that *alt* text is present for the logo image.

If it is known that a menu entry is represented by an image which contains the menu entry's text as a bitmap, this can also be used to improve the checks for text vs. background colour contrast which are mentioned earlier.

Advertisement Users "turn blind" to page areas which they think contain advertisement: Empiric research by J. Nielsen [17] suggests that banner-sized areas tend to be ignored. This can have a severe negative effect on usability if e.g. a site's navigation looks like an ad. Using the information about page areas from within usability validation tools, we can check whether any page areas that are not marked as advertisement in the model actually look like advertisement on screen, either because of animation effects, typical banner sizes or because of their position on screen. Figure 5 (right) shows how a page element could be declared as advertising by settings its "contentType" property.

Liquid layout Using the model, we can easily say which part of the page has the main content. Consequently, the rule that a page's width should adjust to the browser window width can be made more accurate: It is desirable that the *main content's width* increase with the browser window width. Other parts of the layout (such as a navigation bar) may remain fixed. An automatic check needs to verify that the main content's box (in the CSS "box model") does not have a fixed width, which involves checking its ancestor boxes and frames.

Essential content Finally, a tool can alert the user if essential content is missing from the page. Content which should normally be present on every page includes the page creator's identity, a "last changed" note and a link to the site's entry page. Additionally, a complex site's main page will benefit from the presence of a search facility, a "news"-style list of recently updated site content, and other similar items. This check requires from the model that a page area can be labelled with the role of the area's content.

6.2 Navigational and Functional Aspects

Having access to the model also enables a tool to introduce a number of automatic checks (or to improve existing ones) which are related to moving from one page on the site to another.

Search engine support It is possible to ascertain whether all pages with content are accessible by search engines. This can be implemented by first building a list of reachable pages by traversing the HTML pages, and then comparing it to the full list of pages in the model. To improve the quality of the check, the model should specify for each page whether it is supposed to be publically available or only reachable as part of a process which requires manual data entry.

As soon as we know which part of a frameset contains the "main content", we can also determine whether the pages with content contain a mechanism (link and/or JavaScript) to reach the frameset from the single pages. This is important to ensure that if a visitor enters the site via a search engine link, he is not left "stranded" without the navigation aids that are present in other frames of the frameset.

For example, in the extended model from figure 4 (right), an automatic check can determine that TravAgContent contains the main content due to the fact that it is marked with "{ contentType = MAINCONTENT }". If this content is later put into a frame during page generation, the check can be made to fail if that frame does not contain a navigation link to the frameset.

Navigation paths A model-based tool can analyse the possible navigation paths of the site in a variety of ways. Together with user-specified constraints, analysis of the navigation paths yields interesting data which can reveal other shortcomings of the site. For instance, the click distance between arbitrary pages can be calculated. The web developer can subsequently specify e.g. "there should only be 3 clicks from the product view to the final 'thank you for buying' message".

A tool can also warn when general guidelines are not adhered to: From the entry page, "real content" (i.e. either larger amounts of information or interaction possibilites like forms) should only be one or two clicks away. Also, *all* pages should normally be accessible with three clicks. Additionally, if the site is organised in a hierarchical way, the main content should not only contain links to parent or child pages in the navigation tree, but also cross-links to other pages: In our experience, no complex site can be described adequately by a strictly hierarchical navigation tree. Furthermore, menus should not be too long and have an appropriate structure.

Interaction patterns The models of current Web Engineering solutions feature support for certain patterns, such as a "guided tour", i.e. a series of pages connected with "previous"

and "next" buttons. It is possible to offer tool support for automatic recognition of such patterns, e.g. by looking for sequential steps in the model's activity diagrams. This way, it is ensured that typical ways of interacting with a site use appropriate, established interaction patterns.

6.3 Content

When analysing the content, the benefits of additional information from a model become most obvious – as existing tools do not have access to this information, they mostly ignore this aspect.

Intended audience Currently, the models for a website do not allow a developer to specify properties of the site's intended audience. This type of information could be used for a number of automated checks.

For instance, the audience can be assigned a "literacy" value, ranging from "children" to "academic person". An automated check can subsequently warn about site content which is unlikely to be understood by the target audience because it contains too many words which are not part of its vocabulary. Related to this, the ratio of graphics to text would probably need to be higher for a lower "literacy" value to make the site enjoyable by the audience.

Another example for an audience property is its type of internet access. If a significant fraction of the audience are modem users, a rule saying that pages must load within 10 seconds will result in constraints about the size of pages and the amount of graphics they use. With a tool-supported solution, these constraints can be checked automatically. A similar point can be made regarding the type of output device, e.g. desktop computer vs. mobile phone.

Figure 6 shows one possible way of integrating this information into web application development. The properties of the intended audience do not fit easily into the different models in use by current Web Engineering methods. For this reason, it might be better to specify them only at a very concrete level during development, such as in the "project preferences" of a Web Engineering tool. The tool could then perform automatic usability tests of the web application at regular intervals, for example whenever a new version is uploaded to the server during development.

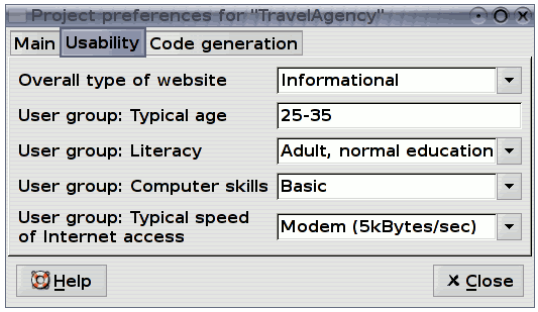

Fig. 6: **User interface design for a "Usability preferences" GUI dialogue in a hypothetical Web Engineering tool. Automated usability validation, performed by the tool, could take advantage of the information given by the developer.**

7 Extending Web Engineering Models

Many usability issues can be solved when using and enforcing guidelines in Web Engineering development tools as outlined above. Some steps can be fully automated whereas others rely on an interactive process that includes the developer and possibly test users. However, to improve fundamental usability concerns, the extension of models is required.

If attributes related to usability are included in the Web Engineering models, this will allow tools to increase usability automatically or to warn the developer when certain guidelines are violated. To summarize the points made in previous sections, we recommend that the following selection of attributes be included in Web Engineering models:

- Information about pages
  - What type of content is positioned where on the page?
  - Which pages provide the site's core functionality to the user?
- Timing
  - Overall contact time of a user with the site?
  - Contact time per visit?
  - How long will the user need for the main tasks?
  - What is the maximum time for delivery of a page?
- Purpose of the site
  - What is the main objective of the web site?
  - What information and navigation complexity is desired?
  - Is the page mainly sensational, educational, or informational?
- Target group, anticipated user
  - What is the main user group?
  - Age distribution of the anticipated users.
  - Computer related skill level of potential users?
  - What infrastructure (e.g. computer type, connection speed) do potential users have?

Timing, site purpose and target group are central to many of the usability issues raised. The concrete attributes in these categories may vary depending on the models and Web Engineering system.

8 Conclusion and Further Work

In this paper, we have analysed existing work in the areas of Web Engineering and automated usability validation. We have demonstrated that Web Engineering solutions currently do not put a focus on usability issues of the generated websites. Similarly, automated usability validators do not take advantage of the additional information stored in Web Engineering models.

However, as our research shows, making validators use information from models could significantly improve validation quality, to the benefit of both of the above areas. In our

analysis of the state of the art in automated usability validation, we have shown that the lack of more abstract information (e.g. in the form of a model) leads to suboptimal performance of the tools; often, the tools just output messages which tell the user to check certain aspects manually.

In our attempt to provide a basis for improvements of automated testing, on one hand we list tests which become possible if the validator has access to a model. On the other hand, the necessary information is sometimes not included in current Web Engineering models – for these cases, we have proposed a number of possible model extensions.

Further work is necessary in a number of areas:

- The ideas in further methods and tools should be examined to see whether they solve some of the issues raised above. For example, WebML takes a different approach to the presentational aspects of modelling. The ideas in [12] should also be analysed in more depth.
- An approach needs to be found for expressing the usability-related information using models or other means, and to integrate the related development steps into existing methods (see section 6).
- Related to the point above, it must be determined what "usability tool support" can look like. Such support is imaginable at various levels. When modelling at a more concrete level, the patterns (like *guided tour*) of existent tools are an example. At more abstract levels, the web developer's knowledge about typical usage patterns of a web application or about the expectations of users regarding website behaviour could be taken advantage of.
- The proposed improved usability tests should be implemented to prove the validity of our arguments. Work on the prototype of such a validator has already begun [2].

Acknowledgement

Parts of the work presented here were funded by the German Federal Ministry of Research (BMBF) in the context of the intermedia research project (`http://www.intermedia.lmu.de`) and by the German Research Council (DFG) in the context of the Embedded Interaction Research Group (`http://www.hcilab.org`).

1. 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*, Budapest, Hungary, 20–24 May 2003.
2. 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, pages 36–41, Springer LNCS 3579, July 2005
3. 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
4. A. Beirekdar, J. Vanderdonckt, M. Noirhomme-Fraiture: Kwaresmi – Knowledge-based Web Automated Evaluation with REconfigurable guidelineS optiMIzation, in *PreProceedings of 9th International Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2002*, Rostock, Germany, 12–14 June 2002.

5. 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, pages 206–215, Springer LNCS 1875, September 2000
6. L. Constantine: Devilish Details: Best Practices in Web Design. In *Proceedings of the First International Conference on Usage-Centered, Task-Centered, and Performance-Centered Design forUSE 2002*, Rowley, MA: Ampersand Press, 2002.
7. A. Cooper, R. Reimann: About Face 2.0 – the Essentials of Interaction Design. Wiley, 2003
8. S. Debnath, P. Mitra, C. Lee Giles: Automatic Extraction of Informative Blocks from Webpages. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, Santa Fe, New Mexico, USA, 13–17 March 2005.
9. J. Gómez, C. Cachero, O. Pastor: Conceptual Modeling of Device-Independent Web Applications: Towards a Web Engineering Approach. In *IEEE MultiMedia* Volume 8, April–June 2001, pages 26–39
10. 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, A. Evans, R. France and A. Moreira, editors. Gesellschaft für Informatik, Köllen Druck+Verlag, pages 158–173, October 2001.
11. F. H. Imai, N. Tsumura, Y. Miyake: Perceptual color difference metric for complex images based on Mahalanobis distance. Journal of Electronic Imaging – April 2001 – Volume 10, Issue 2, pages 385–393
12. M. Y. Ivory, M. Hearts: An Empirical Foundation for Automated Web Interface Evaluation. Ph.D. thesis, University of California at Berkeley, 2001
13. M. Y. Ivory, R. R. Sinha, M. A. Hearst: Empirically Validated Web Page Design Metrics. In *Proceedings of the SIG-CHI on Human factors in computing systems*, March 31 – April 5, 2001, Seattle, WA, USA. ACM, 2001
14. A. Knapp, N. Koch, F. Moser, G. Zhang: ArgoUWE: A Case Tool for Web Applications. *First Int. Workshop on Engineering Methods to Support Information Systems Evolution* (EMSISE 2003), September 2003.
15. N. Koch, A. Kraus: The Expressive Power of UML-based Web Engineering. Second International Workshop on Web-oriented Software Technology (IWWOST 2002), May 2002.
16. J. Nielsen: Alertbox: Current Issues in Web Usability `http://useit.com/alertbox/`, accessed April 24, 2005.
17. J. Nielsen: Alertbox: Is Navigation Useful? `http://www.useit.com/alertbox/20000109.html`, accessed 11 Feb 2005.
18. J. Nielsen: Change the Color of Visited Links. Jakob Nielsen's Alertbox, May 3, 2004 `http://www.useit.com/alertbox/20040503.html`, accessed Apr 20, 2005.
19. J. Nielsen: Top Ten Mistakes in Web Design. Jakob Nielsen's Alertbox, `http://www.useit.com/alertbox/9605.html`, accessed 24 April 2005.
20. J. Nielsen: When Bad Design Elements Become the Standard. Jakob Nielsen's Alertbox, November 14, 1999, `http://www.useit.com/alertbox/991114.html`, accessed 24 April 2005.
21. B. Oestereich, C. Weiss, C. Schröder, T. Weilkiens, A. Lenhard: Objektorientierte Geschäftsprozessmodellierung mit der UML. dpunkt.verlag Heidelberg, 2003
22. siteusability.com – Common usability mistakes. `http://siteusability.com/mistakes.html`, accessed 21 April 2005.
23. Web Style Guide `http://www.webstyleguide.com/`, accessed 5 November 2004.
24. World Wide Web Commitee (W3C): Techniques For Accessibility Evaluation And Repair Tools, Working Draft, 2000. `http://www.w3.org/TR/AERT`, accessed 6 November 2004.
25. World Wide Web Commitee (W3C): Web Accessibility Initiative (WAI), `http://www.w3.org/WAI/`, accessed 5 November 2004.