# Model-Based Automatic Usability Validation – a Tool Concept for Improving Web-Based UIs

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

## ABSTRACT

This paper describes an approach for improving automated usability tool support during the development of websites. Existing usability and accessibility validators only analyse the HTML code of a page when they create a report of possible problems. However, when a web engineering method is used to create a website, additional information is available in the form of models which describe the site. An automated validator can use these models to verify usability guidelines (such as "text on the web page should be easy to understand by the target audience") with higher accuracy. It can also perform automatic validation in situations where existent tools only output instructions for manual inspection by the developer. The paper systematically analyses existent guidelines and tools, and identifies ways in which the use of a model can improve verification quality. An extension to existing web engineering models is necessary to support automated checkers. It specifies properties of the users, the technical platform and the environment of use. A flexible approach allows the models to be used by validators running inside an integrated development environment, but also at a later time, without access to the development environment. Finally, the prototype of a model-based automatic usability validator is presented. It features verification of a number of guidelines which cannot be automated by existent validation approaches.

## Keywords

Web usability, web engineering, automated validation, usability model, accessibility

## Categories and Subject Descriptors

H.5.4 [**Information Systems**]: Information Interfaces and Presentation—*Hypertext/Hypermedia*; H.5.2 [**Information Systems**]: Information Interfaces and Presentation—*User Interfaces*; D.2.2 [**Software**]: Software Engineering—*Design Tools and Techniques*

## 1. INTRODUCTION

The usability and accessibility of a website is a major factor which influences its success. If users have problems navigating the site, providing input to a web application or understanding the text on the web pages, they will quickly abandon the website even if it provides the information or service they are looking for. As a consequence, over time there have been efforts to develop guidelines to improve the usability of a website. They range from fairly concrete rules ("Contrast between text and background colour should be high enough") to more abstract rules ("Documents written to be read online must be concise and structured for scanning") [18].

Today, the area of web engineering [12] provides various methods for efficient, systematic web application development. These approaches use platform-independent abstract models to design the web application and support derivation of the final implementation from the models. An important aspect during this process is to ensure that the usability of the resulting user interface is taken into account. Existing web engineering solutions like UWE (UML-based web engineering [16]) include steps in this direction, e.g. by recommending user tests as part of the method and generating usable default interfaces for user interface patterns.

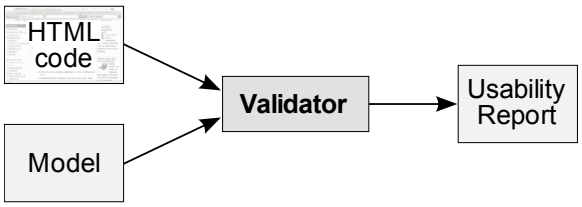


**Figure 1: By processing not only the HTML code, but also additional information from a model, the quality of a web usability validator's output can be improved.**

So far, tools for automatic validation of usability guidelines have not been integrated into web engineering environments. Existing validation programs (section 2) formalize usability rules, apply them to HTML code, and then produce a report to inform the developer of any detected problems. They only process the application's final HTML implementation. Thus, they lack access to more abstract, general information about the developed application. As a consequence, only a restricted set of usability guidelines can be handled by the tools.

Our work takes steps towards improving this situation. It is based on the idea of using the abstract models of web engineering approaches during automatic usability validation (figure 1). This enhancement enables a much broader spectrum of usability guidelines to be formalized and validated. A prototypical validator implementation is also presented. It provides practical insights on how a model-based validator can be realized. The intended usage scenario is as follows:


*NordiCHI 2008: Using Bridges,* 18-22 October, Lund, Sweden

- A web developer uses his web engineering IDE (integrated development environment) to create navigation and presentation models for his web application.
- Apart from the existing models, he also creates a new model which describes properties of the users (age, disabilities etc.), the technical platform and the environment of use.
- During his work of refining the models and creating an implementation from it, automatic usability validation runs in the background at regular intervals.
- As soon as a problem with the implementation is discovered by automatic usability analysis, the developer is alerted to it.

With this approach, the developer can be made aware of problems with his application early during development, when correcting a mistake is still much cheaper than at a later time. Creating the new model involves extra effort, but as this paper shows, this effort is justified because it results in accuracy improvements to the validation process. In many cases, if the application is being developed in a systematic way, information about the target audience etc. will already be available in a requirements catalogue.

Fully automatic validation can never replace other measures such as user tests, but it can complement them; many problems *can* be identified using a program. Furthermore, the developer can improve his website without having expert knowledge about web usability.

After a look at related work in section 2, this paper describes the central parts of a model-based approach to automated usability validation: First, this includes models which specify who the users of the web application are and how they interact with the application (section 3). Second, the paper explains how model information can be employed to improve the accuracy of the validator, to reduce false positives and false negatives, and even allow validation of guidelines which are hard or impossible to automate for non-model-based validators (section 4). Finally, the prototypical implementation of a number of model-based checks is presented (section 5). Its description is followed by a discussion about the evaluation of the concept (section 6) and some concluding remarks.

## 2. RELATED WORK

**Automatic Usability Validation:** The idea of using a model to automatically identify usability problems during development is described in [15, section 5] in the context of GOMS models (goals, operators, methods, selection rules) for desktop GUIs. 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. The USAGE system [7] allows the calculation of learning and execution times for GOMS action sequences. Due to the use of GOMS as the only model, the approach is limited to the detection of problems with the actions taken by the user to achieve goals and sub-goals.

Ivory and Hearst [13] present an extensive overview of different automatic usability evaluation methods. The tool introduced later in this paper belongs to the category "analytical modelling" and supports the activity "analysis" – given a model of the UI and user, it makes a prediction about the expected usability by identifying usability problems. It is interesting to note that despite analysing 132 different usability methods, the authors of [13] state in section 7: "Our survey only revealed automation support for [analytical modelling] methods that focus on user performance". This paper tries to fill this gap: The quantitative approach of measuring user performance is discussed, but the main focus of our tool is a qualitative analysis of web pages.

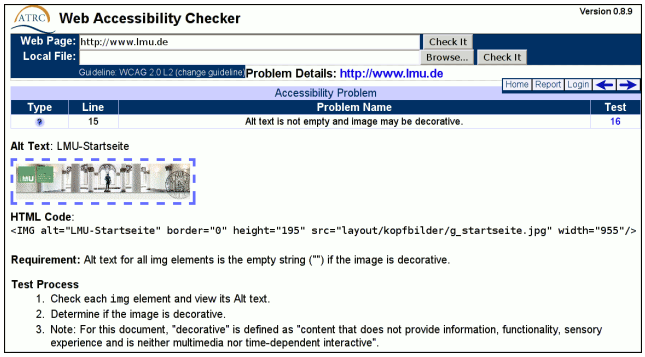


**Figure 2: Problem: Because abstract information about pages is not available to them, current HTML usability/accessibility validation tools (here: ATRC accessibility checker) require the user to check guidelines manually.**

**Guidelines for Improving the Usability of Web Pages:** A large number of different sets of guidelines for accessibility and usability evaluation have been created. The most important ones to this date are probably the W3C's WCAG (Web Content Accessibility Guidelines) 1.0 and 2.0 efforts, which are part of its Web Accessibility Initiative [24]. Other standards which are important in practice are often heavily based on WCAG, such as the German BITV or the American section 508 guidelines. In addition to the W3C documents, many other sources with detailed advice exist, including Jakob Nielsen's Alertbox [20] and the Web Style Guide [18]. The research-based guidelines of the U.S. Department of Health and Human Services [21] are noteworthy because they base their recommendations on published research results. In the context of this paper, a problem with these guidelines is that the language they use to describe usability problems is relatively vague. Still, the sections below show that it is possible to come up with ways to automatically validate many of the rules.

**Existing Web Usability Validation Tools:** Table 1 compares existent web usability and accessibility validators with our tool "Wusab" (section 5). Typically, they are implemented in the form of a web application themselves, only a few are desktop GUI applications. After a URL has been entered in an input field, the validator downloads the page, analyses the HTML and outputs a summary of likely usability problems.

WebXACT (webxact.watchfire.com) is a commercial validator which concentrates on covering as much of the WCAG and section 508 guidelines as possible. Under the name "quality", its results include some page content checks which can be regarded as usability rather than accessibility checks, e.g. finding broken links and measuring the page click depth.

WebTango ([14], webtango.berkeley.edu) takes an unusual approach: Usability guidelines are not implemented in the form of an algorithm for each guideline. Instead, statistical methods are used to calculate the similarity between the website that is evaluated and a set of "known-good" websites whose accessibility and usability has been rated by experts using manual inspection.

Kwaresmi [5] is an academic prototype with an emphasis on quickly specifying additional tests for the validator using a special guideline definition language.

MAGENTA (giove.isti.cnr.it/accessibility/magenta) is an example for a tool which does not only identify problems, but which can also correct some errors if the user wishes. This can be advantageous for web developers who do not have the necessary knowl-

| | 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** [14] | statistics | reference sites (a.+u.) | n/a | site | report? | no |
| **Kwaresmi** [5] | heuristics | accessibility | yes | page | report | no |
| **MAGENTA** | heuristics | accessibility | yes | page | report | repair |
| **EvalIris** [1] | heuristics | accessibility | yes | page? | report | no |
| **ATRC** | heuristics | accessibility | no | page | report + EARL | avoid false pos. |
| **ArgoUWE** [16] | heuristics | – (no acc./usab. tool!) | no | models | IDE warnings | repair |
| **Wusab** [2] | heuristics | accessibility, usability | no | **page, models** | report | no |

**Table 1: Comparison of tools for automated accessibility and usability validation of web pages. Our prototype "Wusab" is the only tool which takes advantage of the information in web engineering models when performing usability validation.**

edge to identify the correct fix for a problem. A similar, XML-based guideline description language is used by EvalIris [1].

The ATRC Web Accessibility Checker (checker.atrc.utoronto.ca, figure 2) is the improved web-based version of the A-Prompt GUI application. Apart from coverage of WCAG 1.0/2.0 and related guidelines, it features output in a machine-readable format (W3C EARL, Evaluation And Report Language). It is laudable in its modularised and systematic approach to WCAG validation. Furthermore, it 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. This way, it reduces the number of false positives.

ArgoUWE [16] is listed for comparison. It is not a web usability validator, but offers model critique for UWE's web engineering models. For example, it will warn if a process use case model does not include any process nodes.

Wusab, whose idea was first introduced in [2], distinguishes itself from other tools in this area because it bases its web usability report on web engineering models as well as the HTML code of the web pages.

**User and Usability Modelling:** Apart from GOMS models which describe interaction at a very detailed level, attempts have been made to model information about users at a more abstract level. Constantine and Lockwood [10] introduce user role models to help the developer with the systematic analysis of software UIs. They also advocate the popular approach to use task modelling.

UsiXML [17] is an approach whose models are useful for our purposes. It attaches context objects to user interface models. They describe the context in which an application is used, including aspects such as the hardware platform, the user, and the environment of use. The models in section 3 are influenced by the UsiXML context model.

For WebML, the Web Modeling Language, a related model was proposed for web applications which automatically adapt their content to context changes [9]. The context information describes the user, his location, the device and further aspects of the environment, but does not provide many properties (such as user goals) that are important for automatic usability validation. Other web engineering solutions either do not feature an explicit user model at all (such as OO-H [8], Object-Oriented Hypertext, which only allows using different patterns for different parts of a site), or only do so for a specific application area (e-learning in the case of early UWE [16] literature).

# 3. MODELLING USABILITY - RELATED ASPECTS OF WEB APPLICATION USE

This section introduces model extensions which are needed by automatic model-based usability validation tools. In web engineering, metamodels are used to describe what the model diagrams of a website look like. To apply our extensions, we introduce a new metamodel which allows attaching context information to a web engineering solution's existing models.

The context model describes the user, the technical platform (e.g. the used browser) and the environment (e.g. "too noisy to hear any audible clues"), and can be associated with existing model classes which represent the pages of the application. In addition to the context model, a few properties are also directly added to existing presentation or navigation models (e.g. to describe the page layout), and yet others are attached to models by means of constraints (e.g. to specify a maximum response time of the web application). Finally, this section also discusses how to embed model information in the final HTML implementation.

The notation used for the models is UML (Unified Modeling Language), the de-facto standard language for modelling. The extensions are designed in such a way that they are applicable to the models of the most important web engineering solutions, including UWE, OO-H and (via its alternative UML-based notation) WebML.

## 3.1 Context Model: User, Platform and Environment of Use

Figure 3 shows an overview of the proposed context metamodel classes in the usability package. They provide the basis for the integration of context and other information into different web engineering models. Due to space constraints, not all details are shown in the figure. UWE [16] is used as an example to show how to add the new elements to an existing set of web engineering models. With a UML «merge» operation of the usability metamodel with the UWE metamodel, the changes can be made available for use by the UWE method and its tools.

The model extension allows the developer to attach a Context class to one or more UiModel classes with the appliesTo association, where the UiModel is intended to be the superclass of an existing web engineering class which represents a web page or a set of pages. In figure 3, UWE's UI View derives from UiModel to permit that the context class EmployeeContext can be attached to the web page MyPage. While creating a web application, the

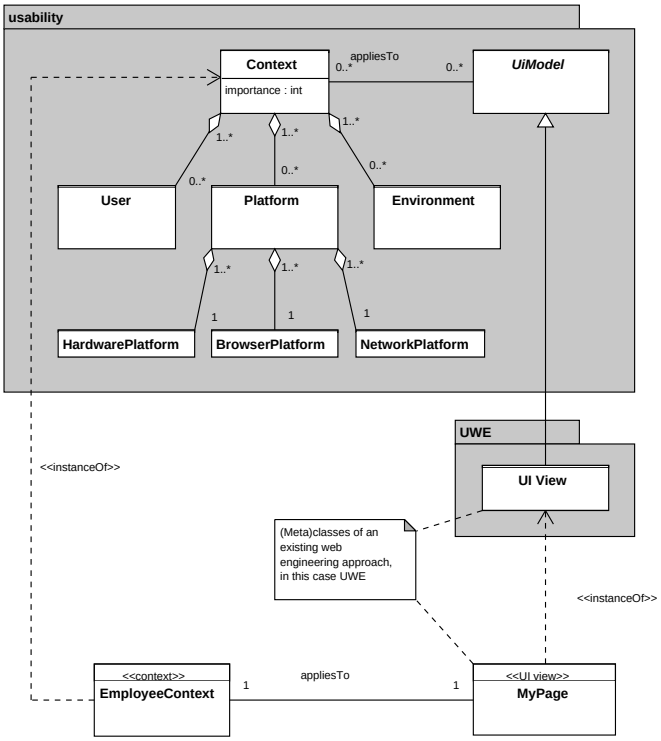


**Figure 3: The new usability package (top) allows attaching context information to the models of an existing web engineering approach like UWE. In the example (bottom), access to the page MyPage is characterized by EmployeeContext.**

developer will typically create context classes by looking at the use cases of the application: There may be different contexts for different user groups (for example normal site visitors, and employees of the company that offers the web application), or the site may be accessed on different types of devices (e.g. desktop PCs and mobile phones), or under varying circumstances (e.g. from home, while travelling etc.).

The extensions were designed by analysing the weaknesses of existing usability validators, such as the fact that they are sometimes unable to implement a guideline check. Figure 2 shows an example: The user is asked to "Determine if the image is decorative" himself. Additionally, the existing guideline documents (see previous section) were examined with the goal of identifying guidelines which cannot be verified by existent validators, but only if additional information about the context of use is available.

**User Model:** The users are described by the properties of the User metaclass. Apart from *basic data* like age and sex, this includes the number of years of school education, the general interaction speed (when reading, clicking etc.) and their relative level of patience (e.g. whether they are in a hurry). Moreover, the users' general *knowledge* (or lack thereof) can be represented (for example "fairly good background in technology", "frequent online shopper", "no experience with e-learning"). This is also possible with their physical *abilities* (e.g. see colours, blinking UI elements allowed, physically capable of inputting text). Finally, the model specifies the *goals* of the users when they access the application, such as retrieving information, performing a transaction or looking for entertainment. The developer can introduce new values for the knowledge, abilities and goals properties. For instance, a particular web shop application can have more detailed goals like buy-book.

**Platform Model:** As shown in figure 3, the technical platform is subdivided into three parts. The HardwarePlatform describes the form factor of the device (desktop PC, handheld, cellphone etc.), the supported user abilities (e.g. audio output enables the "hear" ability), and the available input and output devices (e.g. colour display, audio output, keyboard) together with parameters such as screen size. The BrowserPlatform contains details about the browser running on the device, e.g. the rendering engine, supported output media (screen, print etc.) and whether support for technologies like JavaScript is present. The third part, the NetworkPlatform, specifies the speed and cost of Internet access, and the reliability of the connection.

**Environment Model:** Additional influences on the user experience can be included in the Environment metaclass. First, the environment can limit the available user abilities. For instance, the user's "hear" ability will not be effective if the environment is too noisy to hear the device, or the user may not be allowed to look at it while driving a car. The environment may also be more stressing than average, which results in less patience and greater likelihood of user errors. Finally, general usability requirements for the web application can be specified via name tags. For example, the tag "wcag" can mean that the site must conform to the W3C's WCAG, possibly because it belongs to a government institution for which this is required by law.

## 3.2 Extended Presentation Model

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 later in this paper, 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.

Figure 4 shows how the web engineering presentation model of UWE is extended: A new role attribute (inspired by W3C's XHTML role [25]) is added to all UWE metaclasses which represent parts of the page layout. This way, a role such as "navigation", "main content" or "ornamental" (i.e. just a part of the visual page design) can be assigned to these layout areas. Figure 5 demonstrates with an example how roles are assigned. The following roles (slightly extended from [25]) were identified as being useful for automated usability validation: main (main content), seealso (e.g. sidebar about main content), secondary (secondary content), logo, ad, empty, ornamental, navigation, search, login, help, contact, other.

## 3.3 Extended Navigational Model

Just like the role property provides additional information about parts of a page, a related annotation is necessary for the entire page. Similar to the procedure for presentation models, Contexts can be attached to navigation metaclasses. Additionally, the metaclass for each page is augmented with a purpose property. This property mirrors the goals of the user model – if a user and a page share at least one goal/purpose, then the developer intends the page to be used by the respective user group.

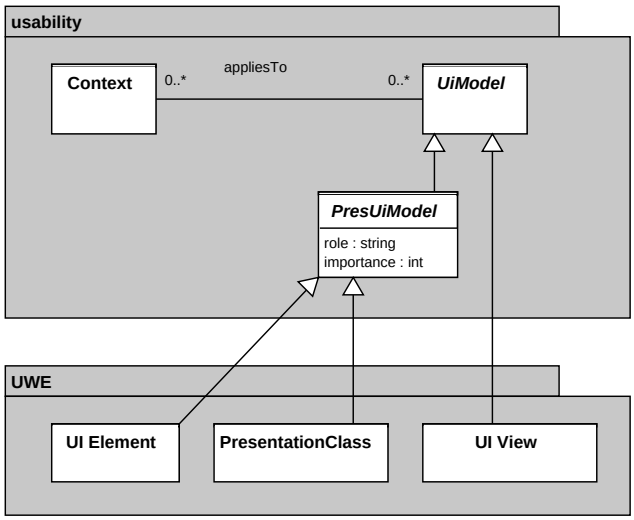


**Figure 4: The presentational model is extended by adding a role attribute to metaclasses which represent layout areas. It describes the area's purpose, e.g. "this is the navigation menu".**

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 such as OCL (Object Constraint Language). This way, the usability validator can evaluate the constraint expression using its knowledge about the application and its users. OCL functions which can be supported include the time it takes to download a page (which depends on the size of the HTML data and the type of network access of the device), the number of bytes or the cost of downloading a page, the number of clicks or the time it takes to traverse a navigation path, or the time it takes to achieve a goal (such as buying an item in an online shop).

## 3.4 Embedding Model Information in the Implementation

While it is possible to store usability-related information in the models of a web engineering environment if the validator executes inside an IDE, there are several reasons why it may sometimes be convenient to add the information from the models directly to the web application's (X)HTML pages, or in other files which are publically available from a site's webspace: First, it should be possible to perform validation at a later time in the same way as allowed by non-model-based validation services, i.e. possibly even by a third party instead of the original developer of the site. In this case, only the implementation will be available – the web engineering tools should automatically embed the models to ensure they remain accessible. Second, embedding model data in a standard way allows tools to access usability-related information independently of the exact modelling approach used. Finally, some models, in particular the presentation model, are not stored in UML form even in some of the UML-based web engineering environments. Thus, to add e.g. the role property to a UML presentation model, it would sometimes be necessary to create this model first.

When making both models and implementation available in the same place, it is desirable to use the same web-based technologies for both. This also facilitates further applications which make use of the model data. For example, it would be possible to partially automate the analysis of log data from user tests, e.g. by comparing expected with actual times to achieve a goal.

Models which are not directly related to any single HTML page, such as the navigation and context models, are better stored separately from the source code of the web page rather than in an embedded form. A reference to them can be added to every page to which they apply. One possibility for this is the use of the XMI (XML Metadata Interchange) format to store the data. A variant of the <link> tag can be used in the head of the HTML page to reference the model data, for example <link rel="model" href="/navmodel.xmi" />. As XMI is a standard model interchange format, most UML-based models can be represented with it.

Information that is directly related to a single (X)HTML page can be embedded directly in its source code. In particular, this is useful for the presentation model. Various approaches for embedding data in HTML have been formulated over time, such as W3C's GRDDL [22], RDFa [23] or Microformats (microformats.org). As many web pages are today still implemented using HTML rather than XHTML, the prototype in section 5 implements an approach inspired by Microformats. It is based on the idea of using the names of CSS (Cascading Style Sheets) classes in the HTML code to semantically annotate the markup. For the role property from the extended presentation model, parts of the layout can be assigned a role by adding a class attribute to the HTML code, e.g.  if the page area is enclosed by a . With the prototype, the class name is "wusabType_" followed by a string for the type of role, such as navigation, main content etc. In the future, support for the XHTML role attribute [25] should probably also be added.

# 4. IMPROVING GUIDELINE TEST QUALITY USING MODEL DATA

While automatic validation will never be able to replace manual testing and user tests, this section attempts to demonstrate that there is still room for improvement: The validation quality can be improved noticeably if the validation program has access to additional information about the HTML code that it is analysing. When existing web engineering methods are used, some useful high-level information is created by the developer in the form of the different web engineering models, such as the presentational and navigational models of UWE [16] or OO-H [8]. However, it is also possible to use other sources of information, for example manual input by the developer, or extensions to models as introduced in the previous section. Finally, log data from user tests or the live website can be included in the analysis, but this is beyond the scope of this paper.

For a number of example guidelines taken from [18] [20] [21] [24], this section presents typical examples of usability-related information that is needed. In each case, it lists a guideline, gives suggestions on how validation of the guideline could be implemented, and specifies the required abstract information that must be provided by the model. In some cases, automatic validation becomes possible for guidelines that cannot be handled by existent validators at all. In other cases, there is only an incremental benefit: The accuracy of the usability report can be improved by taking model information into account. This way, the number of false positives (reported usability problems which do not actually exist) and false negatives (problems that the validator fails to recognize) is reduced.

## 4.1 Presentational Aspects

**Guideline:** *The maximum recommended width of page content is x pixels* (compare: [18, ch. 4] [20, 31 July 2006], partially [21, 6:8])

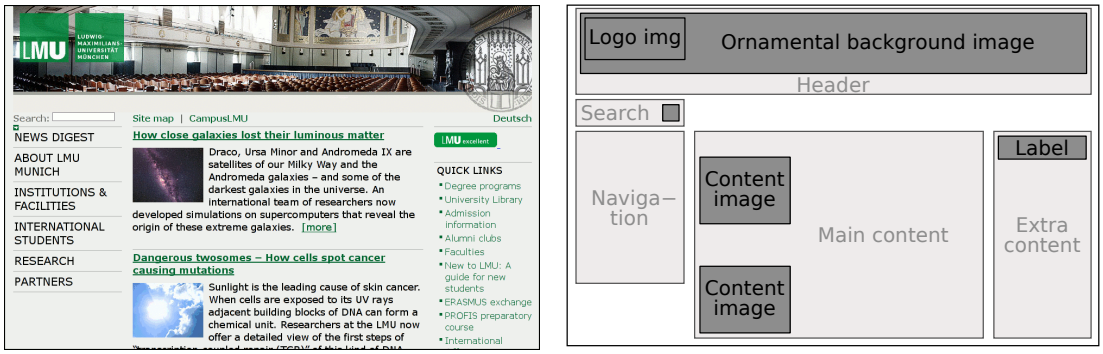


**Figure 5: Model-based usability validation analyses a page (left) together with associated models, e.g. a presentation model (right). This allows tests to become more accurate: The validator can check whether the page layout follows the de-facto standard. It can also distinguish between normal images (non-empty alt text required) and ornamental ones (which should have alt='').**

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. However, a more indirect approach is also imaginable: For example, if the device is just "desktop PC" without further details, but the expected audience of the site is children, it will often be the case that the children's PC is an old machine with a lower-than-average screen resolution.

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". It can distinguish between important and less important content via the presentation model's importance property (see figure 4).

*Each web page should contain items like a menu, link to the homepage, search facility, modification date...*
(compare: [18, ch. 2] [21, 7:1/14:5])

For a human, it is usually easy to distinguish between different types of content on the page (e.g. between navigation and advertising), and thus to determine whether they are present. An automatic validator can perform this task by using the presentational model of a web engineering solution.

*The layout of the page should consistently 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...*
(compare: [20, Top Ten Mistakes in Web Design, Nr. 8] [18, ch. 4] [21, 6:2/7:5/14:5])

Again, using information from the presentational model, the validator can determine how the different content areas are arranged on the page, 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.

*The area occupied by different types of page content should be within certain bounds*
(compare: [20, 10 February 2003] [21, 5:9])

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. Only the first screen of the page (the area typically visible without scrolling) should be taken into account to get meaningful results.

The prototype introduced in section 5 implements the three guidelines above: It checks for the presence of certain required page areas, for the use of a de-facto layout, and for a reasonable subdivision of screen space.

*The width of a web page's main content should adapt to the width of the browser window*
(compare: [20, 31 July 2006] [18, ch. 4] [21, 6:8])

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 non-model-based 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 the role in the presentation model) which part of the HTML contains the main content.

## 4.2 Navigational Aspects

*Links for navigation paths should have certain characteristics*
(compare: [21, ch. 10] [18, ch. 6])

Existing usability validators can 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. A validator can check whether these important links are placed 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, 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.

Using OCL constraints as outlined above, the developer can also express requirements such as 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 users should not have to wait more than 10 seconds for a page to display. Based on knowledge about average user behaviour, a model-based validator like the one presented in section 5 can make guesses as to whether the constraints are likely to be met in practice. However, real users' behaviour is influenced by too many factors when they use the site. Thus, user tests are still necessary. In previous work, we have introduced tool support for this case: UsaProxy [4] is a logging solution which allows detailed recording of user actions. By using the information supplied by the model, semi-automated analysis of the logging data generated during a user test becomes possible. For example, the log data can be used to verify whether visitors do indeed need two minutes for filling out the forms.

*Navigation patterns should be implemented in appropriate ways*

Models like UWE's navigation model allow the developer to specify patterns which describe the intended type of interaction in a more abstract way. 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 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 will be done correctly, but the developer may have supplied a different layout or graphical design for the pattern, or influenced the generation in other ways.

The validator can also offer 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, then it might be desirable to prefer efficiency over intuitiveness: The users may be faster when they fill out one large form than when they have to click through several pages and skip over explanatory text for form entries.

## 4.3 Image and Text Content

*Images should have* alt *text, unless they are decorative*
(compare: [18, ch. 4] [24, WCAG 2.0, guideline 1.1])

The alt attribute is required by the HTML standard, so it must always be present. Automatic usability validators flag its absence as an error, and some validators even do this if the attribute is empty (alt=""). However, empty alt text makes sense if the image in question is only decorative in nature, an invisible spacer image, or similar. If the validator knows from the page's presentation model that certain images are decorative (i.e. its role is "ornamental"), it can allow empty alternative text. For other images on the page, empty alt text generates a warning. Figure 5 shows an example: Most images such as the site logo or a "start search" label are associated with functionality, but the background image at the top of the page is ornamental.

*Write texts in a style that is tailored towards the web*
(compare: [20, 15 March 1997] [18, ch. 6] [21, 16:2])

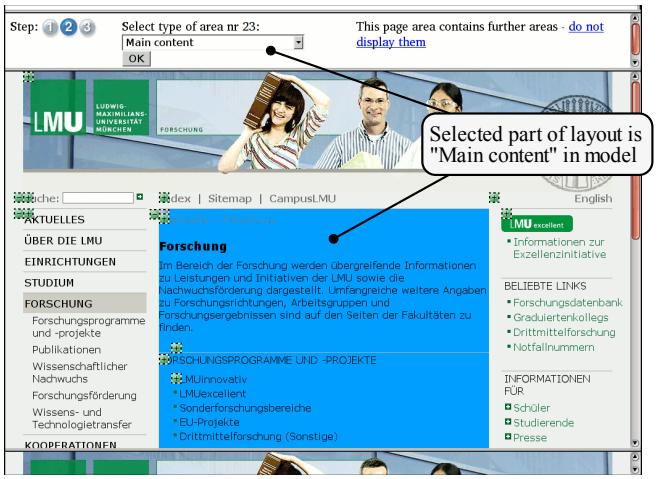


**Figure 6: The Wusab prototype can perform automated usability validation based on model information that is embedded in the HTML or created using a web-based interface.**

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 user model, such as the one introduced in section 3.1:

- The goals (e.g. educational vs. entertainment) have an influence on the vocabulary and on how much effort readers will invest in understanding complicated sentences.
- The age affects the way pages are read, in particular when comparing children to adults.
- The education (e.g. "10 years of school") 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), 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 [20, 14 April 2002] whereas adults prefer a concise style, e.g. with bullets. The readability of a paragraph of text can be validated automatically to a certain extent. The reading level is the school grade 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 like SMOG [19]. 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 [11]), 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 must 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 section 5.

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 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 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 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 7: Excerpt from the results for different guideline tests when running the prototype validator on the web page from figure 5. Other implemented tests include a calculation of the number of years of education required to understand the text, and an analysis of the vocabulary used by the page. With access to model information, some new tests become possible, others become more accurate.**

# 5. WUSAB, A TOOL FOR AUTOMATED MODEL-BASED VALIDATION

To demonstrate the feasibility of some of the ideas from section 4, the prototype of a model-based automated usability and accessibility validator, "Wusab", has been implemented. The program is implemented as a Java-based web application. A 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 1, validation should be integrated into a web engineering IDE so that a report is only presented to the developer when problems are identified.

As discussed in section 3.4, several ways exist to attach the model information to the web page. The prototype currently supports embedding the information in the HTML using class="..." attributes. In addition to this, users are also free to edit the embedded presentation model of the web page manually using a web-based interface. This interface (see figure 6) is displayed after the validator has downloaded the page, but before the guideline checks are run. Initially, any already embedded information is shown and can be changed, or a page without embedded information can be annotated.

Figure 7 shows an example for typical output generated by the tool. The depicted results are part of the usability report for www.lmu.de. The prototype implements the following usability tests:

**Text vocabulary:** Using word lists of the most frequent 1000 English words, the next frequent 1000 words (both based on the General Service List), and the Academic Word List, the validator can determine whether texts are likely to require a college/university education. This corresponds to a knowledge property of "science" in the user model. In practice, further wordlists for other values of the knowledge property would be needed to make this validator test accurate with a wider variety of sites.

**Text complexity:** The complexity of the text on the web page is analysed, and a readability grade (required years of school education) is calculated using a number of different algorithms, including Flesch-Kincaid, FORCAST and SMOG Grade Level [19]. The program is careful only to take real text into account, it ignores style information and scripts which are embedded in the page.

**Text/background contrast:** The CSS properties are parsed to determine the contrast, and output a warning if it is not high enough. Information from the user model can be used to adjust the warning threshold; for instance, older users are more likely to have problems with low contrast values.

**Alternative text for images:** The alt attribute must be present for all <img> tags. If the image is ornamental according to the presentation model, the alternative text must be empty, otherwise it must be non-empty (figure 5). In comparison, most existing validators require all images to have alt text. At most, they advocate a manual check for ornamental images (figure 2).

**Layout analysis:** The different parts of the layout should be arranged in a way which follows one of several allowed de-facto standards appearing in the literature (section 4.1). For example, the navigation should be to the left of the main content, and the logo should be at the top left.

**Area size analysis:** The relative size (in pixels) of page areas like the main content should be within certain limits. For instance, the prototype considers it an error if the main content occupies less than 30% of the entire page. The thresholds are configurable, the defaults are derived from [20, 10 February 2003]. Both area size and layout analysis are implemented by rendering the page in a normal browser. Next, custom JavaScript code is executed in the context of the rendered page, it extracts information about the different page area rectangles from the DOM (Document Object Model) tree. The current version of the tool loads the page into the browser which is used to access the web service, but it would also be possible to perform this step using an embedded rendering engine which can run on a server without a visible user interface.

**Analysis of page width at different resolutions:** The prototype determines whether horizontal scrolling will be necessary to view all content at resolutions of $1024 \times 768$ and $800 \times 600$, again using custom JavaScript which is executed when the page is displayed at these resolutions.

# 6. EVALUATION / DISCUSSION

Looking at the description in the previous section, it becomes clear 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 were performed with a variety of real-world web pages whose presentation model was manually created using the web-based user interface. The prototype's reports 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 hard to objectively compare the new, model-based validation concept with existing validators that only access the HTML code of a page.

One way would be to hand-select a set of sites which optimally demonstrate the improved accuracy of model-based guideline tests, but this selection of the best possible examples could be considered unfair towards existing validators. For example, unlike other validators the Wusab prototype can distinguish between ornamental and non-ornamental images for the alt text check. Model-based validators can also alter thresholds (e.g. for contrast or text complexity) depending on user age or school education, which is bound to produce a more accurate usability report.

An alternative approach would be to select a large random sample of sites and to compare results. Here, our model-based prototype would be at a disadvantage, due to the fact that it implements only a few example guidelines, whereas commercial validators like WebXACT have support for many more. There would be no direct way to weigh the higher quality of a few model-based tests against the larger number of tests with potentially lower accuracy.

A further difference which makes comparison difficult is that models are not usually available for real-world websites. They have to be created manually in the way that the evaluator *assumes* they might have been specified. Apart from the fact that the result might not match the original site author's intentions, this approach would also fail to take another advantage of model-based validation into account: The model should be created early during development, and repeated validation should avoid the introduction of usability problems as early as possible, not when the site is finished.

At a conceptual level, the work of Brajnik [6] is suitable to examine whether model-based usability validation creates higher-quality reports. It characterizes the quality of a validator report using the three attributes completeness, correctness and specificity. The first two are related to the number of false positives and false negatives in the usability report: A *complete* report includes all usability problems that are present in a site, but possibly also additional, incorrectly reported problems. A *correct* report only includes correctly identified problems, but not necessarily all that are present. An ideal validator would produce reports that are both entirely complete and correct. The third attribute, specificity, is related to the level of detail in which a problem is described by the report. Certain characteristics of our model-based validation approach strongly suggest that all three aspects will be improved:

**Greater accuracy of checks:** With model-based validation, as outlined above many checks can be made more accurate: A model-based validator does not include a single, fixed implementation for a particular guideline, but parameterizes it depending on properties of the user, the device that is used to access a site, or the environment in which it is accessed. This results in a more fine-grained report and thus higher specificity. For instance, error messages can be as detailed as "*the word 'specificity' is unlikely to be understood by your target audience*".

**More checks become possible:** Model-based validation permits tests which cannot be performed reliably by existent validators due to lack of information. For example, this includes vocabulary checks (depends on knowledge from the user model) or page layout analysis (depends on the role property). Thus, report completeness is improved, as the number of false negatives drops. Additionally, the number of false positives may also drop, since many messages like "check manually whether the image is ornamental" (figure 2) can be eliminated.

**Fewer checks may be necessary:** Complementary to the previous point, a model-based validator also knows when *not* to perform certain guideline checks: For instance, if the website is not intended to be viewed on mobile phones, it is not relevant (and thus no usability problem) if the layout does not adjust to a very small screen size. An error message to this effect could be considered a false positive – by not displaying it, the correctness of the validator is increased. In contrast, non-model-based tools must take a conservative approach and assume that any recognized usability or accessibility problem may apply for the site being evaluated. This also means that for non-model-based tools, the number of checks cannot be easily extended e.g. to also take mobile devices into account, as too many irrelevant problems would be reported to site designers who do not target mobile devices.

# 7. CONCLUSION

In this paper, we have described a concept to extend the functionality and scope of web development environments: Automated model-based usability validation can take place continuously in the background while the web developer creates the models and implementation of a website. This way, the validator can highlight potential usability problems of the site at an early stage, when correcting them is still easy.

When a validator has access to models which describe the website to be validated, it becomes possible to verify usability guidelines which could previously not be checked automatically due to lack of abstract information about the website. For example, it can be verified whether the page uses the de-facto standard page layout, which reduces the risk that new visitors get confused by the page. Additionally, other guideline checks (which are already implemented in some existent validators) can be made more accurate, so the number of false positives is reduced. An example for such a guideline is checking for sufficient contrast of text and background colours – high contrast is more important for visitors with bad vision, so the guideline can be assigned a higher priority depending on the target audience that the model specifies.

This paper proposes a model extension for the specification of information that is relevant for usability validation. An additional model is necessary because existent web engineering solutions' models do not include all the data required by a usability validator. In particular, information related to the user (age, education etc.), the platform (e.g. device form factor, cost per minute) and the environment (e.g. noisy) is absent. The introduced context model also serves as an additional layer which makes the validator independent of the web engineering solution that is actually used. Furthermore, the model data can be useful for other, related tool concepts, such as automated analysis of log data from user tests.

A working prototype demonstrates the central ideas of the approach. It implements tests for a number of usability guidelines. While a direct comparison with existing validators (which do not rely on model data) is difficult, there are strong indicators that the tool concept is conceptually superior to usability validation which solely relies on access to a site's HTML data.

There is room for future work in several areas: First, additional guidelines can be identified, and suggestions can be made as to how to implement them with higher accuracy in a model-based validator. Also, the integration of usability validation into web engineering IDEs can be investigated. This comprises issues such as machine-readable output of usability reports (e.g. using W3C EARL), suppression of false positives, and possibly an extension of the concept to not only identify problems, but also aid in solving them. Finally, an objective comparison of existing and model-based validation should be conducted, e.g. by letting experts estimate the accuracy of the two methods for a number of examples.

Fully automatic usability validation will never be able to replace user tests or manual inspection by experts. However, as this paper shows, it is possible to push the quality of usability reports much further, allowing the evaluation of aspects which cannot be validated by existent tools. This way, the validator can more effectively support developers who have little experience with or interest in usability issues, and the cost of design mistakes is reduced since problems are identified earlier during development.

**Acknowledgement** This work was funded by the German BMBF (intermedia project). We thank Andreas Singer and Ronald Ecker for parts of the prototype implementation.

## 8. REFERENCES

[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* (poster), 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, 36–41, Springer LNCS 3579, 2005

[3] R. Atterer, A. Schmidt, H. Hußmann: Extending Web Engineering Models and Tools for Automatic Usability Validation. In *Journal of Web Engineering*, vol. 5, no. 1 (2006), 43–64

[4] 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

[5] 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

[6] G. Brajnik: Comparing accessibility evaluation tools: a method for tool effectiveness. In *Universal Access in the Information Society* journal, vol. 3, no. 3–4, Springer, 2004

[7] M. D. Byrne, Scott D. Wood, P. Sukaviriya, J. D. Foley, D. E. Kieras: Automating Interface Evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 1994 (CHI1994)* , Boston, MA, USA, 1994

[8] 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, 206–215, Springer LNCS 1875, September 2000

[9] S. Ceri, F. Daniel, M. Matera, F. M. Facca: Model-driven Development of Context-Aware Web Applications. In *ACM Transactions on Internet Technology (TOIT)*, vol. 7, issue 1, February 2007

[10] 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, USA, 1999

[11] A. Coxhead: A New Academic Word List. In *TESOL Quarterly*, vol. 34, no. 2, 213–238, 2000

[12] Y. Deshpande, S. Murugesan, A. Ginige, S. Hansen, D. Schwabe, M. Gaedke, B. White: Web Engineering. In *Journal of Web Engineering*, vol. 1, no. 1 (2002), 3–17

[13] M. Y. Ivory, M. A. Hearst: The state of the art in automating usability evaluation of user interfaces. In *ACM Computing Surveys*, vol. 33, no. 4, 470–516, December 2001

[14] 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, March/April 2001

[15] 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

[16] A. Knapp, N. Koch, G. Zhang, H.-M. Hassler: Modeling Business Processes in Web Applications with ArgoUWE. In *Proceedings of the 7th Int'l Conference on the Unified Modeling Language (UML2004).* Springer Verlag, 2004

[17] 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, 2004

[18] P. J. Lynch, S. Horton: Web Style Guide: Basic Design Principles for Creating Web Sites, 2nd edition, Yale University Press, March 2002

[19] G. H. McLaughlin: SMOG Grading – a New Readability Formula. In *Journal of Reading*, vol. 12, 639–646, May 1969

[20] J. Nielsen: Alertbox: Current Issues in Web Usability, ISSN 1548-5552, `http://www.useit.com/alertbox/`, accessed February 25th, 2008

[21] 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

[22] World Wide Web Commitee (W3C), Dan Connolly (ed.): Gleaning Resource Descriptions from Dialects of Languages (GRDDL), W3C Recommendation, September 2007, `http://www.w3.org/TR/2007/REC-grddl-20070911/`

[23] 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

[24] World Wide Web Commitee (W3C): Web Accessibility Initiative (WAI), `http://www.w3.org/WAI/`, accessed March 19th, 2008

[25] 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, W3C Working Draft, 2007, `http://www.w3.org/TR/2007/WD-xhtml-role-20071004/`