Using established Web Engineering knowledge in model-driven approaches

Ralf Gitzel*, Axel Korthaus, Martin Schader

Department of Information Systems, University of Mannheim, Germany

Received 2 August 2005; received in revised form 18 April 2006; accepted 2 September 2006

Abstract

A lot of research is currently conducted in the field of Model-Driven Development (MDD), especially regarding its applications to specific domains. Another field that enjoys a great amount of popularity is the Web. As a result, one of the domains MDD is applied to quite frequently is that of Web Applications. However, Web Engineering differs significantly from general Software Engineering and a number of well-established non-MDD solutions already exist in that field. This leads to several interesting questions, which have been left unanswered so far. In this paper, we address this shortcoming by analyzing whether the problems encountered in the field of Web Engineering can really be solved with MDD approaches. We also answer the questions whether MDD will be able to solve these problems better and/or cheaper than traditional Web Engineering approaches and whether the current Web MDD propositions live up to this potential. While answering these questions, we will show that there exists a great synergy between the two groups and that the success of MDD in the Web domain will depend on exploiting the strengths of both.

Keywords: Model-Driven Development (MDD); Model-Driven Architecture (MDA); Web Applications; Web design languages; Web Application frameworks; Code generation

1. Introduction

A recent trend in Software Engineering is the Model-Driven Development (MDD) movement which has stirred up quite a bit of controversy with its claims of being able to fulfill the promises, which were made (but supposedly not kept) during the CASE era. Basically, MDD’s goal is to allow designers to rely on their design models as the only required document for the various tasks connected to the different stages in the life cycle of a software application. While a generic solution applicable to arbitrary problems is currently considered utopian by most researchers, domain-specific approaches in well-specified domains are seen as a good starting point for MDD prototypes.

One domain that seems to be a popular target for domain-specific MDD approaches is Web Engineering. As will be shown later, there are many approaches that claim to realize the MDD paradigm in the Web domain already. However, unlike that of other domains, Web Engineering endeavors are seen as distinct from general Software Engineering
projects (cf. Glass [42], Deshpande and Hansen [27]). So while there is already a strong discussion about MDD in general, there is so far no paper analyzing some very crucial questions with regard to Web MDD. These questions are:

- What are the problems of Web Engineering and how are they traditionally addressed?
- Which of these problems can be addressed by MDD approaches?
- Are MDD approaches better than the traditional ones and therefore fit to replace them?
- If yes, do the currently available MDD approaches live up to this potential?

In order to answer these questions, this article starts with a brief discussion of Web Applications and their unique properties in contrast to general software leading to a list of specific problems to be addressed. In turn, we will examine several concepts helpful for the realization of Web Applications, examine how each of them helps to solve the inherent problems, and illustrate them with examples in order to answer the first question. Next, the strengths and properties of MDD approaches generally identified in the literature are described. It will be shown that traditional Web Engineering approaches can be incorporated into MDD approaches and that MDD also provides additional solutions to Web-specific problems, thus providing a positive answer to the second and third question. Finally, we examine several existing Web-MDD approaches and show that the full potential of MDD is not yet realized for Web Applications.

2. Properties and problems of Web Applications

The World Wide Web has revolutionized modern life. While evolving from a largely text-based collection of information to a world-wide multimedia-capable client–server application, the technology behind the WWW has undergone vast changes, so far without abandoning its basic paradigm. It is for this reason that “Web projects” differ from most other software projects and so the question arises if standard Software Engineering processes are suitable for this field (cf. Glass [42], Deshpande and Hansen [27]). Therefore, the unique aspects of Web Applications should be analyzed and taken into consideration by any domain-specific approach in this field.

The advantages of Web Applications are the introduction of a standard client, thus reducing the required learning effort of users and the costs to create an application’s interface [29, 67 (Chapter 3)], the ability to perform all updates in one place (i.e., on the server) [64, Chapter 3] and increased application portability [64, Chapter 3]. On the other hand, new kinds of problems occur, some of which are technical and others are social or computer-human interaction issues.

2.1. Technical properties and problems

The unique paradigm of Web Application development leads to a couple of technical problems, which need to be addressed in order to avoid complications during software development and maintenance. Some of the key features of Web Applications are their dependence on the HTTP and HTML protocols for communication and the inhomogeneous client-side environment whose exact set of technical and human-related properties is most likely unknown due to a multitude of different browsers, plug-ins, and platforms as well as the global nature of the Web (cf. [29, 67 (Chapter 3)]). Offut [79] identifies a similar heterogeneity on the server side because of the distributed use of various technologies. Another interesting difference to traditional software is the fact that it is technically trivial to switch to another server providing the same service without significant loss of time or money [44, 86, 79 (pg. 10)]. These properties lead to several problems that have to be addressed in order to develop successful Web Applications.

The lack of state incurred by the use of HTTP is the reason for some of the most fundamental problems of Web Engineering. Originally, the Web was intended for the display and intelligent linking of text. If one considers that the Web is a hypertext system whose concept goes back to Vannevar Bush’s hypothetical MEMEX machine for storing and rearranging information from printed books in the 1930s (see [60]), it is not surprising to experience a vast conceptual gap in modern Web Applications (cf. [37]). While the basics of the lack of state issue have been addressed by now using cookies or URL extensions (cf. [64, Chapter 3]) there are still some problems to address in individual implementations. Chief among these is flow control, which is made more complicated because back buttons, caching, and stored URLs allow virtually any state transmission [67 (Chapter 3), 79 (pg. 25)].

Many researchers describe compatibility issues due to the clients’ varying capabilities of dealing with the information transmitted from the server side. Besides, different browser versions may have different script and HTML interpretations, screen resolutions, and color settings. Of even greater impact are variations in hardware characteristics,
for example, with regard to bandwidth and the quality of output devices, which can range from small WAP-capable mobile phones to non-visual output media such as screen readers for blind people (cf. [29,39,67] (Chapter 3),77 (pp. 42–44)). As a consequence, software tests are a lot harder to perform than in the case of traditional software [27,42].

Nevertheless, this additional testing effort pays off even from a business standpoint, as reliability is a major factor in Web site quality. The main reason for this is the ease with which users can switch to other Web sites [11,37,79]. Given the unstable technological foundation of Web Applications, reliability is more difficult to achieve than in some other fields of Software Engineering where stable technologies have allowed the establishment of solutions for specific reliability problems [79].

A related problem is performance, as people are unwilling to wait long for pages to load. Offut points out that this is generally not a problem on the server side, the bottleneck being the available bandwidth and its jitter [79]. Therefore, Web pages should be designed to load reasonably quickly regardless of network capabilities and client performance ([29,77,79] (pp. 27–35), also cf. [11]). On the other hand, more complex applications might very well have server-side performance problems when serving a large number of clients, at which point the issue of scalability has to be considered, even if only as a basis for future expansions [39,67 (Chapter 3),86].

Due to the importance of uninterrupted reliability, performance, and scalability as well as the pressure from competitors, Web Applications undergo constant maintenance instead of fixes being grouped into release packages ([37,42,72], and [79]). Besides added workload, compatibility issues have to be considered [79] and the interface should be designed in a way to accommodate these changes [72]. Also, maintenance should not reduce the availability of the page as any down time will lead to revenue losses since people cannot use the site during that time and are likely to look elsewhere [79].

Of particular importance when developing Web Applications are security issues ([11,27,37], and [79]) because the loss of customer information can lead to loss of credibility, legal problems, repair cost, and revenue loss (cf. Offut [79]). Lack of security is not only a danger to the site itself but also to the internet as a whole, making the internet one of the most important problem areas in connection with information technology today (cf. Neumann [71]).

2.2. Usability issues and properties

While usability problems definitely exist on the Web and many papers and Web sites identify them, there is also a lot of disagreement on the level of abstraction and the required scientific foundation. For example, some sources see the problems in the details (e.g. Nielsen [74–78]) while others believe in guidelines situated at a higher level (e.g. Shneiderman [90, pg. 65]). In addition, some sources are opposed to generic guidelines, demanding rigorous empirical study (cf. Ivory et al. [55]) while others indicate that guidelines can be given if backed up by empirical testing (cf. Catarci and Little [17], Neuwirth and Regli [72], Nielsen [76, pp. 10–11], and the metrics given in [56]). An extreme position is held by Paolini who questions the value of empirical studies on Web usability on the ground that it is impossible to find representative users of the application due to the global nature of the Web [81]. However, such discussions are beyond the scope of this work and the usability issues considered are those most commonly identified in literature.

It should be noted that the focus of the following collection is on issues particular to a Web environment and that it is not unlikely that some of these goals are conflicting and/or incompatible with external conditions. For example, Becker and Mottay point out that the strategic goals of an organization might put less emphasis than required on some of these factors, most likely because they conflict with its external goals, for example the aspired time-to-market [11, 12].

2.2.1. Consistency

An important, maybe even the central, factor for a Web site’s usability is its consistency in appearance/design and behavior both internally and with regard to the Web as a whole [12,37,97 (pg. 65)]. Without consistency one of the main advantages of the Web, the uniform client interface, is partially or completely negated. While the desire to stand out and attract customers is a legitimate one, it should not be achieved by reducing the usability of the site. A consistent Web site supports a high degree of predictability, enhancing the user’s ability to foretell what results the interaction with the Web page will yield [97 (pg. 65),37].

Internal homogeneity is achieved by giving the pages of a Web site the same theme and by establishing conventions such as creating a common layout or using the same icon for the same purpose every time [16]. As a rule of thumb,
Garzotto et al. suggest to treat “conceptually similar elements in a similar fashion and conceptually different elements differently” [35]. Since many organizations have established precise guidelines for a “corporate identity”, at least a consistent look can currently be considered a standard in professional Web pages.

Consistency with the other sites on the Web is also important because few users are willing to learn how to use a Web Application that differs from what they are used to. For this reason it is a bad idea not to comply with “Web conventions” by using different link colors [74] or introducing unexpected link behavior such as pop-up windows [77]. Currently, a wide range of established “design standards” exist such as the standard layout [11] including a logo and a navigation sidebar [75] or naming conventions [11] such as calling the main page of a site the “home page”.

2.2.2. Navigability and site structure

The ability to reach the intended target page is a major factor for usability and many researchers identify navigation and site structure as two of the main usability concerns in Web Applications (cf. Becker and Berkemeyer [11], Nielsen [76], and Paolini [81]). Nevertheless, many Web sites still suffer from problems in this context although the establishment of the “Web conventions” mentioned above has helped to make this problem less pronounced.

The purpose of navigation tools such as a sidebar with links is to give the user the required sense of orientation, not only identifying pages he can reach but also the current position within the Web site [76, pg. 188]. Without a solid concept of the structure of a Web site, users will quickly become lost due to the additional dimensions of movement as opposed to the forward and back in traditional text [95,96].

Nielsen identifies several types of Web site structures such as trees and tables, both of which are commonly encountered on the Web. While a tree-based navigation will lead users deeper into increasingly concrete categories, table-like structures define categories based on the values of the table’s attributes. A linear structure similar to that of traditional media is considered to be useful only in exceptional cases [76, pp. 15, 200–207]. Regardless of the structure, a good home page is important and should give a good overview of the site as it is the starting point for most users [76, pg. 166] — in fact the purpose of a Web Application should be immediately obvious to an incoming user [78,95].

Once a Web site’s structure has been designed, links have to be created, which emphasize this structure. While it is possible to use individual links, a preferable approach is to use navigational patterns such as site maps [74], indices [46], or guided tours [46]. However, not all links are used for navigation within the structure of the site. Other link types identified by Nielsen are associative links which lead to pages containing additional information on a certain topic and reference links [76, pg. 51]. These external links add to the quality of a page but they must be chosen carefully (cf. [76, pp. 67, 70]).

In addition, the design and realization of individual links can also raise problems if not done properly. Generally, the name of the link should be concise and hint at where it leads to [16], giving the application what Garzotto et al. call self-evidence [35]. On the other hand, there should be no explanatory comments near the links [16]. Typical problems are pages that link to themselves [78], orphan pages (i.e. pages whose parent Web site cannot be identified by those who arrive there from a search engine) [78], and broken links.

2.2.3. Visual presentation

The visual presentation is a very important aspect of usability ([11,21], or [76]), covering not so much aesthetic quality but rather how supportive the visual design is for solving the problems the interface was designed for. In fact, Constantine and Lockwood claim that if the aesthetic design is considered more important than usability, the site quality will suffer from it [21]. The same can be said about a focus on technological innovation, which often implies the addition of new multimedia features, as opposed to a focus on quality content and service, particularly if this focus leads to errors or problems for the user [74,75]. When looking at problems of visual presentation, psychological aspects play an important role and a lot of literature exists on how to properly design the visual presentation for all kinds of media. However, these complex issues are beyond the scope of this work and only a few general examples will be given.

It has already been mentioned that the performance of a Web Application is directly influenced by the graphical opulence of its Web pages. Furthermore, the heterogeneous nature of the clients has to be taken into account, i.e., it is prudent to design Web pages in a way that makes their content available to a wide variety of heterogeneous client technologies. This design strategy can be pursued by using generically applicable Web pages, which for example use
scalable fonts [77], or by dynamically creating different versions using some form of stylesheets [76, pp. 27–36, 77], including printable versions for different paper formats [76, pg. 93].

Besides these more technical issues, the visual design should **support the intended task of the user of a page**. For example, the use of high contrast and the avoidance of distractions such as animations can help improve readability [54,77,79 (pp. 126, 143)]. Ivory et al. even claim that “good pages have relatively fewer graphics” [55]. On the other hand, Horton points out that by using interactive animations the advantages of computer applications over traditional paper media can be fully realized [51]. These positions are not necessarily mutually exclusive — it seems sensible to use graphics sparingly and activate animations only when the user is interacting with a graphic.

Moreover, pages are often inconvenient with regard to “what-if support”. Nielsen feels that many users would benefit from being able to compare products listed in a table but that many sites will only display a single product at a time, for example many airline sites only allow specifying a specific date for a flight instead of a range [78].

Generally, a Web page’s **text** should be written in a way that allows readers to skim over it and easily identify the parts relevant to them. This can be done by keeping the text short and dividing it into easily graspable chunks. The highlighting of keywords, clear categories and terms, as well as the limitation to one idea per paragraph are additional measures to improve readability ([76, pp. 101–111], also cf. [16]). Usability is also improved by avoiding scrolling [74,77] and headers which take up too much room [16].

### 2.2.4. Internationalization

Given the global nature of the Web, internationalization is a major concern for those Web sites intended for a multicultural and heterogeneous audience. However, internationalization is not limited to the translation of languages but also includes a consideration of the cultural background of potential users [95]. A lot of work has already been done in the field of internationalization. For example, Becker and Mottay cite global marketing strategies such as those of McDonald’s and Coca Cola as good examples of successful localization. The principles of these strategies offer a good basis for Web localization in turn [12]. Marcus and Gould propose using cultural dimensions such as those identified by Hofstede [49] as a basis for guidelines [66].

Rosson and Carroll identify two approaches to internationalization — standardization and localization. **Standardization** helps facilitate the design process by offering standard solutions and best practices and makes it easier for users to adapt to new systems. However, variations in design are constrained and innovations possibly blocked [83, Chapter 10.2.1]. **Localization** on the other hand strengthens cultural diversity but is more expensive than a global interface. While avoiding culturally offensive symbols and terminology is also possible for standardized user interfaces, other issues such as language as well as formats for date or currency are not that easily addressed [83, Chapter 10.2.2].

A similar distinction of two internationalization approaches is proposed by Nielsen [76, pg. 315]. Becker and Mottay offer three solutions for localization. The first, a common design for all sites, is very similar to standardization as described above except for the adaptation of the site language. The other options are custom design or a hybrid solution with some common and some custom-designed sections [12].

An effort to provide internationalization incurs several problems. The most obvious are **added workload, redundancy** and, as a result, additional costs [12]. Concrete **design problems** include the cultural influence on page layout, colors and terminology (cf. [12,66]). For example, Arabian or Hebrew applications will have a **layout** which reflects the different reading direction (cf. [13,27 (pg. 235)]). The choice of **color** reflects the cultural background as well, e.g., the color red acts as a warning signal in Western cultures but not in Asian ones (cf. [12,50,51]). Similarly, certain **symbols** such as hand gestures used in icons can be highly offensive in some cultures [27 (pg. 235),54]. A special consideration of **terminology** is required because literal translations sometimes sound archaic or are inappropriate in the context where they are used [12]. The locally applicable measurement units and formats for time and date should be used and time zones should be considered when announcing events [21,79 (pg. 318),90 (Chapter 10.2.2)]. Finally, certain **processes** might vary from country to country such as the point in time when taxes are added to a price and shown in a bill [19].

While most cultural influences on Web page design stem from cultural symbolism or local customs, it is also possible to classify cultures according to certain criteria or **cultural dimensions** and use this information to further localize the interface. Marcus and Gould propose to use information on factors such as the role of the individual as opposed to the collective or a culture’s tendency towards strong hierarchies as a basis for Web design. For example,
a culture with strong hierarchies would be more likely to include information about the leaders of an organization on that organization’s Web Site (cf. [66]).

Other cultural issues include different ethical values such as privacy concerns or local laws which should be addressed appropriately, for example by displaying the proper disclaimer messages [19]. Becker and Mottay suggest enhancing user profiles with cultural and regional information, the latter including technical details such as the lower bandwidth in certain areas of the world [12].

2.2.5. Accessibility

Accessibility has become an important issue recently; in fact, many researchers worldwide identify accessibility as a key usability problem (e.g. Becker and Berkemeyer [11], Deshpande and Hansen [27], Ginge and Murugesan [37], Rosson and Carroll [83], and Shneiderman [90, pg. 42]). Addressing the needs of physically challenged people helps limit the problem of the digital divide [83], which is particularly important in contexts such as e-government [90, pg. 38]. However, accessibility also makes sense from a business viewpoint as it draws customers, which are likely to stay with a site that caters to their specific needs [79 (pg. 298),97 (pg. 42)].

Many aspects of accessibility are technical, e.g., the special input and output devices needed for disabled persons. However, accessibility is also a usability problem because Web pages have to be designed in a specific way to support special output devices, e.g. by providing useful and short ALT text for pictures in order to allow the efficient use of screen readers for blind people [78]. Other helpful properties in Web pages are dynamic enlargement, larger fonts, greater contrast, and larger mouse targets [83]. Additionally, Nielsen feels that a focus on content rather than presentation can also help accessibility [76, pg. 298].

Shneiderman points out two interesting aspects of accessibility and suggests addressing its issues from the early design stages on. As many of the accessibility guidelines intersect with those for general usability, an accessible Web site can also benefit non-disabled users. Also, accessibility touches some core aspects of a Web Application’s code. Thus, adding accessibility at later stages is going to be far more expensive than incorporating it from the beginning [90, pg. 42].

2.2.6. User- and domain-related issues

According to several researchers, it is beneficial if information can be presented in different ways to accommodate the different skill levels and personalities of readers, for example as part of a personalization feature ([32,97 (pg. 44)], also see [11]), with the goal of making applications easy to use [35]. Similarly, Becker and Mottay claim that a user’s perception of usability depends on his or her profile, which includes not only the skill level but also gender or age among other things [12]. Personalization means an adaptation of the Web pages sent to the client based on the personal parameters of the user. The simplest example is addressing the user by his personal name. More sophisticated examples include added shortcuts for professional users.

Of equal importance is the influence of the domain (cf. [55,84]). According to Roth et al., “considerations of the users and the tasks they will be performing [. . . ] should be central drivers for system design specification” [84, pg. 164] instead of generic guidelines. The goals and constraints in the application domain play a central role [84]. In a similar vein, Rosson and Carroll state that “user interface design issues depend on the context of use” [83, Chapter 10.2.1]. By incorporating domain-specific constraints, a Web Application can achieve a high degree of goal-orientation [30], focusing on the user’s tasks instead of the structure of the Web site’s parent organization [76, pp. 15, 380].

2.3. Additional problems

While technical and usability problems form the core challenges of Web Application design, additional problems can be identified. Many researchers see ethical and legal problems as an important issue (e.g. [71,95,27]). These include intellectual property [95,71,27], measures aiming at the protection of privacy [71,27], accountability [71], responsibility for the content [95], and regard for liabilities for damages caused [95]. Some of these questions can become quite difficult, for example, Nielsen feels there might be potential copyright issues when using frames to display content from other Web sites [76, pg. 91].

Problems are also incurred by the content of Web sites. Becker and Berkemeyer identify information content and customer service as major quality factors [11]. Outdated information [74] is a problem which should be avoided as well as a lack of important information. While the latter seems to be obvious, there are many Web pages where the
important information such as the prices for different flights is difficult to obtain, requiring many steps even though the information could easily be displayed on a single page in a table [77]. Outdated information on the other hand reduces trust [75] but must not be confused with archives, which are an important feature [75].

The unknown nature of users is also a problem during the design process as Web Applications have a strong focus on the “user experience” [21]. According to Deshpande and Hansen, conventional Software Engineering processes are based on an analysis of user requirements which is hard to conduct without knowing the desires of the potentially global users [27]. On the other hand, CotS software products for a mass market such as word processing applications are also created for an unknown user, yet they reasonably match most people’s requirements.

Finally, Nielsen considers credibility of a Web site to be very important and achievable by appropriate Web page design. He suggests using a professional appearance and a user-friendly information policy on the pages [76, pg. 91].

While it seems that, at a first glance, these questions are beyond the scope of this analysis, whose goal is to identify how suitable Model-Driven Development is for Web domains, these issues should be kept in mind because the solution for many of them can at least be supported by technical features. The best example for this is accessibility, which is in essence an ethical problem (cf. Theng [95], Shneiderman [90, pg. 42]) but has many general design solutions.

2.4. Summary

As has been shown, Web Applications can suffer from a wide range of problems due to their unique nature. The main categories (summarized in Fig. 1) are technical and usability issues as well as some other problems such as legal complications. The arrows in the figure illustrate potential connections between certain kinds of problems and it is interesting to note that some problems of different categories are related in some way. Based on the analysis in this section, it is now possible to evaluate the quality of existing Web Engineering approaches and to judge the applicability of MDD to Web Engineering problems.

3. Conventional approaches to Web Engineering

In order to identify the potential of MDD for Web Applications, it is important to look at conventional approaches to Web Engineering to grasp their strengths and weaknesses. Ginige and Murugesan see Web Engineering not as a subset of Software Engineering but rather as a holistic and multidisciplinary approach which includes contributions from
areas such as graphic design and Process Engineering [37]. However, in the context of this work we limit ourselves to the software development aspects, as mainly these are relevant for our analysis.

3.1. Web design languages

Web design languages are an established tool for the creation of hypermedia systems. Although some researchers criticize the lack of attention paid to the modeling of state changes in Web Applications [79,86], there are some recent publications which include interactive aspects beyond navigation such as forms (cf. [8,86]), and therefore Web design languages are still of interest in the context of modern Web Applications. This section discusses the potential of Web design languages with regard to the problems identified above and describes some existing examples.

Web Design languages can help facilitate solving some of the aforementioned issues by providing suitable design primitives. While the aspired level of abstraction requires a transparency with regard to most technical problems such as compatibility issues, it could be possible to include model elements to support such things as the design of scalable software with deployment diagrams. Also, it can be argued that a powerful modeling language allows the description of solutions, which are then easier to implement.

On the other hand, design languages are far more useful with regard to usability problems. Many researchers are convinced that usability has to be considered from the beginning to avoid costly changes or support later on (cf. [11, 21]). For example, by modeling standard layouts available for pages throughout the site (in their first incarnation called tabletops, cf. Trigg [96]) and other reusable model components, the consistency of a Web site can be promoted. Similarly, personalization can be modeled by introducing parameters and rules based on their values. Links and page elements can be tied to constraints which relate to some sort of user data (cf. [28,32]).

The true strength of Web design languages, however, lies in the support of navigation, whose importance has already been stressed in Section 2. It should be obvious that a graph-like model can be very helpful in designing and understanding a Web site. However, such a solution only scales to a certain degree — more complex sites will produce a tight lattice of links which will be hard to discern (also see [34]). This problem has been addressed in the early days of hypertext research and while it is hard to identify the original source, it is safe to say that the browser and the index-like filebox concepts of Notecards [46] as well as the guided tour as proposed by Trigg [96] are among the earliest examples. The idea is to define access structures to collections of pages which imply certain links but do not show them explicitly in the model. Good examples are indices (i.e., pages listing all pages they reference in a linear manner) and guided tours (in this context considered a linear list of linked pages arranged to address a certain topic of interest). By using indices hierarchically, a tree-like structure can be imposed on a Web site. Fig. 2 illustrates the relationship between explicit links and navigational structures. When indices are included in all subpages as it is often done to show the tree-like structure (i.e. in page 1 and 2 in the figure), the explicit link model quickly mushrooms.
Abstract navigational contexts can also help to avoid the problem of links, which only make sense in a certain context (cf. Paolini [81], Trigg [96]). For example, if a page is part of two contexts, any links to the context root or other elements of the context will be confusing if they appear in the wrong context. In the explicit link model on the right side of Fig. 3, page 3, which is part of two contexts, has a link to each context root. However, if the page is accessed from the index of the home page, the link to page 2 is confusing. When modeling links using navigational contexts, such problems can be circumvented by including the correct link implicitly. Contexts can have other uses, for example when modeling dynamic behavior by using context information as implicit parameters for an operation performed by the user (cf. to the concept of constrained navigation in [8]).

While Web design languages are capable of addressing some of the important issues of Web Application design, they are not without their problems. For example, Catarci and Little believe that in many cases people will be unwilling to learn a new language [17]. In fact, our impression is that Web design languages are often ignored by practitioners due to their lacking adherence to established standards (also see Section 4.5).

As an impressive number of Web design languages exist, it is not possible to do this group of notations justice in this context. For illustration purposes, one of the classical languages, OOHDM, is briefly described, followed by one of the more recent languages called WebML. The section concludes with some examples of alternative approaches. For additional information on Web design languages see Koch [62] or Gellersen and Gaedke [38].

3.1.1. The Object-Oriented Hypermedia Design Method

One of the most popular examples of a Web design language is the Object-Oriented Hypermedia Design Method (OOHDM) (see Schwabe and Rossi [87] for details). Although it is an older approach, it is still often quoted as an inspiration or as related work because it nicely illustrates how a model can abstract from implementation details without requiring additional creative input for an implementation.

The basis for a hypertext system in OOHDM is the conceptual model, which is a UML class diagram describing the data and program logic on which the application will be based. The associations between different classes suggest possible but not actual navigational connections.

The navigational model plays a central role in OOHDM and can be seen as one of many possible views on the conceptual model. The design primitives, which help to avoid a tight web of individual links in the model, are nodes, links, and access structures. A node is a view on or an observer of one or more conceptual classes and has several attributes and link anchors. Links can originate from anchors and are otherwise self-explanatory navigation primitives. Access structures provide a more sophisticated form of navigation, allowing access to sets of links called navigational contexts.

A navigational context is a set of nodes and nested contexts. The nodes contained within a context may be defined via an enumeration of its members or via certain properties. For example, a context may be defined to contain all
objects of a class, all targets of an n-ary link, or a dynamic set such as the history of pages accessed so far. More complex models are the “class based group”, a context that contains (probably disjoint) subsets of objects of a class separated according to some criterion, and “link based groups” which follow the same principle for links. The content of a context is considered ordered and is accessible via an access structure, i.e. a collection of implicit links that can have several forms. Typical examples are indices, guided tours, and index guided tours as described above.

By defining a separate interface model, the navigational design is kept clean of interface-related concerns. The navigational constructs are effectively masked by interface objects, at least from the end user’s point of view. These objects are based on the concept of Abstract Data Views (ADV) (Cowan [23]). Each ADV can be customized via parameters and can handle certain events (such as a user’s clicking on a link). It effectively defines the behavior and organization (i.e. layout) of the interface. An ADV may be composed of other ADVs based on the composite pattern. The ADV displays the data provided by a navigational node and can follow the links provided by it. The events handled by an ADV can have two main effects. First they can lead to a transition to another node, the typical behavior expected for clicking a link. However, more complex behavior may also be modeled with OOHDM, e.g., changes to the display of the current node.

A recent extension to OOHDM, proposed by Schmid and Rossi, allows the modeling of processes in a Web Application. The approach is based on entities which store information and business processes with activities. A business process is a special context, which can be considered a statechart diagram that can be left at any point, leading either to a suspension, or a restarting of the process [86].

3.1.2. The Web Modeling Language

Ceri et al. propose a “notation for specifying complex Web sites at a conceptual level” [18] called the Web Modeling Language (WebML), which is founded on HDM, RMM, OOHDM, and other previous languages (see 3.1.3). WebML consists of four orthogonal perspectives. The first is the structural model, which describes the data foundation of the site, e.g. as a UML class diagram.

The most important aspect is the hypertext model which consists of two submodels. The composition model defines views on the data called data units consisting of subsets of the attributes found in classes of the structural model. Other units such as the index unit or the scroller unit define access structures on specific data units. The units can be grouped into pages. The navigational model links the units to provide a logically connected entirety. These links do not only represent the application’s hyperlinks but can also occur between related elements within a single page. Contextual links allow one unit to give a context for another, e.g. the correct index of individual elements can be identified due to a contextual link to a certain class. Non-contextual links ignore the context of the originating unit. There are also shortcuts for certain Web patterns such as a multi-step index, i.e. an index pointing to several indices.

The presentation model is based on style sheets, which can be defined for the pages found in the navigational model. Of greater interest, however, is the personalization model, which is based on a user and group principle. The group information can serve as a basis for business rules and other mechanisms to alter the presentation of the application to the user. Also, due to the separation of structural and navigational model, different views on a site can easily be established. WebML can be used as a basis for model-driven development of Web Applications as the models can be represented in an XML-based format [18].

3.1.3. Other Web design languages

Besides the two examples described above, many other Web design languages exist. Due to a large conceptual overlap, these languages are not discussed in detail, for example, the Relationship Management Methodology (RMM) [53] is similar to OOHDM regarding its structure and model elements. For a more detailed discussion of these similarities see Gitzel et al. [40].

One of the earlier examples of a Web modeling language is the Hypertext Design Model (HDM). Focusing heavily on a hierarchical definition of the data forming the hypertext application’s foundation, HDM does not offer the different layers typical of other languages. It rather makes assumptions about the navigational structure of the application by providing implicit links which allow navigating the data tree and switching between different perspectives of the tree leaves called units. An application can consist of several data trees called entities, which can be connected by so-called application links and are in turn organized by a tree-like structure called an outline which allows an initial access to the application [34].
An interesting group of Web languages are object-oriented languages. Besides the prominent example of OOHDM described above, there are other examples such as WebComposition [39]. Fröhlich et al. propose a metamodel for hypermedia design which is pretty similar to the other Web design languages of that time and supports the use of a class diagram as a basis for its conceptual model. In contrast to the other languages, the definition of a formal metamodel offers several advantages, giving a precise “grammar definition” instead of an informal description (cf. [32]). While an object-oriented approach might at first seem counterintuitive due to the special paradigm of the Web, it becomes more interesting in the light of object-based frameworks (see below).

Some languages focus on a particular aspect of usability. For example, the Web Site Design Method (WSDM) is, in the words of its authors, a “user centered” rather than “data driven” approach [28], which focuses on personalization using different perspectives on the data, each with a focus on the needs of a particular user. The metamodel proposed by Fröhlich (see above or [32]) also allows personalization by using a model of user data to design constraints.

The dynamic behavior of Web Applications as opposed to Web sites is a recent issue which is addressed by some extensions to existing languages [86] or new language variants such as W2000, which proposes to add new kinds of navigation, i.e. free and constrained, which identify the data implicitly used by operations potentially attached to a link [8].

3.2. Web Application frameworks

The Web design languages discussed so far are quite useful for the handling of certain usability issues, especially with regard to navigation, but offer little potential for the solution of the technical problems described above. This section examines the potential of Web Application frameworks in this regard and describes several existing examples, analyzing to what extent they live up to this goal.

Clearly, the strength of Web Application frameworks lies in the support for the solution of technical issues, for example by incorporating middleware to realize an N-tier architecture which improves security, scalability, and maintainability (cf. Offut [79]). However, advantages described by practitioners such as Wang encompass automatic validation and transfer of client-side data into objects, the support for cross-browser rendering and internationalization, and a reusable controller component as key benefits of Web Application frameworks [98]. It is interesting to note that these advantages mostly help the application developer but do not improve the quality of the application per se. Even internationalization in this sense is limited to mere technical support for providing several versions of a Web site, for example with resource bundles.

Nevertheless, frameworks can help solve the technical problems identified above. The introduction of state into the normally stateless HTTP protocol was a major challenge for Web Application implementation, which can be solved with cookies and the (possibly persistent) server-side storage of user and state information. In addition, timeout schemes for sessions, as well as a clear recovery path, are needed (see Kolawa et al. [64, Chapter 3]). Web Application frameworks can encapsulate technical details and provide complex flow control on a higher level of abstraction. Similarly, the level of abstraction for security can be raised by providing features at a higher level of granularity. Increased abstraction leads to better understandable code, in turn facilitating maintenance and testing (to some degree).

Web Application frameworks can also provide several output modes, possibly transforming the content to provide a format suitable for the various client platforms. In this way, the compatibility issues can be handled automatically to a large degree.

Solutions to scalability, reliability, and performance can also be provided on a general basis. One way is to adapt the output to different bandwidths automatically, to provide automatic server-side caching and to keep down the number of requests to the server. However, the latter is difficult as it is normally impossible to update only parts of the page, unless using JavaScript or something similar, which has its own problems [64, Chapter 3].

Similarly, support for usability is theoretically possible but has not yet been a major goal of Web Application framework designers. So far, there is only support for some aspects. For example, the consistency of Web sites can be ensured by providing uniform templates or by using composites of subpages to allow the reuse of elements (compare to the concept of tabletops as defined by Trigg [96]). Internationalization is often supported by auxiliaries such as resource bundles. Other features are not realized yet, despite the suitability of frameworks for these tasks. For example, the navigability of a Web site can be improved by providing a sensible abstract navigation scheme, which could possibly be based on the context concept introduced by Web design languages (cf. [40]), or by providing different
views on the same piece of information (cf. Garzotto et al. [36]). The “What-if” support demanded by Nielsen [78], personalization, and accessibility could be implemented as standard modules, encouraging developers to incorporate these aspects.

It is important to note that even if there is support for these features, this does not mean that the usability of a Web site will necessarily improve. Unlike technical problems which are easily measurable, usability issues are less tangible and even if provided with the right tools, developers will still be able to produce unusable software, especially when the design given to them is poor. Additionally, due to the large investment of time into the development of such a framework, it should be designed to be usable for a wide variety of problems. Therefore the offered solutions must not be too narrow. This view is reflected in the existing examples for Web Application frameworks.

The great popularity of the Web as a medium of communication has created a multitude of different frameworks and APIs, which are intended to help with the implementation of Web Applications. There are many different approaches and therefore only a small subset can be presented in this context. For further information see Wang [98] or the overview presented in the context of the Barracuda project [9]. In order to illustrate the statements just made, we take a brief look at some examples currently in use.

A good example of the support for security is Apache Turbine [6]. It integrates a role-based security concept that requires practically no additional implementation. Without judging the quality of the security mechanisms of Turbine, it can be said that hiding the details of security within a Web Application framework is the right approach in the light of reliability and maintenance aspects. Another aspect illustrated by Turbine is the support for consistency through frameworks. Turbine allows a modular creation of Web pages, including different layout “templates” and a dynamically created navigation bar.

One of the most popular open source Web Application frameworks is probably Apache Struts (cf. [4]). Struts’ core concept is the Model 2 architecture, a variant of the MVC pattern, providing a controller which can use various technologies for its model and view. The technological requirements on the model are quite lax, allowing for a wide variety of solutions including Java Database Connectivity (JDBC) [92] and Enterprise JavaBeans (EJB) [26]. Although Struts’ focus is on application interactivity, Kamm and Klein criticize the rudimentary concept used for storing state in Struts as insufficient for true user dialogs and propose an extension to solve this problem [59]. While the choice of view technologies was originally limited to the somewhat problematic JavaServer Pages (JSP) [85], leading to the proposal of Struts variants such as Struts CX [54] or Model 2X [70], the recent addition of the Struts Tiles concept, XSL Transformations (XSLT) [20] and Java ServerFaces (JSF) (see below) has helped to increase the decoupling of data and presentation, the lack of which was criticized by Gitzel et al. [40]. Struts can be seen as a good example of the shortcomings of earlier frameworks and the actions taken in recent years to remedy them. As can be seen, Struts’s support for usability issues has increased. Yet, it still requires a good design model to create a user-friendly application, as stated above.

The Apache foundation is also host to a number of other frameworks. Another example is Apache Cocoon, a publishing/Web Application framework using XML/XSLT to prepare data for output. As it is based on a pipeline principle, several transformations can be performed along with aggregations of different XML sources, all of which can be defined in the so-called sitemap [3]. Like the other Apache projects mentioned here, Cocoon is based on the Java Servlet technology, which means that the well-known solutions to scalability problems developed in this context (e.g. load balancing) can also be applied to the Apache frameworks.

A case in point of the ability to support internationalization is the WebWork Framework [80], a Java Web Application development framework stressing code simplicity, interoperability and developer productivity. Its internationalization concept is based on resource bundles, i.e., collections of text fragments that are applied according to the language required.

A recent trend already mentioned is that of moving away from the traditional Web paradigm and using Web pages to “emulate” traditional desktop applications, i.e., a windows-based graphical user interface. The best-known example is SUN’s JavaServer Faces [93], which is also used in a new subproject of Struts called Shale. Other examples include the frameworks proposed by Puder [82], Echo [73], Jakarta Tapestry [5], and Swinglets [57]. Although it is to some extent a question of personal taste, trying to emulate the event-based paradigm of a windows-based GUI with stateless Web technology can be considered problematic. Despite the arguments in favor, such as improved portability and a richer user interface, we doubt that the potential compatibility problems are worth the benefits, barring major changes to the HTTP protocol. One reason is that Nielsen sees pull-down menus in Web Applications as a usability problem because they violate the established paradigms (cf. [76, pg. 195]). It is reasonable to assume that the same argument...
can be applied to other non-Web GUI features as well. Also, custom GUI components which make use of low level events (such as mouse movement) either won’t be supported or will result in massive traffic, a problem that Puder outlines in his paper [82].

Still, there is the possibility that this last aspect is about to change. Fresh impetus for the design of interactive Web Applications is given by the new AJAX development style currently emerging. AJAX stands for “Asynchronous JavaScript and XML” and incorporates already existing technologies in a new way. Its core benefits are standards-based presentation using XHTML and CSS, dynamic display and interaction using the Document Object Model, data interchange and manipulation using XML and XSLT, as well as asynchronous data retrieval using XMLHttpRequest [33].

Essentially, JavaScript is used to implement an AJAX engine, which is downloaded to the client in order to render the user interface, perform simple tasks such as data validation and to communicate with the server on behalf of the user. In that way, both richness and responsiveness of the Web Application can be increased. By decoupling user interaction from the server, asynchronous communication is achieved, so that the client does not block and waiting times can be minimized. The AJAX approach is currently tested in numerous projects. Google, for example, uses the AJAX technique extensively in all of its recently introduced applications like Google Maps, GMail etc. AJAX is also supported in Web Application development frameworks, e.g. in the Wicket framework [100].

It will be interesting to observe whether AJAX will be able to prevail and actually overcome some of the current limitations of the Web. However, we see several problems at the moment. First, there are those difficulties currently associated with JavaScript, like the error-prone dynamic typing concept, an inconvenient OO approach, the temptation to write browser-specific code, and a lack of tool support. Also, given the rise of mobile devices with very different display capabilities and browsers, the use of AJAX jeopardizes core advantages of the Web such as the high level of platform independence. For these reasons, our arguments regarding MDD will stay within the context of the established standards.

Besides the general frameworks mentioned already, there are uncountable other solutions on the market. Of some interest are the Expresso Framework, which is based on a finite state machine (called Controller object) [58], the OO Navigator which is based on OOHDM but is unfortunately not available to the public (cf. [87]), and an academic project called JESSICA [10] by Bartha and Schranz which focuses on content management.

4. MDD approaches to Web Engineering

It has already been mentioned in the introduction that the core concept of MDD is to make the design model of an application the central artifact of the whole software lifecycle. Also, a generic solution is currently considered a strategic goal at best and the current focus rests on domain-specific approaches in well-specified domains. In order to identify the benefits MDD provides to the Web Engineering domain, it is necessary first to briefly describe its requirements and potentials. A more detailed discussion of these issues can be found in the literature (cf. [88,97]). Next, it is described how existing solutions to the typical problems of the domain can be incorporated into MDD. Based on these facts, it is argued that MDD will eventually contribute to the state of the art of Web Applications but that the current code generation approaches do not yet qualify as model-driven.

4.1. The goals and potential benefits of MDD in general

While there are many different interpretations of what the benefits of MDD are or will be, it is possible to identify two often-cited central goals. One of these goals, in the words of Atkinson and Kühne, is to “reduce [the primary software artifact’s] sensitivity to the inevitable changes that affect a software system” [7]. This quotation forms a roof for a catalogue of sub-goals and benefits resulting from the aspired robustness. It is argued that this resistance to change will ultimately result in a reduction of cost, the main selling argument of MDD approaches [68]. Other authors such as Mellor and Balcer [67], Bézivin [14], Selic [89], or Booch et al. [15] see the main goal of MDD in an increase of abstraction which makes the design of complex systems easier. Since the abstraction from programming-level issues makes the involvement of domain experts easier (see Völter [97]), possibly by providing several different views on the same system (cf. Kent [61]), MDD might be considered related to or maybe even a subcategory of the field of End-User Development (see Stutcliffe and Mehandjiev [94]).
One aspect considered prone to change by many supporters of MDD is the technological platform an application is based on. By focusing on a model which is in synch with the software artifacts at all times, MDD supporters hope to achieve a separation of business requirements and technology \[47,68\] and therefore a certain degree of platform independence \[88,68\]. Platform independence in turn implies the ability to quickly react to changing technological backgrounds \[88\], improved portability \[26,71,70\] and support for interoperability \[68\], pp. 1–2.

Another staple of MDD are code generators of various types, which convert business models into platform-specific code \[47,67,68\]. The use of code generation can result in high reusability \[65,88\] and therefore increased productivity \[24,47,97\] as well as rapid development \[47,15\] by deferring many of the less intellectually challenging aspects of programming to the code generator \[15\]. For similar reasons, the use of generated code is also expected to lead to improved software quality \[24,47,65,15,97\].

However, the intended robustness with regard to change extends over the whole lifecycle of an application, as the nature of MDD approaches implies keeping the models up-to-date at any point. Such a model not only greatly aids the understanding of the system but can also be used to quickly implement business level updates, thus providing the potential for improved maintainability \[24,65\]. Maintenance, which is not limited to error correction but rather focuses on the addition of new business level features, is an important but often neglected factor in Software Engineering (cf. Glass \[41\]). Support for alterations to the software at any time of the lifecycle is a major advantage of using model-centric approaches. Also, the in-synch model could be useful for other neglected tasks such as project retrospectives to avoid what Glass calls “making the same mistakes on project after project” \[43\]. For Haywood the proper emphasis on modeling and design seems to be the most important benefit of model-centricity, in fact he doubts that the resistance to change will be as great as advocated by others \[47\].

As has already been mentioned, many researchers, such as Mellor and Balcer, see a higher level of abstraction as the main goal of MDD. Mellor and Balcer hope to achieve this level by using an executable variant of UML \[67\], pg. 11. The increase in abstraction is an advantage also cited by Booch et al. \[15\], Kühne \[65\], Selic \[89\], and Muller et al. \[69\]. Cook and Booch et al. see the introduction of domain-specific languages as opposed to technology-oriented conglomerates of different programming languages as the main reason for this new level of abstraction \[22,15\].

But what does MDD distinguish from its predecessors such as CASE or visual programming languages? First of all, MDD is based on standards such as XMI, MOF and UML, reducing the effort required for infrastructural developments \[88\]. Without such standards, there is a risk when creating a model or visual program, because a proprietary solution might not be supported in the long run \([47\) pg. 118],16, and \[88\]). Furthermore, MDD is deliberately kept free from ties to a specific design methodology \[45, pg. 118\]. Like a compiler, MDD should be usable in many different contexts instead of constraining its users to a specific, possibly sub-optimal design method. Without these two factors, MDD would be little more than a renaissance of past concepts.

After this brief introduction to MDD, the potential of MDD for the Web Application domain can be identified. We start with a short analysis of how frameworks and Web design languages can be incorporated into an MDD approach, for reasons that will become apparent during these descriptions. After that, the potential for negating the drawbacks of the conventional techniques is discussed, leading to insights with regard to the circumstances under which it makes sense to use MDD for Web Applications. The section ends with a brief discussion of existing Web-MDD approaches, which can be examined in the light of the previous analysis.

4.2. Using frameworks and Web design languages as a basis for MDD

Since any domain-specific language and therefore particularly a Web design language can be considered to be at least an implicit, if not explicit, metamodel (cf. \[2, Chapter 11\]), the benefits of using design languages are in principle available in the context of MDD. Indeed, many of the Web design languages such as HDM \[34\] were incorporated into CASE tools back when CASE was the state of the art for automated code generation. Also, there is at least one current “MDD” approach using Web design languages as a basis; Fraternali and Paolini employ a dialect of HDM in their model-driven Autoweb system \[31\]. However, it has to be stressed again that MDD requires the use of general standards and thus an approach that builds on a proprietary metamodel cannot be considered MDD. Yet, it should be possible to create UML profiles which correspond to Web design languages and thus incorporate their advantages into Web MDD tools.

Similarly, frameworks are a natural match for MDD (cf. Völter \[97\]) as they can help to keep the code generation templates simple. A good framework will provide options for different situations and offer a simple way of switching
between them. In the simplest case an MDD tool, to use the term in its most general meaning, will provide a graphical front end for the editing of configuration files such as the one for Struts described by Wang [98]. Ideally, however, the MDD software will make intelligent decisions which are far more abstract, as will be described in the next section.

4.3. Possible improvements due to MDD

The idea of using code generation for Web Applications is not new. Theng, for example, feels that “if some design ideas could be automated so designers need not worry about their implementation, chances are that better applications could be produced, because designers would be freed to concentrate on their critical issues that cannot be automated” ([95], emphasis added). More specifically, it is not only “glue code” which should be generated but rather code based on rational decisions made by the template designers.

As frameworks need to be more general in nature to justify the effort put into them, it seems to be desirable to raise the level of abstraction even higher in MDD tools, thereby giving up even more degrees of freedom. Thus, many of the usability problems can be addressed automatically. For example the navigation scheme of a Web Application can be provided automatically based on general guidelines given by the modeler. Of course, a solution at this level of abstraction can only be applied to a small subset of existing problems and therefore it should be easy and fast to come up with alternatives or extensions for the code generation templates. In this context a hierarchical approach to MDD as mentioned above might be helpful due to its specific properties.

The constant need for maintenance has been identified as one of the key technical problems of Web Applications. While frameworks were identified as helpful for these tasks, MDD offers further improvement. As discussed in Section 4.1, easier maintenance due to a focus on the model as a central artifact of the software development process is one of the major strengths of MDD. Clearly, in the Web environment, this potential can be realized to its fullest.

Heterogeneity of the client-side platforms is a major challenge for the development of universally applicable Web sites. While general platform independence is often deemed challenging or even impossible (see for example Haywood [47]), it is still possible to provide various code generation templates for a small and focused area such as the user interface.

The pronounced need for testing resulting from heterogeneity is supported by the fact that the code generation templates can be tested separately and can benefit from the feedback of all projects using them. Additionally, test cases or simulation runs can be generated automatically.

Whether MDD can help with regard to aspects which are neither technical nor usability-related is debatable. One might argue that the legal issues such as accountability could be supported because of the up-to-date documentation provided by the models or that the Web site’s credibility will be improved due to a higher quality, but at this stage such claims remain speculation.

4.4. Existing approaches

It seems that in the last few years, the Web was (re-)discovered as a domain which easily lends itself to code generation. For this reason, only a subset of the existing examples can be presented in this context. One category of approaches is that by researchers who come from an MDD background and want to illustrate its applicability to the Web domain. Others come from the field of hypertext design and use the newly aroused interest in Web Application generation to introduce their code generation approaches to a different audience. We will start off by discussing the latter group first.

The AutoWeb System proposed by Fraternali and Paolini allows the automatic generation of data-intensive Web sites. Based on a variant of the Web design language HDM (called HDM-lite), database entries defining the navigational and presentational metadata are generated along with specialized tables for the data described by the conceptual model. This information is used by a runtime environment to generate Web pages dynamically. The appearance of the Web site is defined via stylesheets, which are based on presentation elements. These elements can be either black boxes (i.e. code fragments provided by the designer) or built-in and configured via parameters. Legacy systems can easily be integrated using data replication tools [31]. While the approach of Fraternali and Paolini seems to be limited to content management systems, it is interesting because it generates code based on one of the established Web design languages, thus lending support to our reasoning in Section 4.2. All in all, AutoWeb’s strength seems to lie in the strong Web background; however, it can only be called model-driven in the sense that it
generates code from models. It is neither based on the standards that represent the progress of MDD over previous code generation approaches nor is it independent of a specific design methodology. In fact, the abstract of [31] even states that AutoWeb is a methodology. Similar arguments can be brought forth regarding related approaches such as those based on WebML (cf. [99]).

_Hera_ is a model-driven approach to Web information systems (WIS). The paper by Houben et al. stresses the information system nature of many modern Web Applications, either because of their connection to legacy systems or their data-intensive nature. Hera introduces several views on a WIS architecture. The semantic layer consists of a conceptual model as well as an integration process which takes data from different sources into the model. The application layer describes the application in terms of navigation and user adaptation. The presentation layer specifies the details needed to produce a presentation suitable for a specific platform. The models undergo several transformation steps leading to a platform-dependent model (similar to the MDA’s PSM) which is executable within a special Web-based interpreter. Functionality, i.e. business logic, is incorporated via callbacks [52]. However, yet again, the approach is both tied to a specific methodology and non-standard modeling languages.

**VisualWADE**, proposed by Gómez, is another code generation tool for Web Application development. It is based on a Web design method called OO-H. The tool offers the three different design dimensions structure, navigation and presentation, each of which consists of a UML diagram, possibly with stereotypes and statements in the OCL-derived action language provided by VisualWADE. Pages are stored abstractly as XML [44]. Again, this approach cannot be called MDD in the strictest sense, because it is tied to a specific methodology, i.e., OO-H.

Constantine and Lockwood contribute an exotic example in the form of a so-called agile usage-centered model-driven approach for Web Applications. It is based on a model of the various user roles, abstract, simplified, and technology-free use cases, and abstract prototypes called content models. These are combined with secondary models of the domain, the business rules, and the working environment. An abstract prototype describes the user interface’s contents without actually modeling the appearance and behavior and it also organizes the content into interaction contexts. The focus is on usability and interface design. Also important are the navigational design and the visual and interaction scheme, which describes recurring visual elements, common layouts and templates to apply to the interaction contexts [21]. Their approach is unusual in the field of model-driven development because it does not use any code generation. While the paper offers many interesting insights on usability, it is also a good indicator on how the term _model-driven_ is used as a buzzword to attract attention.

However, there are also MDD approaches that are more strongly oriented towards the goals and principles of MDD. Heckel and Lohmann propose a model-driven approach for Web Applications based on what they call _graphical reaction rules_. They criticize the lack of dynamic modeling in most MDD approaches and propose the use of statecharts and collaboration diagrams to describe the dynamic aspects in a fashion similar to that used by Executable UML (cf. [67]). The changes incurred by the state transitions are described by model transformations. One of the main problems addressed by this approach is the consistency between the PIM and the platform specific code. By generating a platform independent statechart handler class which encapsulates the business logic, mappings to other technologies are simple and only require changes in few classes [48]. The approach has a certain appeal, especially as it shows the benefit of using a framework — in this case in the simple form of the statechart handler class. However, the use of model transformations as opposed to code in an action language seems to imply too much overhead. Also, this approach does not explicitly address Web-specific problems and in fact could be used for any kind of software domain with little adaptation.

Another approach using an executable PSM is _Netsilon_ by Muller et al. With avoiding a change to the working habits of the graphic designers and HTML integrators being a major goal, besides a raised level of abstraction and increased productivity, care is taken to allow the inclusion of static HTML fragments. The core models of this approach belong to three different views (called aspects in the paper). These are a business aspect, which uses an action language to describe the logic, a hypertext aspect for composition and navigation, and a presentation aspect [69]. Considering the background of some of the authors, Netsilon can be seen as an example of Web MDD as a specialization of general MDD. Therefore the model-driven aspects such as metamodel and transformations are stressed and less attention is given to the Web aspects. For example, navigation options seem more primitive than those provided by OOHDM. Similar arguments can be brought forth regarding the agile model-driven approach by El Kaim et al. based on Netsilon ([29], also see [1] for further information on agile modeling).

The UWE methodology is based on UML models, which can serve as a basis for the semi-automatic generation of Web Applications. UML is used throughout, for example navigation models are class diagrams, using stereotypes such
as “index” or “guided tour” for access structures. Dynamic behavior is represented primarily by statechart diagrams [63]. The strong points of this approach are its adherence to Web design principles as well as MDD standards. Despite its focus on a methodology, it seems one of the more promising Web MDD approaches, from a purely academic standpoint.

Other examples for Web MDD approaches include WebSA, proposed by Beigbeder and Castro, which uses a UML profile for modeling and involves an additional PIM-to-PIM transformation [13], or the PIM for Web-based systems proposed by Smith and Shrimpton as a basis for an analysis of model transformations [91].

4.5. Evaluation of the existing approaches

As can be seen, the current approaches are either very well founded on Web Engineering or MDD expertise but typically not on both. This is not to say that these approaches (specifically those by experts of the Web domain) are not workable or will produce bad Web sites. Rather, we criticize that the term MDD is often used without true understanding of its implications, using it to “re-label” CASE tools. It cannot be stressed enough that MDD aims at avoiding the problems of the CASE movement. It does so by avoiding a close connection to specific methodologies and by relying on generally accepted standards to ensure model compatibility. Thus, potential users of MDD need not worry about being tied to specific solutions, creating “legacy models” they will be unable to use in the future should they change their methodology, their modeling tool, or their code generator. Only when these principles are embraced will the full potential of MDD be realized in the Web domain.

5. Conclusions

The goal of this paper was to provide well-founded arguments that justify the effort of creating Model-Driven Development solutions for the domain of Web Engineering, a problem which has so far been neglected in the literature. For this purpose, we have asked several questions:

- What are the problems of Web Engineering and how are they traditionally addressed?
- Which of these problems can be addressed by MDD approaches?
- Are MDD approaches better than the traditional ones and therefore fit to replace them?
- If yes, do the currently available MDD approaches live up to this potential?

To answer these questions, the problems unique to the domain of Web Applications have been analyzed, yielding two main groups of problems, i.e. usability and technical issues. It has been shown that conventional approaches such as Web design languages and Web Application Frameworks already contribute to the solution of some aspects but that MDD is not only able to integrate these classical solutions and gain their benefits but also contributes other advantages such as easier maintenance, which is a critical issue in Web Applications. In particular, the advantages offered by Web design languages are better realized in a model-driven context as the models created with them form the central artifacts of the design process and thus do not require redundant work. We have also shown that most current approaches either ignore important Web Engineering knowledge or are actually CASE tools, relabeled as MDD. This means that the full potential of MDD regarding Web Engineering is not yet realized and we hope that this article will encourage others to contribute their ideas to this exciting field of research.

References


