J. Software Engineering & Applications, 2010, 3: 419-425
doi:10.4236/jsea.2010.35047 Published Online May 2010 (http://www.SciRP.org/journal/jsea)

Scientific Research

A Reference Model for the Analysis and Comparison of MDE Approaches for Web-Application Development

João de Sousa Saraiva, Alberto Rodrigues da Silva

INESC-ID/Instituto Superior Técnico, Lisboa, Portugal.
Email: joao.saraiva@inesc-id.pt, alberto.silva@acm.org

Received January 26th, 2010; revised March 18th, 2010; accepted March 20th, 2010.

ABSTRACT

The emerging Model-Driven Engineering (*MDE*) *paradigm advocates the use of models as first-class citizens in the software development process*, *while artifacts such as documentation and source-code can be quickly produced from those models by using automated transformations. Even though many MDE-oriented approaches*, *languages and tools have been developed in the recent past*, *there is no standard that concretely defines a specific sequence of steps to obtain a functional software system from a model. Thus*, *the existing approaches present numerous differences among themselves*, *because each one handles the problems inherent to software development in its own way. This paper presents and discusses a reference model for the comparative study of current MDE approaches in the scope of web-application development. This reference model focuses on relevant aspects such as modeling language scope* (*domain, business-logic*, *user-interface*), *usage of patterns, separation of concerns*, *model transformations, tool support*, *and deployment details like web-platform independence and traditional programming required. The ultimate goal of this paper is to determine the aspects that will be of greater importance in future web-oriented MDE languages.*

Keywords: *Model-Driven Engineering*, *Web Engineering*, *Software Development*

1. Introduction

The worldwide expansion of the Internet in the last years has made it a powerful platform for the deployment of a variety of artifacts and systems. This has led to the appearance of a myriad of frameworks and libraries that attempt to harness the power of Internet-based technologies in order to accomplish various objectives. Typical examples of widely-used web-application frameworks include Microsoft's ASP.NET [1], Sun's Java EE [2], PHP (PHP: Hypertext Preprocessor) [3], Ruby on Rails [4,5], Django [6], or Catalyst [7].

This paper proposes a reference model for the analysis and comparison of MDE (Model-Driven Engineering) approaches to web-application development. This reference model is focused on particularly relevant aspects such as the scope of modeling language scope (*i.e.*, if it addresses modeling of domain, business-logic, and user-interface), separation of concerns, time-saving features such as usage of patterns and/or model-to-model transformations, tool support available, and deployment aspects like web-platform independence and whether the approach still requires traditional programming (*i.e.*, development of source-code). The goal of this reference model is to identify concepts, principles and best practices that are likely to become important in future model-driven web-application languages.

It should be noted that, although currently there are not many MDE-oriented approaches to web-site or web-application development, there are some approaches – the analysis of which was not included in this paper due to length constraints – that we believe should be mentioned. Of those, we highlight WebML (Web Modeling Language) [8,9], UWE (UML-based Web Engineering) [10], XIS2 (eXtreme Modeling Interactive Systems) [11,12], the OutSystems Agile Platform [13], OOHDM (Object-Oriented Hypermedia Design Model) [14], or OPM/Web (Object Process Methodology for Developing Web-Applications) [15]. We also point out that there are other studies and tools regarding MDE-oriented approaches and languages, not directly related to web-application development, but rather to other important aspects such as 1) the usage of meta-metamodels [16], 2) user-interface modeling [17-21], or 3) the usage of prototyping tools to provide a way for stakeholders to draw and communicate

design ideas, as is the case with Microsoft Sketchflow [22].

This paper is organized as follows. Section 1 introduces the context of web-applications and frameworks. Section 2 describes the reference model that we defined for the analysis and comparison of MDE-oriented web-application development approaches. Section 3 provides a discussion regarding this reference model; this discussion also features some examples from web-application modeling approaches that we have analyzed in our research, and to which we have applied this reference model. Finally, Section 4 concludes this paper.

2. Reference Model

This reference model is defined according to a set of aspects that are relevant to MDE-based development, namely(see **Table 1**): 1) the language used; 2) the usage of model-to-model transformations; and 3) the tool support for the design, generation, and deployment of the web-application.

2.1 Modeling Language

The language used by the approach is analyzed in terms of the meta-metamodel used (if any), and whether it addresses a set of modeling concerns, such as domain modeling, business-logic modeling, and user-interface modeling.

Meta-metamodel. Besides the web-application modeling language, it is important to identify the language's meta-metamodel (if any) and the modeling elements that it provides. We consider that this aspect is important because: 1) this meta-metamodel's expressiveness can affect the number of elements that the language itself can provide, which in turn can have a direct impact on the language's own expressiveness and its adequacy to solve complex web-application problems; and 2) the meta-metamodel used by the language can have a profound influence on the tool support that is – or may become – available to the approach, e.g., languages specified as a UML 2.0 profile [23] are likely to be supported by the majority of UML modeling tools that support the Profiling mechanism.

Domain Modeling. Domain modeling concerns the identification of problem-domain concepts, and their representation using a modeling language (e.g., UML diagrams). This aspect is analyzed regarding: 1) whether it is independent of persistence or user-interface details (*i.e.*, domain models do not need to be "adjusted" to support those layers); and 2) the elements provided by the language, such as enumerations (which are useful to avoid specifying multiple concepts with no particular differences between themselves) or associations (namely their arity, *i.e.*, the number of possible associated entities).

Business-Logic Modeling. Although the definition of business-logic modeling can be considered somewhat subjective, in this work we consider it as the specification of the web-application's behavior. This aspect is analyzed regarding the following subjects: 1) whether it supports querying and manipulating domain concepts in a either textual or graphical manner, e.g., using SQL-like statements; 2) whether this querying and manipulation support is "low-level", in a manner similar to traditional source-code; 3) support for process specification; and 4) usage of patterns, such as the typical "create-read-update-delete" (CRUD) set of business-logic patterns. It should be noted that the "low-level support" subject is considered relevant because it often reflects the expressiveness of the language: typical source-code-oriented languages (such as C or C#), although complex, are nevertheless very expressive.

Navigation Flow Modeling. The approach's support for specifying the navigation flow (in the context of the modeled web-application) between different HTML pages, or even inside HTML pages, is also analyzed in this issue.

User-Interface Modeling. Another important aspect is the approach's support for specifying/modeling the user-interface (UI). The subjects analyzed are the following: 1) whether the UI modeling language is platform-independent (*i.e.*, does not require specific software to present the UI); 2) supports access-control specification (*i.e.*, certain controls are shown/hidden according to the authenticated user); 3) allows the definition of custom UI elements; 4) allows the usage of interaction patterns (e.g., create, edit, associate/dissociate); and 5) supports binding between UI elements and domain model elements.

2.2 Model-to-Model Transformations

This subsection examines whether the approach uses (or even considers) the usage of model-to-model transformations. This kind of transformations is typically used to accelerate the task of designing the web-application, by using model analysis and inference mechanisms to automatically specify some parts of the web-application model, thus releasing the model designer from some repetitive and error-prone tasks.

2.3 Tool Support and Deployment

This subsection analyzes the tool support that is available for the approach, regarding the following aspects.

Modeling Tool. This aspect determines whether the approach (and its language) is supported by a design tool, and if that design tool is proprietary (*i.e.*, if it is built specifically for supporting the approach), or if it is a generic tool, adapted to specific details of the language.

Need for Traditional Programming. This aspect analyzes if the tools supporting the approach allow the generation of (parts of) the web-application, and whether the generated application is complete (*i.e.*, it does not require the manual implementation of specific features by programmers).

Deployment Environment. Finally, this aspect determines the target platform(s) considered by the approach, and whether those target platforms are proprietary (in other words, if there is supposed to be a tight coupling between the approach and the target platform).

3. Discussion

This section provides a discussion regarding the criteria described in Section 2. Some of the arguments that we present in this discussion are based on our own application of this reference model to some MDE-oriented web-application development approaches (such as WebML and UWE), while others are based on our own previous experiences in this field of study (such as XIS2).

3.1 Modeling Language

It is frequent for this kind of development approaches to define a graphical modeling language, sometimes with textual elements involved (e.g., OCL constraints in UWE). Nevertheless, the language's users can certainly benefit from having textual specifications assisted by graphically-oriented features, such as using drag-and-drop to place such textual elements.

Regarding syntax, UML-based approaches and languages tend to use UML's graphical syntax (e.g., boxes for representing classes or enumerations, a "stickman" figure for representing a stakeholder), mainly due to their nature as UML-extending metamodels and as UML profiles. On the other hand, other languages typically define their own syntax, albeit some parts of those syntaxes are usually based on existing modeling languages, with the objective of facilitating their learning process.

Finally, regarding semantics, the difference between UML-based and non-UML-based languages is also quickly noticeable. The semantics of most UML-based languages are defined verbally – in their respective language specification documents – with some OCL restrictions defined; validation of models specified using these languages is dependent on the expressiveness of OCL and the modeling tool's support for OCL restrictions. On the other hand, the semantics of other languages, although not specified in a "formal" manner, are supported by custom-built tools that can easily address all aspects of the language's semantics, because of those tools' use of low-level programming languages. Nevertheless, we consider that this difference is present mainly because of the compromise, that each approach must assume, between the "tool support" and "model exchangeability" factors: 1) for "real-world" languages (with a high degree of complexity), it is usually easier to provide *adequate* modeling and validation facilities via a custom tool with a low-level programming language, than by a UML generic modeling tool with a "UML Profile + OCL restrictions" customization mechanism; 2) however, due to their typically academic nature, UML-based approaches are also concerned with exchanging models between tools, a concern not shared by commercially-oriented approaches, which accounts for the choice to use UML in such approaches.

Meta-metamodel. We have noticed that commercial-oriented approaches usually define their languages without using a standard meta-metamodel, while academic approaches tend to use existing meta-metamodels (e.g., UML [23]) as the basis for their web-application modeling language.

This is possibly due to the common belief that the meta-metamodels available today, such as UML or MOF, are too complex and attempt to address too many problems (which is one of the major factors that drive the discipline of Domain-Specific Modeling [24]). Considering that commercial approaches are supposed to be practical and "easy to use", the creators of such approaches are likely to opt for the creation of a language from scratch (even if this involves some initial extra effort), in order to avoid dealing with issues due to the inherent complexity of existing meta-metamodel languages.

On the other hand, academic approaches are typically more focused on obtaining and divulging results that contribute to advance the state-of-the-art, while aspects such as simplicity and "ease of use" are usually considered secondary. Thus, in this kind of environment, using a standard meta-metamodel is not only recommended, but it is also a way to ensure that other researchers can quickly build upon those results and advance the state-of-the-art even further.

Domain Modeling. Domain modeling is a subject that *must* be addressed by any application development approach. This is because an application, with the purpose of solving a certain problem, will undoubtedly manipulate a set of entities, directly or indirectly related to that problem; these entities, in turn, are described by a domain model.

Most software development approaches and best practices advocate the independence between the domain model and other parts of the application functionality (e.g., separating the domain model from persistence or UI layer details). However, most languages usually end up requiring that the domain model be "adjusted" to UI- or persistence-oriented details (e.g., in the Address Book example at the UWE tutorial [25], the "Address Book" class has an attribute "introduction", in order for the main screen of the application to show a small introductory message to the user). Although from a theoretical perspective this can be regarded as a bad thing, in practice it does have the advantage of endowing the designer with a greater level of control over the web-application implementation itself, instead of just relying on possibly fallible automated mechanisms. Nevertheless, this advantage can also be obtained by the technique of adding modeling elements, outside of the domain model, to establish the mappings between the domain model elements and the elements of other models.

Also, most languages usually provide some degree of support for important concepts like enumerations and associations. Associations are obviously important, as they enable the definition of relationships between entities. On the other hand, enumerations are also important because they enable the a-priori definition of "instances" while avoiding a potential explosion of entity types. UML-based approaches can inherit support for these concepts automatically; other languages typically provide support for associations, but enumerations are seldom supported (e.g., WebML does not support enumerations, but the OutSystems Agile Platform does).

Business-Logic Modeling. Business-logic modeling is usually addressed by this kind of modeling approaches, although how this is handled varies greatly according to the approach. Modeling features typically found are the usage of pre-defined business-logic patterns, the possibility for designers to define their own business-logic patterns, and the deference of business-logic specification to traditional, source-code-oriented, development. However, most approaches frequently adopt a combination of these possibilities: XIS2 provides the typical create-read-update-delete operations automatically, and the designer can add more operations; on the other hand, the OutSystems Agile Platform only supports its pre-defined operations, although the wide variety of operations that it supports should be adequate for most of the business-logic that the designer would want to specify.

A relevant issue that should be mentioned here is the trade-off between *abstraction* and *expressive power*. Although languages such as XIS2 or WebML try to raise the abstraction level at which designers have to think and specify their models (as opposed to just developing the web-application in a manual source-code-oriented fashion), the fact is that this higher abstraction usually also affects the expressive power available to designers in a negative manner. On the other hand, approaches that are not particularly oriented towards raising the abstraction level (e.g., business-logic modeling in the OutSystems Agile Platform is intended to provide *developers*, who have experience in dealing with web-application-related concepts such as "request parameters", "session values" or "asynchronous requests", with a graphical environment for them to do their tasks) suffer almost no impact on the expressive power available to business-logic modeling.

Navigation Flow Modeling. Navigation flow modeling is an aspect that is fundamental to any kind of web-application, and so these development approaches must address it. Due to their "web-application modeling" nature and all that this implies (e.g., the existence of HTML pages, hyperlinks to navigate between pages, using request parameters or a similar mechanism to provide necessary parameters), this kind of development approach typically all follow the same guidelines: a directed graph is drawn, where nodes correspond to HTML pages, and edges correspond to the possible navigation flows that are available to the user within the context of a certain page.

The difference between most approaches, however, lies in whether the designer can specify the sets of edges (*i.e.*, actions or links) available in each node. An example of this difference can be found in XIS2 and WebML: in XIS2 the designer can specify actions, associated to UI elements in a page, and the navigation flows will afterward be associated with the corresponding page's actions, while in WebML each Data-Unit node has a well-defined set of links, for input and output, and so the designer cannot specify additional links.

User-Interface Modeling. Most web-application development approaches address UI modeling in a graphical manner, using a WYSIWYG (What You See Is What You Get) approach. However, there are many variations on what can be modeled in the UI. For example, the WebML language, as described in [26], only allows the specification of *what* elements will be present on the page, but not *where* they will be located (although its commercial tool does provide support for this kind of UI modeling). On the other hand, other approaches (such as the OutSystems Agile Platform or XIS2) do allow the specification of the location of such elements.

An issue that should be mentioned, related to WYSIWYG-like UI modeling, is the possibility of using and or capturing UI patterns to address the UI's behavior. From our experiences in using this reference model, we have noticed that most approaches (such as XIS2, WebML and the OutSystems Agile Platform) usually address this behavioral aspect through the capture of UI patterns (e.g., associate/dissociate, list, create item), enabling applications to *interact* with users, instead of just displaying requested resources.

An aspect where most approaches actually differ between themselves is the possibility of reusing page or interface components. As an example, WebML and OutSystems' Platform support this feature, while XIS2 or UWE do not. Nevertheless, we consider this to be a very important feature, as it allows designers to specify certain screen sections only once, and "import" those sections into some/all of the application's screens, with the obvious added advantage that changes to such a section need to be done only in a single point in the model; a good example of such a component would be a web-site banner, or a web-site contents' navigation tree.

Regarding platform-independence, we believe that there is not much to be said, considering that these approaches have to generate regular HTML to be sent to a web-browser, and so can be considered as target-platform-independent.

Finally, it is very frequent for these approaches to allow binding UI elements to domain elements (e.g., configure the rows of a table to show the values of field in specific instances). However, these approaches usually differ on

whether they allow the customization of those bindings in the model (e.g., change a cell in a table row to show a different value for each instance). Although not being able to customize these bindings can certainly simplify the modeling task and avoid designer errors, in practice this is also a limitation on the expressiveness of the language, which can force developers to change generated source-code in order to address specific requirements.

3.2 Model-to-Model Transformations

From our own experience regarding both academic and non-academic web-application tools and approaches, support for model-to-model transformations is an aspect seldom addressed by these approaches, and typically found only on the UML-based ones. This is possibly because model-to-model transformations are typically implemented in such a way that user-defined information on the destination model may be lost. Additionally, most approaches use a single modeling language, and so the information that would be present in the destination "model" can be inferred from the information that has already been modeled.

The approaches that do consider the usage of model-to-model transformations typically use them to accelerate modeling tasks, by taking the information already modeled at the time the transformation is performed and generating new views. These transformations reflect how those views would *typically* be modeled, and the designer is free to change the generated views to suit the application's needs (e.g., showing an extra field in the UI, or adding an extra step in an activity diagram).

3.3 Tool Support and Deployment

For any web-application modeling approach to actually be usable, it has to provide some kind of tool support.

Modeling Tool. Tool support is an aspect typically influenced by the meta-metamodel used (if any), and by the academic/commercial orientation of the approach. Academic approaches based on UML are usually tool-agnostic and are supposed to be usable in any UML modeling tool that supports the Profiling mechanism. On the other hand, commercial approaches provide their own proprietary (and language-specific) modeling tools; nevertheless, it is sometimes possible to specify UML profiles that enable the modeling of such languages in regular UML modeling tools (albeit in a somewhat limited fashion). An example of this can be found in [27], which defines a WebML-oriented UML profile.

It should be noted, however, that such language-specific tools have a great advantage over "generic" tools, as they can assist the user throughout the entire development life-cycle (by means of mechanisms such as contextual help or helpful validation tips), in a manner that is well adjusted to the approach.

Need for Traditional Programming. Although it would certainly be desirable that a modeling approach required no programming efforts (for reasons such as avoiding programming errors, or reduced software development costs), the fact is that most web-application modeling approaches typically consider typical programming tasks (*i.e.*, manual editing of generated source-code artifacts) as an activity to occur during the development life-cycle, in order to account for particular requirements that may not be expressible by the approach's modeling language. Of the approaches that are known to us, only the OutSystems Agile Platform considers that source-code should not be edited in a manual fashion: designers/developers can *only* edit the model itself, and generated code and databases are always kept "behind-the-scenes".

This is an aspect that clearly illustrates one of the consequences of the traditional MDE question "how expressive should the modeling language be?" This question is actually a dilemma because increasing the expressiveness of a language typically involves adding elements to it, in order to address more semantic possibilities, which in turn can increase the difficulty for a new designer to learn the language, as well as possibly reducing its level of abstraction. On the other hand, if the language does not provide many elements, it is easier to learn, but it may also be unable to specify some desirable features. So, current modeling languages are actually the result of a "balanced compromise" between these factors, always taking into account the purpose of the language.

Deployment Environment. Most web-application modeling approaches are actually independent of the deployment environment, because they do not target a specific technology or platform (e.g., MySQL database, Apache web-server); nevertheless, although the concepts in the languages of some approaches may pose some technological restrictions, developers are sometimes able to use "workarounds" to remove such restrictions (e.g., replacing a Java servlet with an adequate ASP.NET class).

As an example of this kind independence, we can point out WebML, UWE and XIS2: because of their usage of high-level elements, they can generate code for any web-oriented platform (e.g., ASP.NET, Apache Tomcat); in fact, XIS2 can also generate code for desktop-based platforms. However, models that are created in the OutSystems Agile Platform can only be deployed to the OutSystems deployment stacks, which use either 1) the JBoss application server and Oracle database, or 2) Microsoft's IIS application server and SQL Server Express database.

Nevertheless, we consider it important to note that, although in the deployment aspect the Agile Platform is apparently much more limited than other approaches, this "disadvantage" is actually what allows the Agile Platform to automate most of the web-application development life-cycle, namely deployment and application upgrade/rollback scenarios, something that is clearly lacking in

most of the web-application modeling approaches that we know.

4. Conclusions

Most web-applications are still created in a traditional manner, by manually developing source-code, which is a slow and error-prone task. This is a problem, not specific to just web-applications but to the software engineering discipline in general, that MDE aims to address. Furthermore, the web-based development paradigm introduces a set of Internet and HTTP-related concepts that have to be addressed by the designer (e.g., page, navigation, request, GET, POST).

In this paper, we proposed a reference model for comparing current MDE-driven approaches for web-application development, according to relevant criteria such as support for domain, business-logic, UI and navigation-flow modeling, as well as usage of model-to-model transformations and tool support. **Table 1** summarizes the key issues of the proposed reference model. Finally, this paper discussed the defined reference model.

Some reflections can be made regarding the criteria identified in this reference model. The first aspect concerns the *compromise between language expressiveness, simplicity*, *and learning curve* for new designers. Although the *purpose* of the language must always be a factor to weigh in on this compromise, the fact is that very expressive languages can be harder to learn because of their great number of elements than languages with few elements, which are typically simpler. Moreover, if a creator of a language starts increasing its expressiveness by adding new elements, it is likely that its level of abstraction over another language is also diminished, because those new elements must reflect particular semantic possibilities of the lower-level language. Note that we say "very expressive languages" instead of "languages with many elements". This is on purpose, as a language can provide many elements, and still not be very expressive; in these cases, some of those elements have exactly the same semantic meaning but different syntax, what we typically call *syntactic sugar*. Java and C# e.g., have about the same level of expressiveness, although C# provides a greater number of elements (e.g., delegates).

Another aspect concerns the *compromise between tool support*, *and integration with the deployment platform*. Although, in theory, using a generic tool to create models and generate the desired source-code is a good idea, in practice it actually increases the difficulty of obtaining a *working system* from the model. A good example of this can be found in the OutSystems Agile Platform vs. a generic UML-based modeling approach such as XIS2: the latter requires the generation, compilation and deployment of artifacts (clearly a task to be performed by a programmer or a technical user), while the former handles all these deployment aspects in a manner completely transparent to the designer.

Table 1. Main issues of the proposed reference model

Analysis Aspects			Examples of Possible Analysis Options
Modeling Language	**Meta-metamodel**		UML 2.0 (Profile) None
	Modeling	**Domain Modeling**	Independence of database Independence of UI Support for enumerations and associations
		Business-Logic Modeling	Support for domain model query and manipulation Usage of business-logic patterns
		Navigation Flow Modeling	Designer-defined navigation links between pages
		User-Interface Modeling	Custom interface elements Designer-defined UI patterns
Model-to-Model Transformations	**Modeling levels concerned**		PIM-to-PIM PIM-to-PSM
	Transformation languages		Programming-language specific Independent of programming-language (e.g., QVT)
Tool Support and Deployment	**Modeling Tool**		None Integrated into (or an extension for) a generic CASE/IDE tool Language-specific tool
	Need for Traditional Programming		None Yes
	Deployment Environment		Platform-independent Targets specific platform
			Platform-independent Targets specific platform

Finally, although there is work regarding MDE approaches for the development of web-applications, it does not address platforms of a more specialized nature, such as CMS (Content Management System) or DMS (Document Management System) platforms. Although this is understandable, as these platforms provide additional concepts and supporting them would introduce even more problems (e.g., what to do when a language element can not be mapped to an implementation concept), we consider that it also constitutes additional motivation to *use a set of languages and the corresponding model-to-model transformations*: each language could address a specific kind of platform, and the model-to-model transformations would be responsible for mapping concepts between those languages/platforms.

REFERENCES

[1] "The Official Microsoft ASP.NET Site". http://www.asp.net

[2] "Java EE". http://java.sun.com/javaee

[3] "PHP: Hypertext Preprocessor". http://www.php.net

[4] "Ruby Programming Language". http://www.ruby-lang.org

[5] "Ruby on Rails". http:// rubyonrails.org

[6] "Django–The Web framework for perfectionists with deadlines." http://www. djangoproject.com

[7] "Catalyst–Web Framework". http://www.catalystframework.org

[8] http:// www.webml.org

[9] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai and M. Matera, "Designing Data-Intensive Web Applications," Morgan Kaufmann, 2003.

[10] "UWE–UML–Based Web Engineering". http://uwe.pst.ifi.lmu.de

[11] A. R. Silva, J. S. Saraiva, R. Silva and C. Martins, "XIS-UML Profile for eXtreme Modeling Interactive Systems," 4*th International Workshop on Model-based Methodologies for Pervasive and Embedded Software*, IEEE Computer Society, Los Alamitos, March 2007, pp. 55-66.

[12] A. R. Silva, J. Saraiva, D. Ferreira, R. Silva and C. Videira, "Integration of RE and MDE Paradigms: The ProjectIT Approach and Tools," "On the Interplay of .NET and Contemporary Development Techniques," *IET Software Journal*, Vol. 1, No. 6, December 2007, pp. 294-314.

[13] "Agile Software Development and Management," OutSystems. http://www.outsystems.com/agile

[14] "Object-Oriented Hypermedia Design Model". http://www.telemidia.puc-rio.br/oohdm/oohdm.html

[15] I. B. Reinhartz and D. Dori, S. Katz, "OPM/Web–Object–Process Methodology for Developing Web Applications," *Annals of Software Engineering*, Vol. 13, No. 1-4, 2002, pp. 141-161.

[16] J. S. Saraiva and A. R. Silva, "Evaluation of MDE Tools from a Metamodeling Perspective," *Journal of Database Management*, Vol. 19, No. 4, October-December 2008, pp. 21-46.

[17] W. Kozaczynski and J. Thario, "Transforming User Experience Models to Presentation Layer Implementations," *Proceedings of the Second Workshop on Domain-Specific Visual Languages*, Seattle, November 2002.

[18] P. P. Silva and N. W. Paton, "User Interface Modeling in UMLi," *Institute of Electrical and Electronic Engineers Software*, IEEE, Vol. 20, No. 4, July 2003, pp. 62-69.

[19] J. Van den Bergh and K. Coninx, "Towards Modeling Context-sensitive Interactive Applications: the Context-Sensitive User Interface Profile (CUP)," *SoftVis'*05: *Proceedings of the* 2005 *ACM Symposium on Software Visualization*, ACM, New York, 2005, pp. 87-94.

[20] P. Azevedo, R. Merrick and D. Roberts, "OVID to AUIML-User-Oriented Interface Modelling," *Proceedings of* 1*st International Workshop*, *Towards a UML Profile for Interactive Systems Development*, York, October 2000.

[21] N. J. Nunes and J. F. Cunha, "Towards a UML profile for interaction design: the Wisdom approach," *Proceedings of* 1*st International Workshop*, *Towards a UML Profile for Interactive Systems Development*, York, Springer Verlag, October 2000, pp. 101-116.

[22] "Microsoft Expression: Sketchflow Overview". http://www.microsoft.com/expression/products/SketchflowOverview.aspx

[23] "Unified Modeling Language: Superstructure–Specification Version 2.0," Object Management Group, August 2005. http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf

[24] "DSM Forum: Domain-Specific Modeling". http: //www.dsmforum.org

[25] "UWE–Tutorial". http://uwe.pst.ifi.lmu.de/teachingTutrial.html

[26] M. Brambilla, "The Web Modeling Language," Politecnico di Milano. http://home.dei.polimi.it/mbrambil/webml.htm

[27] N. Moreno and P. Fraternalli and A. Vallecillo, "WebML Modeling in UML," *Institution of Engineering and Technology Software*, Vol. 1, No. 3, June 2007, pp. 67-80.

