

# Platform Independent Web Application Modeling

Pierre-Alain Muller<sup>1</sup>, Philippe Studer<sup>1</sup>, and Jean Bézivin<sup>2</sup>

<sup>1</sup> ESSAIM Université de Haute-Alsace  
12 rue des Frères Lumière  
68093 Mulhouse Cedex, France  
{pa.muller, ph.studer}@uha.fr

<sup>2</sup> ATLAS Group, INRIA & IRIN  
Université de Nantes  
2, rue de la Houssinière, BP 92208  
44322 Nantes cedex 3, France  
jean.bezivin@irin.univ-nantes.fr

**Abstract.** This paper discusses platform independent web application modeling in the context of model-driven engineering. A specific metamodel (and associated notation), companion of the UML metamodel, is introduced and motivated for the modeling of dynamic web specific concerns. Web applications are represented in three independent aspects (business, hypertext and presentation). A kind of action language (based on OCL and Java) is used throughout these aspects to write methods and actions, specify constraints and express conditions. The concepts described in the paper have been implemented in a tool and operational model-driven web information systems have been successfully deployed.

**Keywords:** model-driven engineering, MDA, web, metamodel, PIM.

## 1 Introduction

At the end of the year 2000, the OMG proposed a radical move from object composition to model transformation [1], and started to promote MDA (Model Driven Architecture) a model-driven engineering framework to manipulate both PIMs (Platform Independent Models) and PSMs (Platform Specific Models). The OMG also defined a four level meta-modeling architecture, and UML was elected to play a key role in this architecture, being both a general purpose modeling language, and (for its core part) a language to define metamodels. As MDA will become mainstream, more and more specific metamodels will have to be defined, to address domain specific modeling requirements. Examples of such metamodels are CWM (Common Warehouse Metamodel) and SPEM (Software Process Engineering Metamodel). It is likely that MDA will be applied to a wide range of different domains.

We found interesting to apply the MDA vision to web engineering; a field where traditional software engineering has not been very successful, mostly because of the gap between software design concepts and the low-level web implementation model [2].

We believe that model engineering gives the opportunity to reinject good software engineering practices into web application developments. Models, together with aspects, favor the collaborative work while preserving different stakeholder's points of view. Graphic designers should be able to continue to create static presentation artifacts, and the software engineers should use models to explain how these static artifacts (or part of them named fragments) get combined and augmented with dynamic business information coming from the business model and hypertext logic coming from the hypertext model

The work described in this paper has been done in the context of the development of Netsilon [3] a visual model-driven environment dedicated to web application development. We will present a metamodel specific to dynamic web page composition and navigation. This metamodel has to be used as a companion metamodel of UML in order to build PIMs for web information systems. A graphic notation, based on directed graphs, will also be presented.

## 2 Web Applications

A web application is an information system which supports user-interaction through web based interfaces. Typical web applications feature data persistence, transaction support and dynamic web page composition.

A web application is split into a client side part, which is running in a web browser, and a server side part, which is running on a web server. The client side is responsible for page rendering while the server side is responsible for business process execution and web page construction. The process of page construction varies widely in dynamicity, ranging from completely static, in the case of predefined HTML pages, to totally dynamically constructed pages, when the HTML pages are the result of some computation on the server.

A web interaction can be decomposed into three steps:

- Request. The user sends a request to the web server, usually via a web page already visualized in a web browser. Requests can be sent to the server either as forms or as links.
- Processing. The Web server receives the request, and performs various actions so as to elaborate a web page, which contains the results of the request. This web page is then transferred to the web browser from where the request originated.
- Answer. The browser renders the results of the request, either in place or in another browser window.

A web page may be composed of several kinds of graphic information, both textual and multimedia. These graphic components are mostly produced with specialized authoring tools, using direct manipulation and wysiwig editors.

When it comes to visualize a web page in a web browser, these various components have to be glued together by HTML formatted text, which is either embedding some of the page content (for instance the text) or referencing the files that contains the data (for instance the images). This process may involve translations as well, for instance to translate XML code into HTML.

In the case of dynamic web pages, the final HTML formatted text is not stored on the server, but is generated at runtime, by programs either compiled (like Java) or

interpreted (like PHP). These programs integrate into web pages the graphic elements and the information coming from many kinds of data sources (like databases, files, sessions or context variables...). To increase performances, pages may be cached, so that dynamic pages do not have to be generated when their sources have not been modified.

Building web applications requires several different qualifications. Typical worker roles include:

- Marketing to define the target audience and the content to be delivered.
- Graphic designers to define the visual appearance (including page layout, fonts, colors, images and movies).
- HTML integrators to develop HTML pages and fragments to web enable the visual appearance.
- Programmers to write programs (in Java, PHP or other script languages) which will generate the dynamic pages, by combining HTML fragments and information coming from databases and current context.
- Content writers to feed the application with value added information.

Page layout and graphic appearance is an area where a lot of creativity comes into play. Graphic designers use imaging and drawing tools to generate images (as well as animation and movies) stored in binary files. Graphic designers often collaborate with HTML integrators, who know how to write efficient HTML code. HTML integrators implement the graphic charter into web pages; this involves compressing and splitting the images, mapping the fonts to style sheets, establishing links, making buttons, writing Javascript for rollovers and writing HTML text to embed all these various components. Occasionally, they also have to integrate the data produced by the content writers. They use web authoring tools, automatic or semi-automatic HTML generators, which store their production into files, either as textual notation (HTML) or as some binary data (GIF, JPEG, FLASH...). Current tools mainly export HTML (and not XML), and the separation between page content and page layout is poor.

As long as these teams used to produce static web sites, they had no real technical problems. Things changed as they started to develop more and more dynamic sites with the help of programmers. Web applications are far more demanding; they are real software systems, and they require following a software development process. This was kind of a cultural chock; a lot of web agencies were unable to overcome the challenge. Nowadays, web applications are mostly built by software houses, they have gained in dynamicity, but they do not really progress in graphic creativity, because it is difficult to write the programs that would animate sophisticated graphic charters. It is also difficult to define an optimal development process; communication and coordination between all these different roles is often a challenge [4].

We believe that model engineering and MDA, is an excellent opportunity to reconcile graphic designers with programmers, and to increase the overall productivity. The challenge is to define the right modeling concepts; to find how to introduce the power of model transformation in a world (the art of graphic design) traditionally hostile to any kind of rule enforcement.

### 3 Modeling Web Applications – Related Work

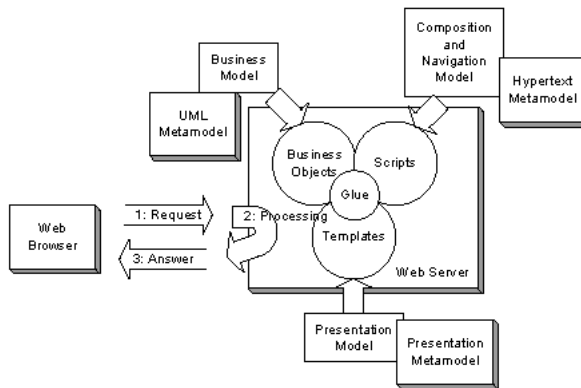
Before going further, let's take four definitions.

- By presentation we refer to the visual elements that compose a web page. These elements contain textual, graphic and multimedia elements (images, animations, movies...).
- By navigation we refer to the network of paths within the application, in other words all the possible scenarios of pages a user can browse through.
- By composition we refer to the process of constructing a web page by combining several fragments together.
- By business model we refer to the description of the business classes and their relations.

Several different approaches are undertaken in the modeling community to model web applications:

Schattkowsky and Lohmann [5] are defining an approach for small- to medium-size applications; they plan to generate page and code skeletons. They stay close to the current state of UML tools, like Rational Rose, which generate partial code. We take a more radical position; we want to be able to build PIMs from which totally executable applications can be generated.

Conallen [6] focuses on notation and has defined a UML profile; the main advantage of this approach is that UML modeling tools may be reused to model the web; the drawback is that Conallen mainly represents implementation, and is therefore suitable for PSMs rather than PIMs.



**Fig. 1.** Multi-aspects modeling of web applications.

The WebML [7] people emphasize conceptual modeling and have defined four modelling aspects, the structural model (data content, entities and relation), the hypertext model (composition and navigation), the presentation model (layout and graphic appearance of page) and the personalization model (individual content based on users and groups).

At a first glance our work may seem close to WebML as we have retained three aspects, the business model, the hypertext model (composition and navigation) and

the presentation model (see fig. 1). Aspect modeling justifies itself in the context of web modeling, because web applications are complex systems involving several different stakeholders (marketing, graphic-designers, developers, content writers...). Therefore it is very convenient to separate models into several aspects, and to be able to focus on each aspect independently of the other. However there are major differences with WebML; we have a complete object model, including operations and methods, our hypertext model is decorated with conditions and constraints written in a model-aware action language, and the presentation model has been designed so that existing web authoring tools can be integrated seamlessly, without changing the work habits of graphic designers and HTML integrators.

### 3.1 Metamodel or Profile

We found that it is not obvious to choose between making a new metamodel or profiling an existing one.

A metamodel defines a specific domain language. It may be compared to the formal grammar of a programming language. The MOF (Meta Object Facility) can be used to specify metamodels; this involves defining classes, attributes, relations and constraints.

A lighter alternative to making new metamodels, is to customize existing ones. Therefore, MDA provides facilities (known as profiles) to extend or constrain existing metamodels (by terms of stereotypes and tags). Interestingly, some metamodels may also be defined as UML profiles, as it is the case for SPEM.

We took the position that creating a new metamodel and associated notation would make more sense when the semantic distance between existing UML modeling elements and newly defined modeling elements is becoming too large.

The Conallen extensions describe subtypes of coarse-grained web implementation artifacts and profiling UML classes or components is fine for that. In our case, we have defined new modeling concepts, which have little in common with classes, objects or states. There is no obvious inheritance between our modeling elements and UML modeling elements. Desfray [8], who has been very active about profiles, explains that defining a new metamodel instead of a profile is justified when the domain is well defined and has a unique well-accepted main set of concepts; this is precisely the case for web-based user interfaces. We also had a practical reason to define a metamodel; as explained by Atkinson, Kuehne and Henderson-Sellers [9], meta-modeling in terms of stereotypes lacks transitivity in the derivation of properties, and inheritance-based approach was important in the design of our tool. A last reason that pushed toward a metamodel (and associated notation) was the fact that we could not reuse existing UML tools anyway, because their user interfaces were not aware of the specific behavior that we wanted to give to our modeling elements (most notably by using position in graphics to convey ordering information).

Therefore, considering the context of our work, we estimated that profiling UML was not adapted and that it was justified to define a new metamodel to be used in conjunction with UML.

### 3.2 The Business Model Aspect

The first aspect, the business model, goes well beyond WebML's entity-based structural model. We use UML class diagrams to represent the business classes, their attributes and operations, and, in addition to WebML, we are using a kind of action language (named Xion) to describe the behavior of the methods. In the context of UML, S. Mellor has enumerated the requirements [10] for such an action language. Xion follows a large part of these requirements, excepted for the points related to separation of data access and computation, state charts and signals. The syntax of Xion is based on the syntax of OCL augmented with Java-like control structures and affectation. Therefore, to the contrary to OCL which is a side effect free language, Xion can be used to create, update and delete instances at runtime; it has direct support for association navigation, role exploration and collection manipulation. We are currently generating either Java or PHP (and the embedded SQL statements) from the same Xion statements; in this respect, Xion is truly part of PIMs.

Besides business description, we also use this aspect to describe pervasive web services, like session management, personalization, search or statistics. When required, packages may be used to partition the business model, and separate the various concerns.

### 3.3 The Hypertext Model Aspect

The second aspect, the hypertext model, is an abstract description of composition and navigation between document elements and fragments of document elements. In the context of web modeling this model describes how web pages are built and linked. In a wider context, it can also be used to handle multi-channels distribution, mixing electronic and paper documents, as we have experienced in earlier work about document modeling [11].

Composition describes the way the various web pages are composed from other pages (fragments) as well as the inclusion of information coming from the business model or current context (like sessions). Again, Xion is used as a query language to extract information from the business model and as a constraint language to express the various rules, which govern page composition.

Navigation details the links between the web pages. Navigation includes the specification of the parameters that will be transferred from page to page, as well as the ability to specify actions to be applied when triggering a link. The Xion language allows the specification of the actions to be performed when transitioning from one page to another and the declaration of predicates that lead to the selection of a particular path between pages according to the current context and the business model.

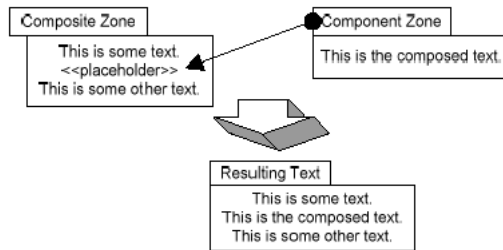
The hypertext model makes it possible for a tool to check the coherence and the correctness of the navigation at model compilation time. This removes all the troubles related to parameter passing and implementation language boundary crossing (mix of interpreted languages, un-interpreted strings, parameter passing conventions...).

### 3.4 The Presentation Model Aspect

The third aspect, the presentation model, contains the details of the graphic appearance of web applications.

The goal of the presentation model is to make it possible for graphic designer and HTML integrators to keep their work habits; to be able to produce static HTML pages and fragments by using conventional tools. It does not make sense to create another way of specifying the graphical appearance of web pages. Therefore, we have not provided explicit support to model the graphical appearance of the user interface, because we consider that wysiwig authoring tools are already available, and perform a good job to cover this aspect. We believe that existing tools must be integrated, without change, in the process of dynamic web page development, but that their production must be able to be controlled by an implicit presentation model.

In this vision, a dynamic web application is composed of fragments, which can be developed as static HTML, supplemented with some special placeholders, easily identifiable by graphic designers and HTML integrators. Whenever some dynamic information must be inserted into a web page, the graphic designers simply designate the spot in the file where this information must be inserted (see fig 2).



**Fig. 2.** At runtime, the placeholder is replaced by the component text.

The consequence is that we have shifted the focus of modeling to the parts that are out of the scope of these web authoring tools, and that require typically to program complex behavior, using conventional programming languages like Java or PHP.

## 4 Modeling Elements for Web Page Composition and Navigation

We start by defining an abstract metaclass `WebElement` derived from `ModelElement`. Web elements are intended to capture web design elements at any kind of granularity (specifically with much finer grain than URLs), so that individual links and fragments of text or layout can be taken into account.

Web elements are specialized in `WebFiles` (which contain or reference presentation artefacts) and `Zones`. `DecisionCenters` explain how web files relate together.

### 4.1 Webfiles and Zones

Web files are statically associated to real files on disk (which contain data suitable for rendering in web browsers). Web files may be considered as fragments, in which case they are necessarily included in some enclosing web file (potentially also a fragment) until a non-fragment web file is reached. At the time of transformation of PIMs into executable PSMs, such top-level non-fragment web files become entry points in the web application. They are translated into target language, and are executed on the server (they can be referred to by an URL).

Zones further specialize web files to better promote separation of concerns between graphic designers and programmers. They have no statically associated files; their content is generated or retrieved at runtime, they make it possible to describe web pages at a purely conceptual level, establishing a very clean separation between logic and presentation. Using zones, software engineers can model a web user interface without entering in the presentation details, while graphic designers can focus on appearance, using conventional tools as if they would be doing static web sites.

A zone is an abstract representation of some chunk of information, which makes sense in a given context. A zone is not aware of the type of content it refers to. Zones are not limited to web development; they can be used to represent any kind of content. In the specific case of web pages, a zone refers to some characters stream, which can be understood by a web browser; the simplest example of zone content would be some HTML formatted text. As this paper focuses on web development, we will often refer to HTML text in the following lines, although there is no limitation to HTML for zones.

### 4.2 Decision Centers

Decision centers represent variation points in the web user interface. Each of these centers is responsible for a decision to be taken at runtime, while the user is interacting with the system. Decision centers support an entry action, which has access to session variables and may define local variables.

We have identified six kinds of decision centers to cover the complete range of variation points in web user interfaces (see fig 3).

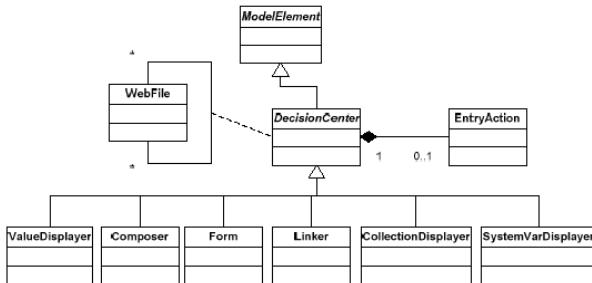








Fig. 3. Simplified excerpt of the metamodel. Web files are related by decision centers.



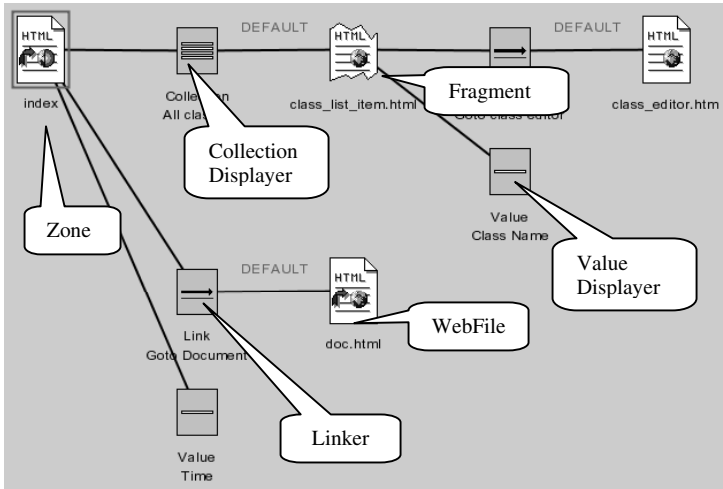
We give below the definition and graphic representation of these six kinds of decision centers.

Icon	Name	Description
	Composer	Composers compose fragments into pages. A composer selects a target fragment to be inserted in place of the placeholder
	Value displayer	Value Displayers display single values. A value displayer evaluates an expression, converts the result in a character string and inserts this string in place of the placeholder in the generated text.
	Collection displayer	Collection displayers display collections of values. A collection displayer acts as a composer applied iteratively to all the items in the collection. For each element a specific target fragment may be chosen.
	Linker	Linkers link webfiles to other non-fragment webfiles. Linkers augment the navigation power of static HTML links, because they can point to various targets, and change the state of the system when activated.
	Form	Forms link webfiles to other non-fragment webfiles. Forms handle the HTML forms. All input elements that can appear in a form are considered as local variables in the context of the form and are initialized with the value posted during the submission of the form.
	System variable displayer	System variable displayers display platform specific system environment variables, like HTTP server name, server root path, or target language extension.

Composers, Collection Displayers, Linkers and Forms require a least one target web file. Potential targets are ordered in a sequence, and each target is guarded by a Boolean condition. At runtime, when the web page is generated, the decision center evaluates the conditions in the order of the sequence, and the first expression that resolves to `True` determines the choice of the target web file. In case there is no such `True` condition, the decision center selects no web file and an empty string replaces the placeholder. It is possible to specify a default decision, which will be chosen if no other was taken.

## 5 Graphic Notation

We have rendered the composition and navigation model under the shape of a directed graph whose principal nodes are web files. The edges are either composition relations between pages and fragments (or between fragments themselves) or hypertext links between pages. In fact, on a finer level of granularity, the composition relations or hypertext links are themselves nodes of the graph and are modeled by decision centers. An example of hypertext model for composition and navigation is given in figure 4.



**Fig. 4.** : Example of hypertext model for composition and navigation. Nodes are web files; edges are decision centers. The graph is implicitly directed; it grows from the left to the right, and from the top to the bottom.

While on a static picture this kind of graph may seem somehow similar to a class diagram, it is important to note that the behavior of the user interface is fundamentally different from a class diagram editor. Differences have to do with ordering of the modeling elements (evaluation based on relative vertical position), elided cycle representation, representation of conditions and several other minor details.

As hypertext graphs can be huge, we have defined visualization principles, which focus on one web element at a time. The view is split in three swim lanes; the left most contains the current web file, the middle one shows all the decision centers which belong to the current web file, and the right most one the several targets for the current decision center. The graph is directed from the left to the right. Thus in the case of a composition, the left most element is likely to contain one or more elements of the right-hand side and in the case of navigation, the left most element is the hypertext link holder, while an element of the right-hand side represents a potential target web file (see fig 5).

## 5.1 Detailed Metamodel for Web Page Composition and Navigation

Figure 6 presents an overview of our metamodel for web page composition and navigation. The central element is `WebFile`, which describes a document element (or a fragment). A web file can be treated as a static or a dynamic element: if static, it is simply copied unmodified during deployment; if dynamic, it is generated as server side code and participates in the dynamic part of the web application. HTML tag filtering and stripping makes it possible to use web-authoring tools for the design of fragments as easily as for the design of entire pages.

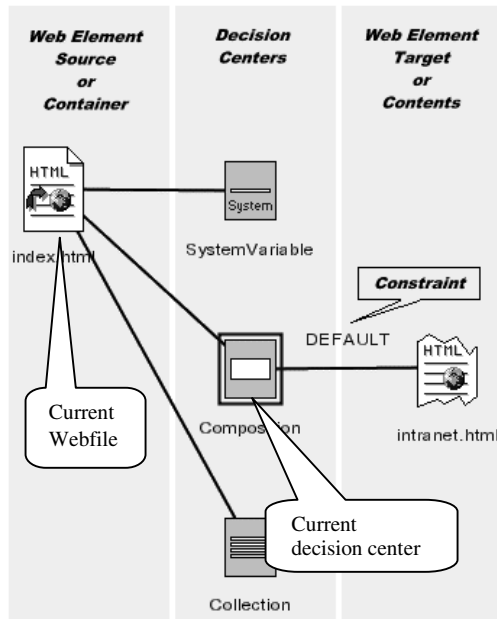


Fig. 5. Example of split view of the hypertext model.

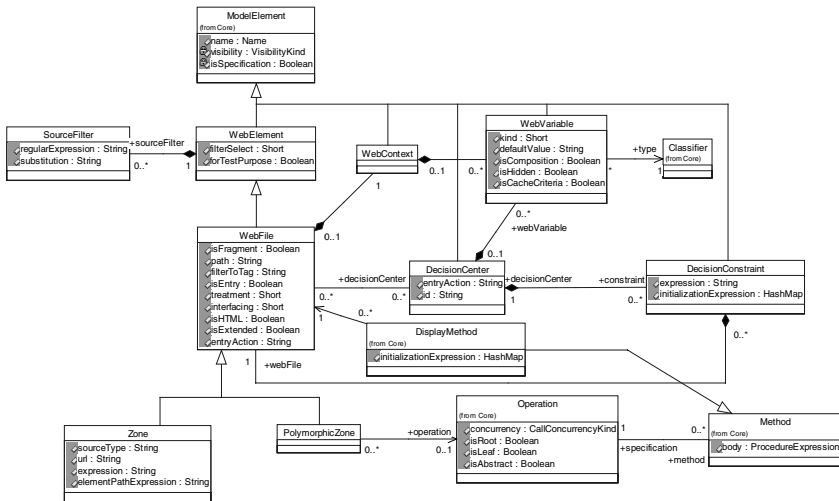


Fig. 6. Excerpt of the hypertext metamodel for web page composition and navigation.

A web file can be specialized as a Zone whose associated content is obtained dynamically by the evaluation of an expression that returns the URL of the content or the content itself.

A web file can also be specialized as a PolymorphicZone. A polymorphic zone is the mean to introduce the object notion of polymorphism in the composition model. In fact, a polymorphic zone is associated to an Operation that is in charge of producing the content. Since the operation can be implemented by overridden methods, the content can be generated according to the real class of an instance. To reinforce the separation between the business model and the hypertext model, we introduce a subclass of Method named DisplayMethod that is associated to a web file. The production of content by an Operation implemented by one or more display methods is thus realized by webfiles.

Each web file has a context WebContext that describes its entry parameters. A parameter is described by WebVariable and has a type, which is an instance of Classifier.

DecisionCenter define variation points in the hypertext model. A decision center has an *entryAction*, a unique *id* to identify its placeholder, local variables (WebVariable) and an ordered sequence of DecisionConstraint. A decision constraint defines a guard whose evaluation to true leads to the selection of its associated web file.

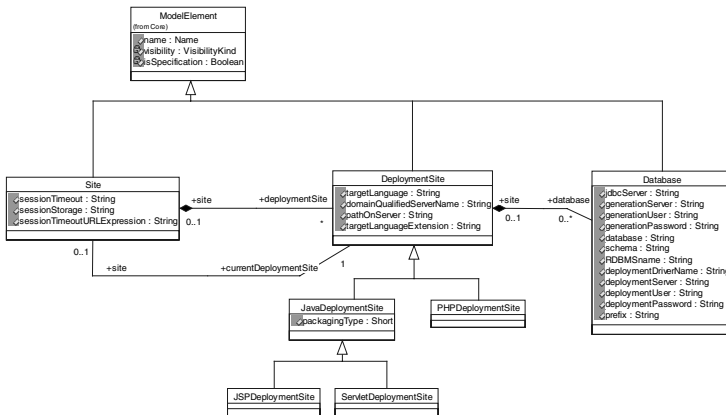


Fig. 7. Excerpt of the metamodel for transformation from PIMs to web PSMs.

## 5.2 Transformation of PIMs into PSMs

The transformation of the PIMs into executable PSMs has been coded in Java. Several different deployment sites can be defined for the same project. A deployment site specifies a target environment and a persistence system (see fig 7). Information describing the current deployment site is taken into account at code generation time, when the PIMs are translated into executable PSMs. The encapsulation provided by the decision centers (SystemVariableDisplayer, Linker, Form) allows the design of effective PIM for the hypertext model. The business model in the PIM is transformed

as a set of classes in the target language and a database schema. The hypertext model is transformed as a set of static documents, classes or scripts in the target language. Finally, the action language is translated in the target language.

A deployment site specifies a target environment (web application server, language, site information). Our code generator currently targets any combination of (PHP, J2EE) applications server with (Oracle, MySQL, PostgreSQL) databases. Our tool is able to transparently create and incrementally update the target database according to the modifications applied to the PIM.

## 6 Conclusion and Future Work

We have applied model-driven engineering in the field of web engineering. We believe that the MDA initiative is a good opportunity to raise the level of abstraction, and increase the productivity in the development process of web applications, provided that the overall process does not change the working habits of graphic designers and HTML integrators. Model developments remain the duty of software developers.

We have favored the development of a specific metamodel dedicated to web application development (instead of a profile) to better cope with specialized behavior and tool user-interface. This metamodel has to be used in conjunction with UML and promotes concerns' separation between graphic development and software development.

Our experience shows that web user interfaces can be modeled using a small and well-defined set of modeling elements (3 types of web files and 6 types of decision centers). An obvious question about this work has to do with completeness of the metamodel; do we have identified enough concepts to model any kind of web applications? In fact, we have followed a very iterative approach, and the metamodel has been validated by the development of about a dozen of significant totally model driven web applications (for online examples visit <http://www.domaine.fr>, <http://www.namestep.fr/>, <http://www.solinest.fr/>, <http://www.mydocmail.net>, <http://www.prh-france.fr>). We have achieved complete model engineering; we can generate various executable PSMs from the same PIMs.

Another question is about level of abstraction; what about the granularity of our web modeling concepts? We have been looking for a balance between expressiveness and design freedom, and so our modeling elements should be considered as relatively fine grained. The next step would be to develop either patterns or wizards, providing shortcuts, and here we could probably reuse work done in the WebML effort. This also includes adding support for state charts, especially in the area of session management (as many web applications have to maintain state information for every sessions).

It would also be interesting to re-align our action language with the on-going standardization efforts led by the OMG; this involves better understanding the interactions between OCL, the action semantics and QVT [12]. Another major point would be to make the PIM to PSM transformations explicit, and therefore allow customization of the code generation phase.

## References

1. J. Bézivin, “From Object Composition to Model Transformation with the MDA”, in proceedings of TOOLS’2001. IEEE Press Tools#39, pp. 350–354 . (August 2001).
2. H.-W. Gellersen, M. Gaedke, “Object-Oriented Web Application Development”, IEEE Internet Computing, pp.60–68, January-February 1999.
3. W. El Kaim, O. Burgard, P.-A. Muller, MDA Compliant Product Line Methodology, Technology and Tool for Automatic Generation and Deployment of Web Information Systems, Journées du Génie Logiciel, Paris, December 2001.
4. A. McDonald, R. Welland, “Web Engineering in Practice”. Proceedings of the fourth WWW10 Workshop on Web Engineering, 21–30, May 2001.
5. T. Schattkowsky, M. Lohmann, “Rapid Development of Modular Dynamic Web Sites Using UML”, UML 2002 Conference, LNCS 2460, pp. 336–350, 2002.
6. J. Conallen, “Building Web applications with UML”. The Addison-Wesley Object Technology Series, 2000.
7. S. Ceri, P. Fraternali, A. Bongio, “Web Modeling Language (WebML): a modeling language for designing Web sites”, Ninth International World Wide Web Conference, May 2000.
8. P. Desfray, “UML Profiles versus Metamodeling Extensions... an Ongoing Debate”, Uml In The .Com Enterprise: Modeling CORBA, Components, XML/XMI And Metadata Workshop, 6–9 Novembre 2000, Palm Springs.
9. C. Atkinson, T. Kuehne, B. Henderson-Sellers, “To Meta or not To Meta – That is the Question“, Journal of Object-Oriented Programming, 13(8): 32–35, 2000.
10. S. Mellor, S. Tockey, R. Arthaud, P. Leblanc, “An Action Language for UML: Proposal for a Precise Execution Semantics”, UML 98, LNCS 1618, pp. 307–318, 1998.
11. S. M.-C. Roch, P.-A. Muller, B. Thirion, “Improved flexibility of a document production line through object-oriented remodeling”, Second congress IMACS, Computational Engineering in Systems Applications, Hammamet, CESA’98, Vabeul-Hammamet, Tunisie, Vol III, pp 152–159, April 98.
12. OMG, MOF 2.0 Query/Views/Transformations RFP), OMG document ad/02-04-10, April 2002.