SPECIAL ISSUE PAPER

**Pierre-Alain Muller · Philippe Studer ·
Frédéric Fondement · Jean Bezivin**

Platform independent Web application modeling and development with Netsilon

Abstract This paper discusses platform independent Web application modeling and development in the context of model-driven engineering. A specific metamodel (and associated notation) is introduced and motivated for the modeling of dynamic Web specific concerns. Web applications are represented via three independent but related models (business, hypertext and presentation). A kind of action language (based on OCL and Java) is used all over these models to write methods and actions, specify constraints and express conditions. The concepts described in the paper have been implemented in the Netsilon tool and operational model-driven Web information systems have been successfully deployed by translation from abstract models to platform specific models.

Keywords MDA · PIMs · PSMs · Web application development

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^{TM} [2] (Model-Driven Archi-

P.-A. Muller (✉)
INRIA Rennes, Campus de Beaulieu, Avenue du Général Leclerc,
35042 Rennes, France
E-mail: pierre-alain.muller@irisa.fr

P. Studer
ESSAIM/MIPS, Université de Haute-Alsace, 12 rue des Frères
Lumière, 68093 Mulhouse, France
E-mail: ph.studer@uha.fr

F. Fondement
École Polytechnique Fédérale de Lausanne (EPFL) School of
Computer and Communication Sciences CH-1015 Lausanne,
Switzerland
E-mail: frederic.fondement@epfl.ch

J. Bezivin
ATLAS Group, INRIA & LINA, Université de Nantes, 2,
rue de la Houssinière, BP 92208, 44322 Nantes, France
E-mail: jean.bezivin@lina.univ-nantes.fr

tecture) 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 it 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 [3].

We believe that model engineering gives the opportunity to re-inject good software engineering practices into Web application developments. Models, together with views, favor the collaborative work while preserving different stockholder's points of view. Graphic designers should be able 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 from the Hypertext Model.

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. The work described in this paper has been done in the context of the development of Netsilon [4] a visual model-driven environment dedicated to Web application development. The paper is organized as follows: Sect. 2 provides an overview of Web application development. Section 3 presents the kind of models that we use for Web application modeling. Section 4 details the novel modeling elements that we have

introduced for composition and navigation modeling. Section 5 presents the Xion language that we use to express both constraints and actions. Section 6 discusses the transformation of PIM into PSM. Section 7 examines related work. Finally, in the last section we draw conclusions and outline future work.

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, and can be considered as hybrids between a Hypermedia [5] and an information system.

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 (which may be composed of several processors). 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 (which vary in terms of content, presentation and navigation), 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 WYSIWYG editors.

When it comes to visualizing 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 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 performance, 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 adapt the visual appearance to the Web.
- Programmers to write programs (in Java, PHP or other scripting 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 chart 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 shock; 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 charts. It is also difficult to define an optimal development process; communication and coordination between all these different roles is often a challenge [6].

Fraternali [7] provides very good criteria to classify tools and approaches for developing Web-applications, and these criteria are very convenient to understand the scope of a research project related to Web engineering. He presents the development of Web applications as a multi-faceted activity, involving technical, organizational, managerial, social and

even artistic issues. He distinguishes three major design dimensions: the structure, the navigation and the presentation. He classifies the tools in five categories: Visual editors and site managers, Web-enabled Hypermedia authoring tools, Web-DBPL (Database Programming Language) integrators, Web form editors, report writers and database publishing wizards, and Model-driven application generators.

As we have seen, Web application development is becoming an important challenge and the generative aspects are more and more apparent in the associated process. 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 the world of graphic designers and Web application developers.

3 Modeling Web applications

3.1 Definitions

Model-driven engineering has received a lot of attention recently, especially with the raise of the MDA initiative of the OMG. Meanwhile, there has been a blossoming of new terms [8] and TLAs (Three Letter Acronyms), like PIM or PSM. However, basic notions like the platform concept itself are still loosely defined, and therefore for the sake of precision (and in the context of the work presented in this paper) we will explain what we mean by the following notions:

- By (Web) platform we refer to an execution environment for Web applications, based on two fundamental technical items: databases and application servers.
- By PIM we refer to a model which is independent of the (Web) platform. This means that the specificities of databases and/or application servers are totally abstracted away from the PIM.
- By PSM we refer to artifacts which depend on the Web platform. We consider executable programs as some kind of PSM.
- 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 Web 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 fragments we refer to excerpts of Web pages which become meaningful in the context of a container.
- By Business Model we refer to the description of the business classes and their relations.
- By dynamic Web application we refer to the late binding between the content of the information base and the Web pages delivered to the client.

- By models we refer to descriptions which conform to metamodels which in turn conform to the MOF.
- By model-driven we refer to an engineering process which takes models as input and generates models and other artifacts as output.

3.2 Context and scope of the work

The research presented in this paper takes place in the context of the development of Netsilon, a visual model-driven tool for Web application development.

The main target is the Business-to-Consumer segment of the Web application development market. These applications are characterized by strong constraints in terms of usability and performance. The number of users can be large, and the application must be able to react reasonably, even during rush hours (when the number of simultaneous sessions can grow by several orders of magnitude). Users are very sensitive to the quality of the interaction, both in terms of visual presentation and in terms of navigation. Often, the content, presentation and navigation are customized, based on explicit or implicit profiling of the users, and usage statistics are gathered to enhance the business processes.

From a lifecycle point of view, we provide support mainly for the development activities, but we seek to replace code-driven development by model-driven development. We do not want to develop a new method for Web application development, nor do we want to enforce a specific methodology. Gomez et al. [9] have listed the common notions shared by the most relevant methods and methodologies for Web development (UWE [10], WebML [11], OO-H [9]...). The major similarities are:

- A specific navigation model and an agreement that a new notation is required.
- A clean separation between the content, navigation and presentation spaces.
- The necessity of a constraint language to augment the precision of Web application models.

We have taken these common notions as a basis of our work, and we have focused our attention on the navigation model, by defining novel modeling concepts dedicated to Web application modeling, with the double goal of total code generation and total freedom of the graphic designers. In addition, we want to provide tool support for incremental and iterative development, in the content, navigation and presentation spaces.

3.3 Web platform for dynamic Web applications

As defined earlier, a Web platform is an execution environment for Web applications, based on two fundamental technical items: databases and application servers.

An application server is an execution environment, integrated with a HTTP server, which is able to host and execute server-side code, to support the Web interactions

defined in Sect. 2. The server-side code implements the business logic (which is Web-independent) and the user interaction logic (which is Web-dependent) implemented *as application server scripts*.

Some examples of concrete application servers are:

- J2EE servers which support the Java language (Servlets and Java Server Pages), and include Tomcat, JBoss, BEA WebLogic, IBM Websphere, Oracle Application Server and Sun Java System Application Server.
- Microsoft proprietary.Net Windows Server
- PHP application servers

In addition to application servers, Web applications require databases, most usually relational, like MySQL, Oracle or SQL Server.

A Web platform can therefore be considered as a tuple (application server, database server). In our context, a model which contains no information specific to either application server or database is said to be platform independent.

Based on this definition, the level of dependence of a model to the Web platform can then be analyzed more in details. The dependences that we have identified and taken care of are summarized in Table 1.

These dependences have to be abstracted away from the PIM. In the Netsilon tool, they are specified as parameters of the platform (we talk about *deployment site*), and only used late in the process of application generation, when the PIMs are translated into the final executable PSMs, see Sect. 6.1 for more details.

Table 1 Platform dependences

Source of dependence	Kind of dependence
Application server	URL or IP address of the HTTP server
	Root path of the Web application
	Mode of file transfer (copy or FTP) for automatic deployment purpose. In case of FTP (URL or IP address of the server, port, login, password, and remote directory). In case of copy (remote directory).
	Session information stored in URLs, cookies or both (using URL if cookie technology is not available on the client).
	Target language (PHP, JSP, Servlet…) In case of PHP (target language extension as.php,.php4, etc.). In case of Java (compiler, jar tool, class path, etc.).
Database server	Support of SQL (MySQL does not support inner queries).
	Support for local transactions
	Name or Id of the database
	Table prefix (to restrict them to a category of names).
	DB access from the IDE (server name and port, user and password) for automatic deployment purpose.
	DB access from the generated application server scripts (server name and port, role name, user and password)

3.4 The business model

Once the requirements have been gathered (via techniques such as usecases and activity diagrams), the Web application is modeled from three points of view, the *Business Model*, the *Hypertext Model*, and the *Presentation Model*. These models are independent of the Web platform; they capture a comprehensive description of the Web application. With the help of Xion, an action language described later in the paper (in Sect. 5), they contain enough information to drive the generation of the final Web application.

The Business Model describes the organization of the business concepts managed by the Web application. For an e-business application, typical examples of such concepts are *products*, *catalog*, or *cart*. We use UML class diagrams to represent business classes, their attributes, operations and relations. The implementation of methods is specified with the action language Xion.

The Business Model is used as an input of the model-driven process which generates the business layer of the Web application. Object persistence is provided by a relational database, whose schema is derived from the Business Model, using guidelines as described for instance by Marcos et al. [12]. The object-to-relational mapping is designed so that incremental modifications to the Business Model have as little impact as possible on existing information in the database; we talk of M0 preserving transformations.

The relational database is completely abstracted away. The classes from the Business Model are all persistent by default, the code is completely generated.

The advantage is that the designer does not have to care about persistence, the process of creating and updating the database schema is completely automated, and implementation classes for the Business Model are generated in the Web platform target language (either Java or PHP).

The drawback is that persistence does not work with an already populated database, because the M0 preserving transformations do not work with arbitrary database schema, and therefore legacy databases cannot be readily reused. Data must be imported in the object database, and this may raise some synchronization issues. However, this is not necessarily an issue for the Internet (versus Intranet), as the databases which are online (hosted by Web access provider) are usually different from those in the organizations.

Besides business descriptions, Netsilon uses this model to automate Web specific concerns, like session management, personalization, profiling, search or statistics. When required, packages may be used to partition the Business Model.

3.5 The hypertext model

The second model, 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 [13].

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. As an example for an e-business application, the page which shows the content of a cart is typically composed of the repetition of another page showing the details of a given product. The current cart can be retrieved from the current session and the products in the presented cart and a Xion expression would retrieve the products. This collection is then iterated for including the product presentation page, which is given the product to show as a parameter, into the cart presentation page.

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 (typically to execute a method defined in the Business Model). 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. For instance, a link labeled "Add to cart" would add a given product to the cart.

The Hypertext Model makes it possible for a tool to ensure the consistency and the correctness of the navigation at model checking 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...) encountered when programming Web applications by hand. The graphic notation for composition and navigation is presented later, in Sect. 4.4.

3.6 The presentation model

The third model, the Presentation Model, contains the details of the graphic appearance of Web applications. As mentioned earlier, we do not want to restrict the field of possible graphic designs and technologies. We want to be able to generate any kind of Web user interface.

We have defined the Presentation Model with the goal to make it possible for graphic designers and HTML integrators to keep their work habits and to be able to build dynamic Web applications while using conventional tools which produce static HTML pages or fragments.

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 WYSIWYG authoring tools are already available, and
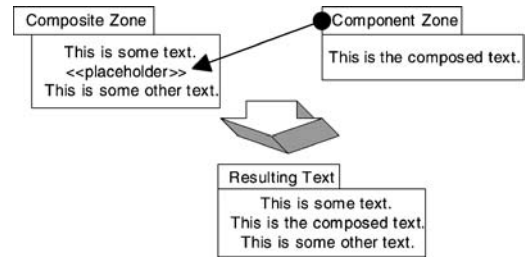

Fig. 1 At runtime, the placeholder is replaced by the component text

perform a reasonable 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. The Presentation Model is therefore embedded in the Netsilon Tool which provides a repository for the files produced by the graphic designers.

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. 1). The Presentation Model is not limited to HTML and can be used to manage any other textual formalism like JavaScript, XML or SVG.

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 typically require to program complex behavior, using conventional programming languages like Java or PHP.

We have made a tradeoff between freedom of the programmer and freedom of the Web designer, clearly in favor of the latter. Our goal is to provide an automated solution for the software part, able to embed any kind of artistic production of the Web designers, and then to generate a reasonable implementation.

Figure 2 shows how the Web application results from the weaving of the three different kinds of models described in
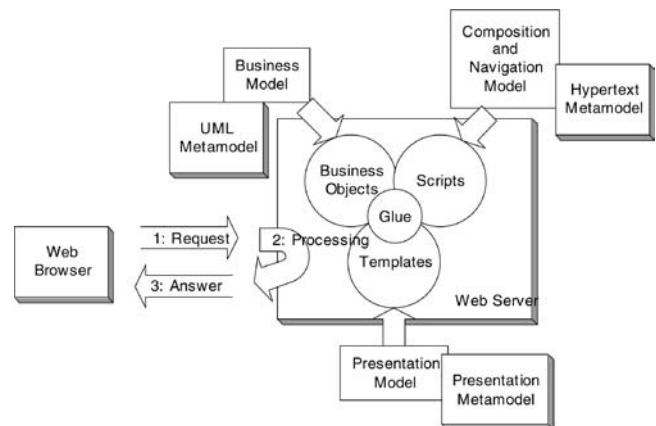

Fig. 2 Multi-view modeling of Web applications

the previous subsections. A much more detailed discussion of the generation process is given in Sect. 6.

3.7 Reusing models

Reusability is an important concern in software development. It is of paramount importance to be able to cluster a consistent and generic set of modeling elements addressing generic purposes (classes, relations, decision centers, Web files, zones, HTML files...) in a reusable Web application component. These Web application components can then be further reused in Web applications models, or by other Web application components. Examples of such "*off the shelf*" Web components are *Logging* and *Forum*. It is interesting to note that the Forum component can actually reuse the Logging component.

To achieve such reusability, we have defined a standalone packaging element (the *WebModule*), and, for the sake of interoperability, the concepts of *required* and *provided interfaces*, which are featured by both Web applications and Web modules. Interfaces are either *business interfaces* in the Business Model (defining operations only), or *Web file interfaces* in the Hypertext Model (defining the parameters to be exchanged with the realizing Web file, and whether it should be a fragment or not).

Web models that use these Web modules must define for each required interface the realizing element, with the constraint that this element must fulfill the surrounding contract. We will further call such realizing element a *mapped realization*. A mapped realization can be either an element of the Web application, or a provided interface of a used Web module. Note that the same Web module can define both a required interface and a provided interface that can realize it. For the Web application to be executable or compilable, all required interfaces must be matched by exactly one mapped realization. An action on a required interface will actually end up in an action on the mapped realization, e.g. instantiating a business interface will actually instantiate the mapped business class.

In the next section, we will detail the novel modeling elements that we use for Web page composition and navigation modeling.

4 Model elements for web page composition and navigation

Our goal is to achieve total code generation from models, while making no restrictions on the visual appearance of the Web application. Therefore, we have been defining novel modeling constructs, for the description of the composition of Web pages from various sources of information, and for the specification of the navigation between pages. These models are precise, abstract and independent of the platform. They are later transformed into executable code which runs on the Web platform. The new modeling elements have been packaged in a new metamodel (see Fig. 3).

4.1 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, i.e. the abstract syntax. The MOF [14] (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 means of stereotypes, tags and constrains). However, with such extension technique, it is difficult to define concepts that are radically different from those already defined by the metamodel before profiling. Interestingly, some metamodels may also be defined as UML profiles, as it is the case for SPEM [15].

In this paper we define metamodelling as the approach consisting of writing brand new domain languages wrt profiling as the customization of an existing metamodel. The relations between these two approaches are still generating much debates going beyond the scope of this paper. We take here the position that creating a new metamodel and associated notation will make more sense when the semantic distance between existing UML modeling elements and newly defined modeling elements is becoming too large.

The Conallen [16] 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 [17], who has been very active in the definition of UML 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 et al. [18], metamodeling 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.

4.2 Web files and zones

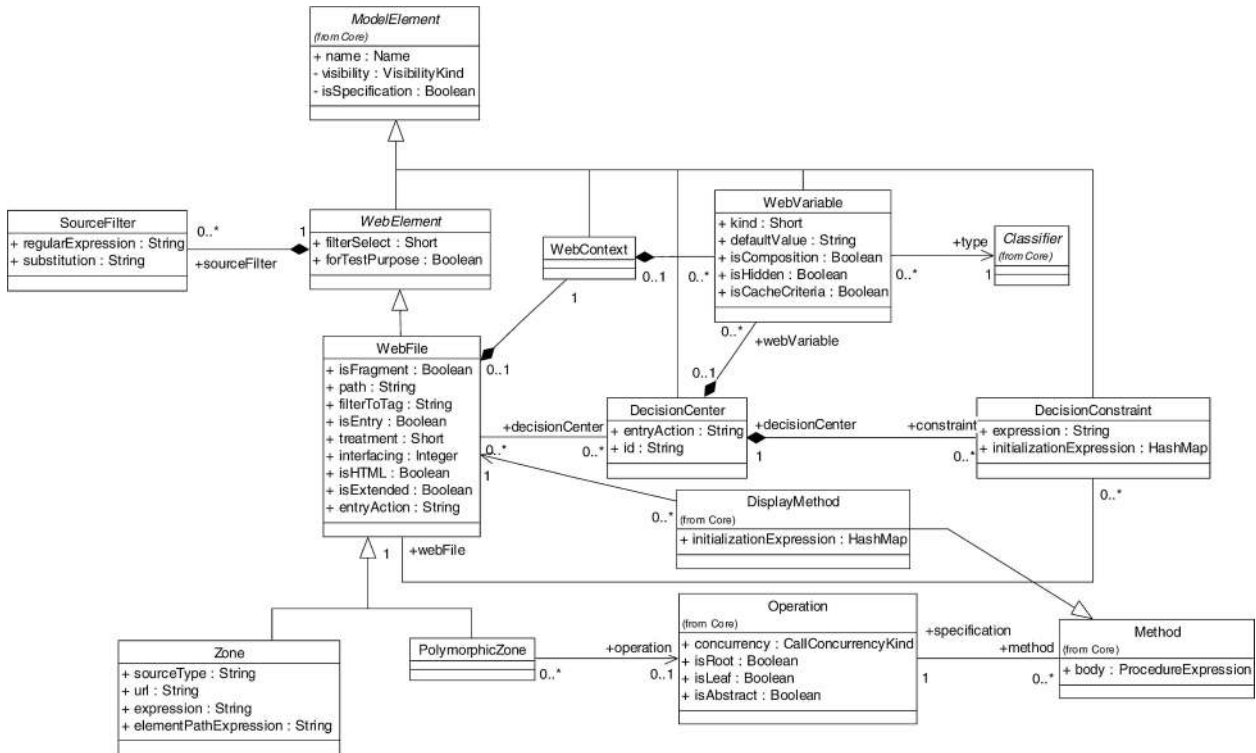Figure 3 presents an overview of our metamodel for Web page composition and navigation.

Fig. 3 Excerpt of the hypertext metamodel for Web page composition and navigation

We have started 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 presentation artifacts) and `Zones` and `Polymorphic Zones` (which both refer to presentation artifacts). Web Files may be standalone or 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. HTML tag filtering and striping makes it possible to use Web-authoring tools for the design of fragments as easily as for the design of entire pages.

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`. At the time of transformation of PIMs into executable PSMs, the 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). Web Files make it possible to describe Web pages at a purely conceptual level, establishing a very clean separation between business and navigation logic, and presentation. Using Web Files, 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 were doing static Web sites.

Zones provide support for late binding to their actual content which can be generated or retrieved at runtime. Therefore, it is possible to generate and release a generic model-driven Web application, which can be customized later by graphic designers and content writers, without having to get back to programmers. A zone is a representation for 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.

Web Files can also be specialized as Polymorphic Zones. A Polymorphic Zone is the means to introduce the notion of polymorphism in the Hypertext Model (based on a subtype in the Business Model). In fact, a Polymorphic Zone is associated to an operation defined in a class (of the Business Model) which is in charge of producing the content. Since the operation can be implemented by overridden methods in the subclasses, the content can be generated according to the real class of an instance of the Business Model. 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 `WebFile`. The production of content by an operation implemented

Table 2 Definition of decision centers and graphic representations

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 a Xion 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 denoted by a Xion expression. For each element a specific target fragment may be chosen.
	Linker	Linkers link Web Files to other non-fragment Web Files. 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 Web Files to other non-fragment Web Files. 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 values 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.

by one or more display methods is thus realized by Web files.

4.3 Decision centers

`DecisionCenters` explain how Web Files relate together. 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 Center 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 `WebFile`.

We have identified six kinds of decision centers to cover the complete range of variation points in Web user interfaces. Table 2 gives the definition and graphic representation of these six kinds of decision centers.

`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 written in Xion. 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.

At transformation time, the PIM is translated into PSM and decision centers are translated into programs (either in Java or in PHP), which are downloaded on the application server. At execution time, they are executed (actually, the programs which were generated from them), and the dynamic Web pages are generated on the fly. The content of these pages will then vary accordingly to the abstract description captured by the Hypertext Model.

4.4 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 represent 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 Fig. 4.

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
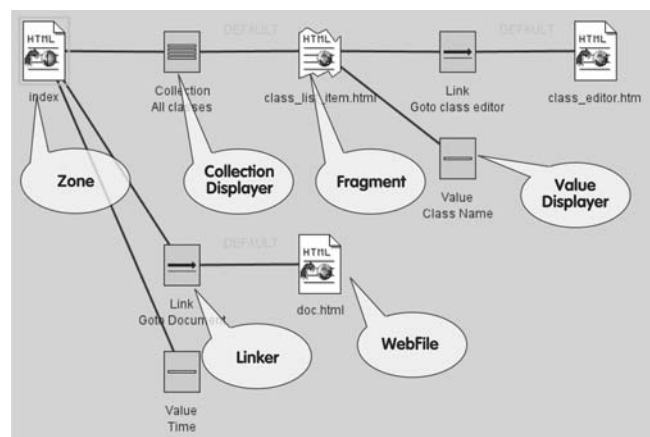
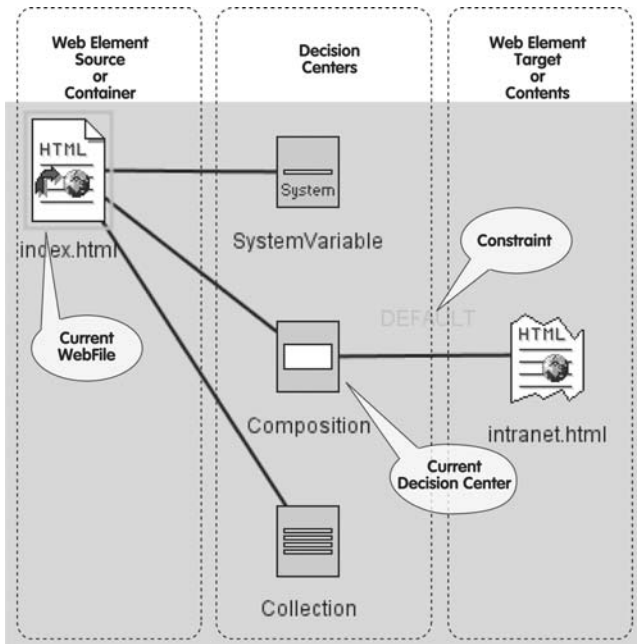**Fig. 4** Example of hypertext model for composition and navigation

Fig. 5 Example of split view of the hypertext model

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 for an example).

5 The Xion action language

A stated by Mellor et al. [19]: "An action language, in conjunction with UML, can be used to build complete and precise models that specify a problem at a higher level of abstraction than a programming language or a graphical programming system."

We have implemented such an action language (named Xion) in the Netsilon tool, in order to be able to generate fully executable Web applications from platform independent models. Xion is used in the Business Model to express the methods and in the Hypertext Model whenever a constraint or a behavior has to be expressed. Xion is not used in the Presentation Model, which contains only presentation artifacts.

The behavior of dynamic Web applications depends on the overall state of the system. This state is stored for part in databases (which implement persistence of the Business Model) and files (HTML templates or cookies for instance), but also in volatile memory (data of the currently active sessions and parameters). Xion has been designed to provide uniform access to all kinds of state in the system, and is used to describe queries, constraints and actions. A typical example would be to have the possibility to look for a specific set

of objects according to some constraints, like retrieving a customer command from a character string carried by a Web page parameter or contained by a cookie.

In the next paragraphs we present code generation approaches adopted by CASE tool vendors, examine two specifications related to UML (the Action Semantics and the Object Constraint Language) and finally describe the Xion language more in detail.

5.1 Code generators

One of the key ideas of the MDA is that models can be used to build programs using model transformations, therefore moving the modeling process from contemplative to productive [1]. Many CASE tools generate useful code fragments out of models. We may distinguish three categories of code generators depending on the degree of completeness of the PIM and resulting PSM: *skeleton generation*, *partial generation*, and *full generation*.

Skeleton generation has been adopted by most CASE tools, especially UML CASE tools. This kind of generation is a partial generation of the system which deals only with the static structure. For object-oriented technologies, this means generating classes, attributes, relations and operations, but not method bodies. However, this can be considered as MDA as it is a transformation from an abstract model to a specific set of programming languages and execution environments (for instance generating code for CORBA/C++ or EJB/JAVA). The code generation remains incomplete and further refinements have to be done at code level. This rupture raises synchronization issues, between the generated and the hand-modified code, and ruins the platform independence. Some tools propose specific annotations (comments in the code) to instruct the code generator not to overwrite code which was modified out of their scope, at the expense of making the code less readable; in the end, the PSM becomes the reference model. Other tools perform the refinement job directly in the source model, making it thus platform specific as well. Anyhow, changing the target platform means developing again these refinements, according to the selected platform. An example of such a tool is AndroMDA [20].

Partial generation is going one step further than skeleton generation by taking a more complete specification as input for the code generation. Behavior, when considering UML, may be modeled by statecharts, collaboration diagrams, sequence diagrams or activity diagrams. However, these diagrams are most of the time "incomplete" at PIM level, mainly because there is no precise relation between the dynamic and static models. For instance, there is no standard way to specify a terminate action or control flows like if-then-else or loop statements. The result of a partial generation is code that needs to be further refined to integrate these changes, even if it already integrates this time some significant pieces of behavior implementation. Examples of such tools are Parallax [21], which specifies behavior in

collaboration diagrams, and UMLAUT [22], which takes statecharts as input complemented with pieces of Eiffel code. Once more, changing generation platform means manually refining the generated code.

The last kind of generation is the *full generation*. In this case, tool vendors introduce a new PIM language to complement the modeling elements. Generated applications can then be fully functional without any refinement required at PSM level. One of the drawbacks of this kind of approach is that developers must learn a new language. Another issue is that it supposes to have a development environment for the new language, including syntactic editor, debugger and high-quality code generators. There are several examples of such kind of approach, like iUML [23], Projtech BridgePoint [24], Kabira Object Switch [25], or Telelogic Tau architect and developer [26].

5.2 Action semantics

In late 2001, the OMG has integrated support for the description of actions [19] in UML (the abstract syntax is standardized as part of UML 1.5 [27]). The Action Semantics is a model for specifying actions at the PIM level. The Action Semantics is not a concrete action language. Action languages are free to provide more sophisticated constructs, as long as these constructs can be translated into the basics concepts defined by the Action Semantics. The details of concrete syntaxes are left to so-called surface languages, which have to comply with the Action Semantics. This is indeed what tools supporting the Action Semantics have done. For instance iUML [23] has defined the ASL language, BridgePoint [24] the AL language, Object Switch the AS language and Tau has even defined two syntaxes: a textual one and a graphical one [26]. All these languages have the Action Semantics as their abstract syntax. This makes theoretically possible to exchange "executable" UML models among them; exactly like compiled Java programs are supposed to work the same way whatever virtual machine is actually interpreting them.

The Actions Semantics specification considers that all actions may execute concurrently, unless explicitly stated differently, either by a data flow or by a control flow. The goal is to allow reorganization of the actions to achieve the most efficient implementation as possible.

The Action Semantics can be used to define:

- entry, exit and do activities of states in statecharts,
- effects of transitions in statecharts,
- stimulus and message triggered activities in interaction diagrams (collaboration and sequence diagrams),
- method and constructor bodies in class diagrams,
- expressions that are used in many places like initial values of attributes, derived attributes and associations, transition guards, event conditions, etc.

The specification is very detailed and describes notably how to query and manipulate the static models, i.e.:

- create and delete objects,
- get and set attribute values,
- call an operation,
- create, delete and traverse links.

The Action Semantics is an imperative and structured language that allows control flow, like conditional and loop statements. The language can be used on its own, or in complement of other diagrams, like statecharts or interaction diagrams.

5.3 Object constraint language

OCL stands for Object Constraint Language [28], but OCL is actually much more than a language to express constraints. OCL has been defined with several different purposes in mind:

- as an object-oriented query language,
- to specify invariants on Classes, Types and Stereotypes,
- to describe pre- and post-conditions on Operations and Methods,
- to describe Guards,
- to specify derivation rules for Attributes,
- for any expression over a UML model.

OCL derives from earlier work done in the context of Business Modeling within the IBM insurance division, under the influence of Syntropy [29] itself influenced by Z [30]. The language has been defined both mathematically [31] and as a MOF 2.0-compliant metamodel, which defines the concepts and semantics of the language.

OCL is a textual formal language, used to describe expressions on UML models, that remains easy to read and write for the average business or system modeler. With OCL it is possible to refer directly to elements in UML models, like classes, attributes, operations, roles, associations... OCL is a pure specification language; an OCL expression cannot change anything in the model and it simply returns a value. The evaluation of an OCL expression is instantaneous; the states of objects in a model cannot change during evaluation.

OCL is not a programming language, although it may be used to specify a state change (e.g., in a post-condition). Like the Action Semantics, OCL is a platform independent language. OCL has become rapidly very popular in the UML community because of its simple yet powerful capability to navigate through model instances, especially class-diagrams. Another reason is that the OCL designers managed to provide a syntax quite easy to use for the average programmer, despite the strong mathematical foundations of the language. OCL is used in numbers of other projects dealing with the UML or its MOF subset. For instance, the OCL plays an important role in the upcoming QVT standard [32] which aims at querying, viewing and transforming models: the OCL is a good candidate for the query part, and can ease the task of selecting parts of models to transform or view.

5.4 Description of the Xion language

Xion is a platform-independent action language which abstracts away the details of data access, while being translatable into different target languages (like PHP or Java – see Sect. 7).

Xion is based on OCL and not the Action Semantics for historical reasons. When we defined Xion, OCL was already a standard but the Action Semantics specification was not yet completed. If we had to design Xion nowadays, we would certainly take a subset of the Action Semantics, although there is no standard way to "remove" elements of an abstract syntax. Nevertheless, we would still face the issue of defining a concrete syntax, and as navigation is so important to us, we would certainly base this syntax on the one of OCL again. Xion has to provide support to query the Business Model and to express methods and state changes. OCL is a good candidate for querying instances of the Business Model, and further, as shown by other declarative approaches, like the B predicates [33] or the Alloy [34] language, OCL could also be used to define the state changes performed by methods, via constraints to specify the state before and after the method execution. However, we did not chose that solution, because generating efficient code out of a declarative specification is still an open issue [19], and because most software developers are more used to the imperative approach, like the Action Semantics, with a well-defined sequence of actions to perform.

Thus, we have decided to extend the OCL *query expressions* to define a new imperative language, Xion, for our actions and queries needs. This means to add side-effects capability to OCL, and to provide imperative constructs, like blocks and control flows. In the context of the Business Model, supporting side-effects means:

- create and delete an object,
- change an attribute value,
- create and delete links,
- change a variable value,
- call non-query operations.

It was also necessary to remove some constructs of the OCL, which are out of the scope of our approach:

- context declaration, only useful for defining constraints,

- @pre operator and message management, only meaningful in the context of an operation post-condition,
- state machine querying, as there is no equivalent concept in the Web architecture we propose.

Since most Web application developers are already familiar with the Java language, we re-used part of its concrete syntax. Constructs we took from Java are:

- instruction blocks, i.e. sequences of expressions,
- control flow (if, while, do, for),
- return statement for exiting an operation possibly sending a value,
- "super" initializer for constructors.

Moreover, for Xion to look like Java as much as possible we decided to keep Java variable declaration, and operators (==, !=, +=, >>, ? ternary operator, etc.) rather than those defined by OCL. The standard OCL library was also slightly extended, by adding the `Double`, `Float`, `Long`, `Int`, `Short` and `Byte` primitive types, whose size is clearly defined unlike the OCL `Integer` or `Real`. As Web application often deals with time, we have also added the `Date` and `Time` predefined types.

In the following paragraphs, we present some examples of Xion code.

Considering the Business Model shown in Fig. 6, an example of the Xion language for implementing the marry operation of the class Person is provided below in Fig. 7.

As we can see here, Xion looks like Java with `if`/`else` control blocks, the `null` value and the `return` statement. Notice that `this` and `self` can be used indifferently. We can also see that enumeration literals are treated as OCL 1.3 prescribes, starting with a # sign.

Another example is provided in Fig. 8. Here, we can better feel the OCL affiliation of the language. This example is a parameter transmitted to a Web file of the Hypertext Model. The target Web file is in charge of displaying the given list of `Person`. This Web file is integrated into the calling Web file by means of a "Composer" decision center. The purpose, here, is to display sisters of a given person, represented by the `person` variable. First we navigate the `parents` association end. This navigation returns the *set* of `Person` instances representing the parents of the `person` instance. To get all children of these parents, we then navigate from the obtained instances through the `children`
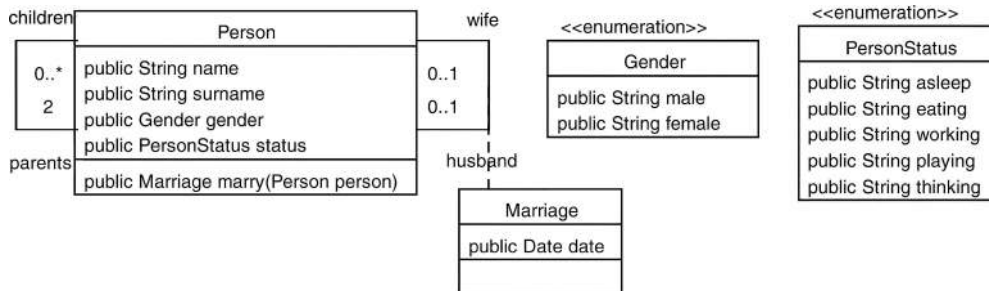

Fig. 6 Business model for a family management system

```
//person is parameter of the operation
Marriage ret = null;
if (this.gender == person.gender) {
      ret = null;
} else {
      ret = new Marriage();
      ret.date = Date.getCurrent();
      Person wife, husband;
      if (self.gender == #female) {
            wife = this;
            husband = person;
      } else {
            wife = person;
            husband = this;
      }
      ret.wife = wife;
      ret.husband = husband;
}
return ret;
```

Fig. 7 Example of Xion code, implementation of a method

association end. Note that this last navigation directly comes from the implicit `collect` predefined operation call defined in the OCL (UML 1.5, Sect. 6.6.2.1) [27]; we obtain indeed a *bag* of `Person` instances, which is the union of navigation results from each parent over the children association end.

As a consequence, the `Person` instance and his/her siblings, with the same father and the same mother, will appear twice in the resulting collection. "Half siblings" will only appear once for they have only one common parent. As we are not interested in making a difference between sisters and half sisters, we remove duplicate instances with the `asSet` OCL predefined operation. To remove `person` from this collection, we use the OCL `excluding` predefined operation. In this list, we only want sisters of `person`; this can be done by selecting only instances which are declared to have a female literal value in their gender slot. This is achieved by the OCL `select` predefined operation. The `select` operation is what OCL calls an iteration operation, i.e. an operation that applies a given expression iteratively to each element of the input collection. The "*iterated*" element can be given a name by being declared just before the parameter of the operation. This is the purpose of the "p :" in the expression. In real OCL, this should be done by a "p |", but Xion defines the | operator as the Java bitwise `or` operator, which is why this iterating variable declaration ends with a column. Once the `select` predefined operation has performed, we want the obtained collection to be ordered by name. This is achieved by another predefined iteration operation `sortedBy`.

In the former example, we examined a Xion expression very similar to an OCL query. The following expression stresses two of the main differences between OCL and Xion, i.e. control flow and side effects:

```
if (person.gender == #female)
 person.children
   ->select(c : c.status == #asleep)
        ->collect(c: c.status = #eating);
```

This expression is used to wake up children in the morning. The `person` variable represents the `Person` instance supposed to play the role of the mother. Therefore, we use an `if` statement to make sure that the procedure is only performed when `person` is a female. Then, her children which are still sleeping are selected. For all of them, the attribute `status` is updated with the `eating` enumeration literal, using the `collect` iterating predefined operation. As for OCL, `collect` gathers the results of an iteration expression for each element of the input collection. In this case, the expression is an attribute assignment. In Xion, an assignment returns a `Void` value. As a collection of `Void` is not meaningful, the result of the expression is `Void` as well, rather than `Bag(Void)`.

We have only given an overview of the Xion language here, and provided small examples of Xion statements. An exhaustive presentation of the language is given in the help of the Netsilon tool, and some practice is required to appreciate the language.

We have presented in this section an action language based on OCL and extended with some Java constructs, and we have described shortly its concrete syntax.

At the time we designed Xion we have had to implement the abstract syntax from scratch. Today we could have benefited from the advance of the UML standard, in two ways. We could either extend the OCL abstract syntax which is defined by a MOF metamodel (OCL 2, Sect. 8) [28] via the profile mechanism or we could use parts of the Action Semantics. Going one step further, we could also define model transformations to translate Xion expressions from abstract syntax trees (conform to a profiled OCL metamodel) to valid Action Semantics models, for instance for interchange purposes. As a summary, although it has been originally defined on top of OCL, Xion can be seen as a concrete syntax for a subset of the Action Semantics.

6 Translation from PIM to PSM

The combination of the Business Model, Presentation Model, and Hypertext Model defines a complete specification for a given Web application. However, in order to deploy the application, it is still necessary to integrate configuration data of the platform, as introduced in Sect. 3.3. From this point, the Netsilon tool can then generate and deploy the fully executable code for the selected platform.
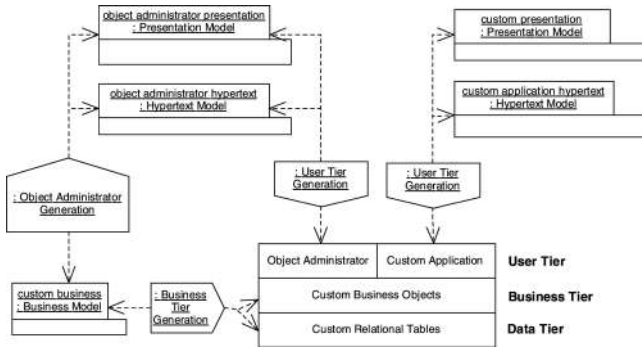
```
person.parents.children->asSet()->excluding(person)
       ->select(p : p.gender == #female)->sortedBy(p : p.name)
```

Fig. 8 Xion expression to query the sisters of a person

Fig. 9 Overview of the Web application generation process

In the following paragraphs, after an overview of the Web application generation process, we introduce the PIM to PSM transformation process of the Netsilon tool, discuss the process on a concrete example, and highlight performance issues to be solved to build real applications beyond simple prototypes.

6.1 Overview

The Web application is generated as a three-tier application. Figure 9 shows how the three kinds of models are transformed into the different layers.

The *Business Tier Generation* transformation reads the Business Model and creates both Business and Data Tiers. The *User Tier Generation* transformation reads the Hypertext and Presentation Models and creates the User Tier.

In addition, a generic Web application (the *Object Administrator*) is generated by derivation of the Business Model, for the purpose of administrating the objects stored in the database. The Object Administrator application is actually generated in a two-step process by the *Object Administrator Generation* transformation which derives Hypertext and Presentation Models from the Business Model, and then by the *User Tier Generation* which generates the generic user interface of the tool. Such derivation of the Hypertext and Presentation Models from the Business Model is achievable because the user interface of the Object Administrator does not have to be customized, and can therefore follow simple user interaction patterns.

The Object Administrator is very similar to the phpMyAdmin [35] tool, with the notable difference that interaction is achieved at the object level. Objects can be created and deleted, their attributes can be updated, links can be created between objects, and operations can be executed. This administration tool is very helpful, and is used both for initialization of the object database and for data maintenance purposes.

6.2 Compilation process

As already stated in Sect. 3.3, we consider our platform to be the composition of a database and an application server.

As a consequence, the possible number of different deployment platforms is the Cartesian product of the supported databases and application servers. Netsilon supports the following technologies:

- MySQL, Oracle 8i, and PostgreSQL for the database,
- PHP, JSP and Servlet for the application server.

The translation from PIM to PSM goes through two PSM steps: a platform-dependent layer, and a technology-dependent layer.

The transformation process from the PIM to the technology-dependent code is presented in Fig. 10. The PIM is the combination of the three platform-independent models: *Business Model*, *Hypertext Model*, and *Presentation Model*.

Xion is considered part of the Business Model, but can be used in the Hypertext Model, which is actually referring to both Business and Presentation Models.

The first layer of the PSM is the combination of the *SQL Abstract Syntax* and a generic application server script, called *Intermediate Language*. The SQL Abstract Syntax is a subset of the SQL 92 language, supposed to be managed by most RDBMS. This syntax allows managing database schemas (*Schema Management*), for creating, deleting, or modifying table schemas. This SQL abstract syntax also allows data access to the recordsets (*Data Access*), for instance by `Select` queries (*Query*). The Intermediate Language is an abstraction of the concepts of scripts that application servers handle. The *Classes* can define structure of Web application classes. Behavior is expressed by *Instructions*, possibly database manipulations (*Database Request*), depending on the above-mentioned *Data Access*. *Scripts* are specialized in integrating pieces of server-side behavior in files to be sent to the client, expressed with *Instructions*.

Since business information is to be stored in a database, a transformation *Object Relational Mapping* creates a database schema from the *Business Model*. In the case a database schema already exists, the transformation will alter it as necessary. The business information, to be easy to integrate and reuse, is encapsulated into *proxy* server-side classes by the *PSM Business Logic* transformation. Behavior in the *Business Model*, described in Xion statements as bodies of constructors and methods, are also translated by this *PSM Business Logic* transformation inside corresponding server-side classes, taking advantage of the *Xion to Intermediate* transformation. The *Presentation Model* and the depending *Hypertext Model* are compiled together by the *Navigation to Scripts* transformation to produce *Scripts* of the *Intermediate Language*.

Dynamic Web pages may be cached in files to avoid systematic computation each time a page is called (with the same parameters' values). The current implementation is based on timestamps. A better approach would be to trace a cached page back to the values that were actually taking part in the computation of the page, and then to remove the page from the cache whenever one of the values is updated. This is an area that we would like to explore in future work by using trace in model transformations.
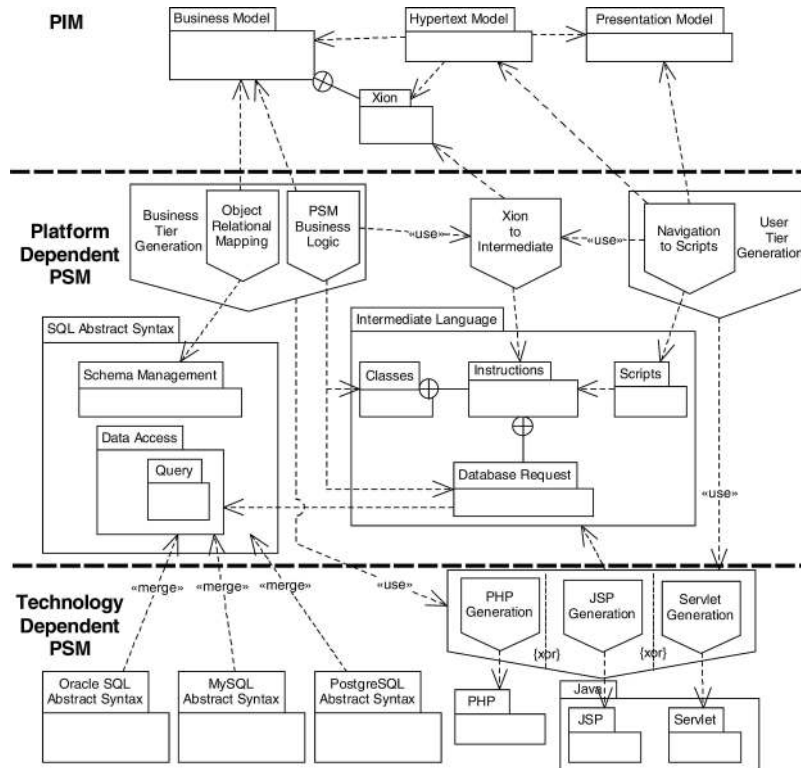
Fig. 10 The Netsilon compilation process

The transformation *Business Tier Generation* introduced in Sect. 6.1 is the composition of the *Object Relational Mapping* and *PSM Business Logic* transformations. Both transformations take as input the *Business Model* and produce respectively the *Data Tier* layer and the *Business Tier* layer. The *User Tier Generation*, which is responsible for generating the *User Tier* embeds the *Navigation to Scripts* transformation. Finally, the two transformations *Business Tier Generation* and *User Tier Generation* use a target language transformation to refine the *Intermediate Language* into the concrete *PHP*, *JSP* or *Servlet* technologies.

This transformation phase, and the following deployment phase, cannot be performed without a precise platform description. Some additional information must be provided to resolve the platform dependencies as described in Sect. 3.3. This generation configuration is provided as a deployment model whose metamodel is presented in Fig. 11.

The Site metaclass is the main container of the complete deployment information. It can define several DeploymentSite, which conform either to JSPDeploymentSite, ServletDeploymentSite, or PHPDeploymentSite, for defining which application server is to be targeted, and where to export generated and static Web Files. Each deployment site uses a certain number of databases for storing business information. The chosen platform for generation is referenced in the site by the currentDeploymentSite association end. Deploying the same Web application on another platform is done by changing the DeploymentSite referenced in the

currentDeploymentSite association end. Creating a new deployment platform is done by providing information requested in Sect. 3.3, without changing anything in the PIM for Web application.

Some strong assumptions are made in the *Intermediate Language* about what an application server can deal with. These assumptions may limit target technologies: it may be difficult to integrate the ASP technology that is not object-oriented. However, these choices are reasonable and worked well considering the Java and the PHP technologies, and one can imaging easily integrating the .Net application server technology by generating C# code.

6.3 SQL optimization

As stated in Sect. 5, the Xion language is responsible for manipulating business information. This business information is stored in relational databases and is encapsulated in proxy scripts served by a Web application server. In the previous paragraph, we have explained how the compilation process translates Xion statements into scripts that access these proxies.

Unfortunately this approach leads to inefficient code because it generates too many database requests. In order to reduce the number of requests, and the amount of data to be transferred between the application server and the database server, we have had to insert an SQL optimization pass in the transformation process from PIM to PSM.
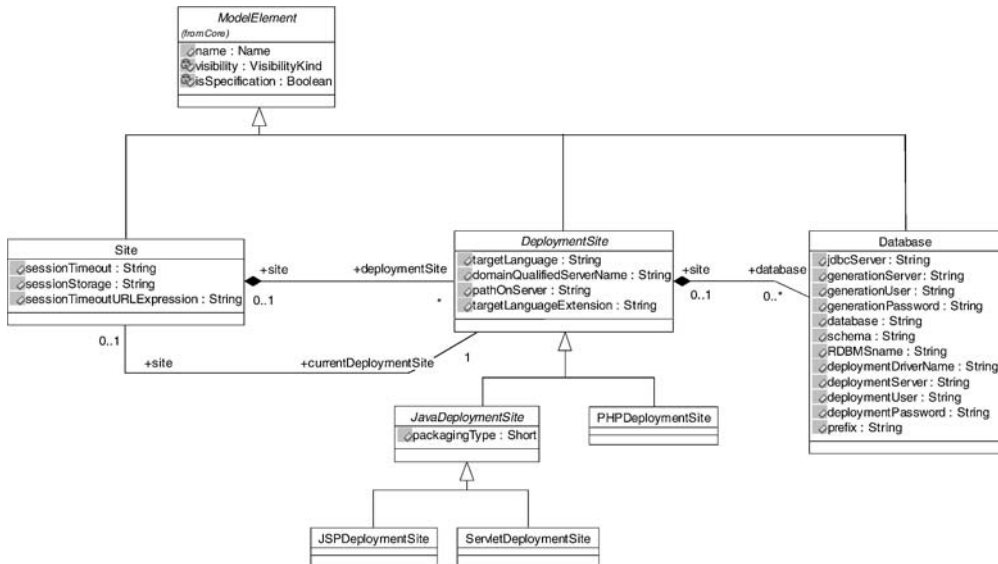
Fig. 11 Excerpt of the metamodel for Web platform specific information

We will get back to our example to discuss this issue of optimization. The database schema produced from the model presented in Fig. 6 is the following:

```
person (person_id, name, surname, gender,
  status)
person_person (#parents, #children)
marriage (marriage_id, date, #wife,
  #husband)
```

The first table named `person` stores `Person` instances. This table contains four fields: the `person_id`, as a primary key, is a unique identifier for stored objects; the `name`, `surname`, `gender`, and `status` store the attributes of the person. The `person_person` table implements the association between `parents` and `children`. It defines two primary keys: `parents` and `children`, that are also foreign key on the `person_id` field of the table `person`. The `marriage` table stores the `Marriage` association-class. Because `Marriage` is both a class and a recursive association on the `Person` class, the table defines the `marriage_id`, `wife` and `husband` primary keys, with `wife` and `husband` also foreign keys on the `person_id` field of the table `person`. Pseudo-code snippets of the proxy script for encapsulating the data about persons are provided in Fig. 12.

The proxy is a script class that provides access to the data through SQL queries on the database. This class has a reference on the corresponding record in the database by means of an attribute, named `OID` in the Fig. 12. This attribute has the same value as the primary key in the record corresponding to the object, in this case `person_id`. Each attribute defined in the Business Model is encoded in the script in getter and setter methods, plus an attribute acting as a reading cache for the value. The first time an attribute is read, the `activate` method is invoked and performs an SQL query for initializing the `name` attribute which is used

as a cache for the name. The getter method, `get_name` in the example, requests an activation, which is only performed if needed, and returns the value of the associated cached attribute. The setter method, as `set_name` in the example, sends an update to the database, and updates the cache.

Access to association-ends is performed in a slightly different way, as there is no reading cache in this case. This is due to the fact that attributes are not well suited for storing associations. In this case, only getter and setter methods are provided, using SQL queries and updates, as shown by `get_parents` and `get_children` getter methods. Methods and constructors in the Business Model are also translated into scripts, but this is not shown in the example.

The translation for the Xion statement responsible for retrieving sisters of a person, which is previously presented in Fig. 8, is provided below in Fig. 13.

Some temporary variables are used here, as `_t1`. The first `for` statement retrieves siblings of `person`, with the duplicate problem already explained in Sect. 5.4. An iteration is made for each parent of `person`, who is queried for all its `children`. The result is made available in the `_t1` temporary variable. Duplicates are then removed by the predefined operation `asSet`, and person is removed from the collection by means of the `excluding` predefined operation. The second `for` statement iterates on this modified collection, and tests for each `Person` instance whether his/her gender is female. In this case, the instance is added to a new collection in another temporary variable `_t4`. The last `for` statement iterates on this collection, and references each element in the `_t7` temporary variable, result of the query. The latter variable is of a special collection type, specialized in ordering elements according to a certain criteria, using the *insert sort algorithm* [36]. This algorithm was chosen because it allows cascading sorts (to support the Xion language which allows cascading `sortedBy` predefined operations). In this example, the criterion is the name of the element.

```
class Person

attribute OID : String;
attribute name : String;

method activate()
  if self.needsActivation()
  then
    copy_to_attributes(
      'SELECT name, …
      FROM    person
      WHERE   person_id = '
              + self.OID);
  endif
end method activate

method get_name() : String
  self.activate();
  return self.name;
end method get_name

method set_name(new_name : String)
  execute_SQL('
    UPDATE person
    SET    name = ' + new_name +
    'WHERE  person_id = ' + self.OID);
    self.name = new_name;
end method set_name

method get_parents() : Set(Parents)
  return to_Parent_Set_from_OIDs(execute_SQL(
    'SELECT parents
    FROM    person_person_
    WHERE   children = ' + self.OID));
end method get_parents

method get_children() : Set(Parents)
  return to_Parent_Set_from_OIDs(execute_SQL(
    'SELECT children
    FROM    person_person_
    WHERE   parents = ' + self.OID));
end method get_children

...

end class Person;
```

Fig. 12 Snippets of the person script class

```
Bag _t1 = new Bag();
for _t3 in <person>.get_parents() do
  _t1.addAll(_t3.get_children());
done
Set _t4 = new Set();
for _t6 in
  _t1.asSet().excluding(<person>) do
  if (_t6.get_gender() = 'female')
    _t4.add(_t6);
done
Sequence _t8 = new SequenceSorted();
for _t7 in _t4 do
    _t8.add(_t7.get_name(), _t7);
done
```

Fig. 13 Script for querying a person's sisters

Now, let's imagine that the person person has *fs* sisters and *fb* brothers with exactly the same parents, and *hs* half sisters and *hb* half brothers. A first query is sent to the database by the application server, because of the get_parents method; the answer is composed of the identifiers of the two parents. Two more SQL queries are sent to each of the two parents to ask for their children; the global answer is composed of $hs + hb + 2(fs + fb)$ person identifiers, received twice for sibling with the same two parents. Then, $fs + hs + fb + hb$ other queries are sent from the get_gender method to determine the gender of each sibling, to make a difference between brothers and sisters; according to the reading cache mechanism, the answer, for each query, is composed of a *recordset* containing values for each one of the three attributes declared in the Person business class, this means $3(fs + hs + fb + hb)$ values. The sort iteration does not send any more SQL query, because the reading cache has already cached the gender value for each sibling. In this example, we end up with $1 + 2 + fs + hs + fb + hb$ SQL queries sent to the database, leading to the transmission from the database server to the application server of $2 + hs + hb + 2(fs + fb) + 3(fs + hs + fb + hb) = 2 + 4(hs + hb) + 5(fs + fb)$ values for this simple example, where attribute values are of a reasonable size.

This has to be compared with the single SQL query shown below in Fig. 14, which would limit the number of transmitted values to $hs + hb$.

As a conclusion, to cope with performances, we had to translate Xion to SQL rather than application server scripts as much as possible. Unfortunately, this is not possible for all Xion statements, like while or return. However, many Xion expressions may be expressed in SQL; since Xion expressions are derived from OCL expressions, this is a task which is pretty similar to translating OCL expressions to SQL queries; a work related to the OCL to SQL compiler of the Dresden OCL Toolkit [37]. The SQL optimization step is a refinement of the *Xion to Intermediate* transformation shown by Fig. 10, to make it generate as much *Database Request* instructions of the *Intermediate Language* as possible, i.e. SQL statements.

The optimization operates on Xion abstract trees decorated with type checking information. The type checking information together with the object-to-relational mapping as computed by the *Object Relational Mapping* transformation is mandatory for the optimization to be aware of tables

```
SELECT DISTINCT N2.children
FROM person_person_ N1,
     person_person_ N2
     LEFT JOIN person A1
        ON N2.children = A1.person_id
WHERE  N2.children <> <person.OID>
  AND N1.parents = N2.parents
  AND  N1.children = <person.OID>
  AND  A1.sex = 'female'
ORDER BY A1.name
```

Fig. 14 SQL for querying a person's sisters

where business information is stored. For instance, when encountering an attribute query, the optimization need to have information on what field in the database corresponds to this attribute in order to build the right SQL statement. The Netsilon tool concentrates on optimizing Xion queries. This is performed in a sequential way, analyzing query expressions, wherever they are used, from left to right. Different optimization elements are specialized in optimizing a precise kind of node of a Xion abstract tree, by creating SQL queries with the *SQL Abstract Syntax* of Fig. 10. Since there are many differences in the SQL dialects of the different RDBMS that Netsilon targets, some optimization elements are platform-dependent, and can process only if the corresponding platform is selected.

7 Modeling Web applications—related work

Several different approaches are undertaken in the modeling community to model Web applications, among which:

- The WebML [11] project has defined four models: 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).
- Conallen [16] 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.
- Schattkowsky et al. [38] 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 code skeletons. We take a more radical position; we want to be able to build PIMs from which totally executable applications can be generated.
- The UWE [10] team has defined a methodology covering the whole life-cycle of Web application development proposing an object-oriented and iterative approach based on the standard UML and the Unified Software Development Process. The main focus of the UWE approach is the systematic design followed by a semi-automatic generation of Web applications. Again, we seek total automation of the application generation.
- OO-H [9] is an object-oriented method which centers on the authoring process and provides views that extend UML to provide a Web Interface model. A code generation process is then able to generate a Web interface from the extended diagrams and their associated tagged values.

The major differences with these related works are:

- The finer granularity of the novel modeling concepts that we have defined for composition and navigation, because we want be able to model and generate any kind of Web applications, with no restriction on their visual appearance.
- The availability of the Xion language which is used to express precisely constraints and actions.
- The ability to generate fully executable (and efficient) Web applications from platform-independent models.

However, it must be stressed that our modeling elements for composition and navigation could be used to implement the navigation models of all these related approaches. This would certainly be a way to raise the productivity of Web developments, because automation would be applied at a higher level in the development process.

8 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.topmodl.org a model-driven Wiki Wiki Web site, and http://www.domaine.fr a dynamic Web application which features domain names selling and account management). We have achieved complete model-driven engineering; we can generate various executable PSMs from the same PIM.

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. As a consequence, the modeling elements that we have introduced for composition and navigation modeling are very general, and we could develop patterns to support the modeling approaches presented in the WebML, UWE or OO-H methods.

We have noticed that almost all the applications that have been developed with Netsilon had to maintain state

information for all active sessions. This is currently implemented in the Business Model with a hidden class whose attributes implement the states. We would like to put explicit support for state machines in our tool, by adding some support for statecharts to model these 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 [31]. Another major point would be to make the PIM to PSM transformations explicit, and therefore allow customization of the code generation phase. Some of us are currently working on a new metamodeling infrastructure [39] that we intend to substitute to our current programmed technology. We believe that this kind of customization ability, by explicit models and metamodels, is essential in the process of adoption of model-driven tools by programmers.

This work may be viewed as an experimentation in platform modeling and PSM generation. It is obviously far from bringing definitive answers to these complex problems. However the presented material may contribute, with many other ongoing research works on similar subjects, to a better understanding of hard related research problems.

References

1. Bezivin, J.: From object composition to model transformation with the MDA. In: Proceedings of TOOLS'2001, pp. 350–354 IEEE Press Tools#39 (2001)
2. Object Management Group, Inc.: MDA Guide 1.0.1. omg/2003-06-01 (June 2003)
3. Gellersen, H.-W., Gaedke, M.: Object-oriented Web application Development. IEEE Internet Computing, pp. 60–68 (1999)
4. El Kaim, W., Burgard, O., Muller P.-A.: MDA Compliant Product Line Methodology, Technology and Tool for Automatic Generation and Deployment of Web Information Systems. Journées du Génie Logiciel, Paris (2001)
5. Nielsen, J.: Hypertext and Hypermedia: The Internet and Beyond. Academic Press (1995)
6. McDonald, A., Welland, R.: Web engineering in practice. In: Proceedings of the fourth WWW10 Workshop on Web Engineering pp. 21–30 (2001)
7. Fraternali, P.: Tools and approaches for developing data-intensive Web applications: a survey. ACM Computing Surveys (3), 227–263 (1999)
8. Mellor, S.J., Clark, A.N., Futagami, T.: Model-driven development. IEEE Software, pp 14–18, (2003)
9. Gomez, J., Cachero, C.: OO-H Method: Extending UML to Model Web Interfaces, pp. 144–173. IDEA Group Publishing (2003)
10. Koch, N., Kraus, A.: The expressive power of UML-based Web Engineering. In: Schwabe, D., Pastor, O., Rossi, G., Olsina, L. (eds.), Proc. 2nd Int. Wsh. Web-Oriented Software Technology (IWOOST'02), CYTED (2002)
11. Ceri, S., Fraternali, P., Bongio, A.: Web modeling language (WebML): a modeling language for designing Web sites. In: Ninth International World Wide Web Conference (2000)
12. Marcos, E., Vela, B., Cavero, J.-M.: A methodological approach for object-relational database design using UML. Soft. Syst. Mode. **2**(1), 59–72 (2003)
13. Roch, M.-C., Muller, P.-A., Thirion, B.: Improved flexibility of a document production line through object-oriented remodeling. In: Second Congress IMACS, Computational Engineering in Systems Applications, Hammamet, CESA'98, vol. III, pp. 152–159. Vabeul-Hammamet Tunisie, (98)

14. Object Management Group, Inc.: Meta Object Facility (MOF), 1.4. formal/02-04-03 (2002)
15. Object Management Group, Inc.: Software Process Engineering Metamodel (SPEM), 1.0. formal/02-11-14 (2002)
16. Conallen, J.: Building Web Applications with UML. The Addison-Wesley Object Technology Series (2000)
17. Desfray, P.: UML Profiles versus Metamodeling Extensions... an Ongoing Debate. Uml In The.Com Enterprise: Modeling CORBA, Components, XML/XMI And Metadata Workshop, Palm Springs, 6–9 (Nov. 2000)
18. Atkinson, C., Kuehne, T., Henderson-Sellers, B.: To meta or not to meta—that is the question. J.Object-Oriented Program. **13**(8), 32–35 (2000)
19. Mellor, S.J., Tockey, S., Arthaud, R., Leblanc, P.: An action language for UML: proposal for a precise execution semantics. UML 98, LNCS1618, pp. 307–318 (1998)
20. AndroMDA: http://www.andromda.org/
21. Silaghi, R., Strohmeier, A.: Parallax, or viewing designs through a prism of middleware platforms. In: Proceedings of the 38th Annual Hawaii International Conference on System Scienes, HICSS, Hilton Waikoloa Village, Big Island of Hawaii, HI, USA, January 3–6, 2005, part of the Mini-track on Adaptive and Evolvsble Software Systems, AESS. IEEE Computer Society (Digital Library), 2005. Also available as Technical Report IC/2004/69, Swiss Federal Institute of Technology in Lausanne, Switzerland, August (2004)
22. Sunyé, G., Pennaneac'h, F., Ho, W.-M., Le, Guennec, A., Jézéquel, J.-M.: Using UML action semantics for executable modeling and beyond. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.), Advanced Information Systems Engineering, CAiSE 2001, vol. 2068 of LNCS, pp. 433–447. Interlaken, Switzerland, Springer (2001)
23. iUML AS: Kennedy Carter, Ltd.: UML ASL Reference Guide, ASL Language Level 2.5, Manual Revision C (2001)
24. ProjTech AL: Project Technology, Inc., Object Action Language TM Manual. vol. 1.4 (2002)
25. Kabira: http://www.kabira.com
26. Telelogic, A.B.: UML 2.0 Action Semantics and Telelogic TAU/Architect and TAU/Developer Action Language. vol. 1 (2004)
27. Object Management Group, Inc.: UML 1.5. formal/03-03-01 (2003)
28. Object Management Group, Inc.: UML 2.0 OCL Final Adopted specification. ptc/03-10-14 (2004)
29. Cook, S., Daniels, J.: Object-Oriented Modelling with Syntropy, 1st edition. Prentice Hall (1994)
30. Spivey, J.: The Z Notation: A Reference Manual, Second edition. Prentice Hall (1992)
31. Richters, M.: A Precise approach to validating UML models and OCL constraints. PhD Thesis, Universität Bremen, Biss Monographs vol. 14 (2002)
32. Object Management Group, Inc.: MOF 2.0 Query/Views/ Transformations RFP. ad/02-04-10 (2002)
33. Abrial, J.-R.: The B-Book. Cambridge University Press (1996)
34. Jackson, D.: Alloy: A Lightweight Object Modelling Notation. Technical Report 797, MIT Laboratory for Computer Science, Cambridge, MA (2000)
35. phpMyAdmin: http://www.phpmyadmin.net/
36. Knuth, D.E.: The Art of Computer Programming, Sorting and Searching, Second Edition. vol. 3. Addison-Wesley (1998)
37. Hussmann, H., Demuth, B., Finger, F: Modular Architecture for a Toolset Supporting OCL. Sci. Comput. Program. (Special issue on UML 2000) **44**(1), 51–69 (2002)
38. Schattkowsky, T., Lohmann, M.: Rapid development of modular dynamic Web sites using UML. In: UML 2002 Conference, LNCS 2460, pp. 336–350 (2002)
39. The TopModL Open Source Initiative: http://www.topmodl.org

Pierre-Alain Muller is an associate professor of Computer Science at the University of Mulhouse, France, he is currently spending two years with INRIA in Rennes, France. His research interest includes software engineering and model-driven engineering; he is leading the TopModL open source initiative. Before joining academia, he has been CEO of ObjeXion Software from 1999 to 2002 and Consultant with Rational Software from 1988 to 1993. He has authored "Instant UML" in 1997 and founded the ⟨⟨UML⟩⟩ series of conferences (with J. Bezivin) in 1998.

Frédéric Fondement is a PhD student at the Software Engineering Laboratory of the Swiss Federal Institute of Technology in Lausanne (EPFL). His area of interest includes model- and language-driven software engineering. In 2002 he was a research engineer at INRIA Rennes, part of the MTL model transformation language development team. After receiving his master degree in computer science in 2000 from the University of Mulhouse, he joined the research and development team of ObjeXion Software, under the direction of Pierre-Alain Muller and Philippe Studer.

Philippe Studer is a research fellow at the University of Mulhouse, France, his area of interest includes compilation, parallel computation and model-driven engineering. From 1999 to 2002, he was CTO of ObjeXion Software, where he led the development of the Netsilon tool. In the early nineties he was involved in research about parallel Postscript interpretation on Transputers applied to large scale textile printing.

Jean Bezivin is professor of Computer Science at the University of Nantes, France, member of the ATLAS research group recently created in Nantes (INRIA & LINA-CNRS) by P. Valduriez. He has been very active in Europe in the Object-Oriented community, starting the ECOOP series of conference (with P. Cointe), the TOOLS series of conferences (with B. Meyer), the OCM meetings (with S. Caussarieu and Y. Gallison) and more recently the ⟨⟨UML⟩⟩ series of conferences (with P.-A. Muller). He also organized several workshops at OOPSLA like in 1995 on "Use Case Technology", in 1998 on Model Engineering with CDIF, on Model Engineering at ECOOP in 2000, etc. His present research interests include model engineering, legacy reverse engineering and more especially model-transformation languages and frameworks.