Chapter 12

AN OVERVIEW OF MODEL-DRIVEN WEB ENGINEERING AND THE MDA

Nathalie Moreno[1], José Raúl Romero[2], and Antonio Vallecillo[1]
[1]*Dept. Lenguajes y Ciencias de la Computación, University of Málaga, Spain.*
[2]*Dept. Informática y Análisis Numérico, University of Córdoba, Spain.*

1. INTRODUCTION

Model-Driven Software Development (MDSD) is becoming a widely accepted approach for developing complex distributed applications. MDSD advocates the use of models as the key artifacts in all phases of development, from system specification and analysis, to design and implementation. Each model usually addresses one concern, independently from the rest of the issues involved in the construction of the system. Thus, the basic functionality of the system can be separated from its final implementation; the business logic can be separated from the underlying platform technology, etc. The transformations between models enable the automated implementation of a system from the different models defined for it.

Web Engineering is a specific domain in which MDSD can be successfully applied. Most of the technology is here to implement systems that exploit the Web paradigm, but the effective design of Web applications is still a concern: the complexity and requirements on Web applications are constantly growing, while the supporting technologies and platforms rapidly evolve.

Existing model-driven Web engineering (MDWE) approaches already provide excellent methodologies and tools for the design and development of most kinds of Web applications. They address different concerns using

separate models (navigation, presentation, data, etc.), and are supported by model compilers that produce most of the application's Web pages and logic based on the models. However, these proposals also present some limitations, especially when it comes to modelling further concerns, such as architectural styles or distribution. Furthermore, current Web systems need to interoperate with other external applications, something that requires their integration with third party Web-services, portals, and also with legacy systems. Finally, many of these Web Engineering proposals do not fully exploit all the potential benefits of MDSD, such as complete platform independence, model transformation and merging, or metamodelling. (Miller and Mukerji, 2003) from the Object Management Group (OMG™) has introduced a new approach for organizing the design of an application into (yet another set of) separate models so portability, interoperability and reusability can be obtained through architectural separation of concerns. MDA covers a wide spectrum of topics and issues (MOF-based metamodels, UML profiles, model transformations, modelling languages and tools, etc.) and also promises the interoperability required between models and tools from separate vendors. On the other camp, Software Factories (Greenfield and Short, 2004) provide effective concepts and resources for the model-based design and development of complex applications, and it is our belief that they can be successfully used for Web Engineering too.

In this chapter we will introduce the main concepts involved in MDWE, and discuss its current strengths, weaknesses and major challenges, especially in the context of the MDA initiative.

2. DOMAIN SPECIFIC MODELLING

Domain-Specific Modelling (DSM) is a way of designing and developing systems that involves the systematic use of Domain Specific Languages (DSLs) to represent the various facets of a system. Such languages tend to support higher-level abstractions than general-purpose modelling languages, and are closer to the problem domain than to the implementation domain. Thus, a DSL follows the domain abstractions and semantics, allowing modellers to perceive themselves as working directly with domain concepts. Furthermore, the rules of the domain can be included into the language as constraints, disallowing the specification of illegal or incorrect models.

DSLs play a cornerstone role in DSM. In general, defining a modelling language involves at least two aspects: the domain concepts and rules (abstract syntax), and the notation used to represent these concepts (concrete syntax—let it be textual or graphical). Each model is written in the language of its metamodel. Thus, a metamodel will describe the concepts of the

language, the relationships between them, and the structuring rules that constraint the model elements and combinations in order to respect the domain rules. We normally say that a model conforms to its metamodel (Bézivin, 2005).

Metamodels are also models, and therefore they need to be written in another language, which is described by its meta-metamodel. This recursive definition normally ends at that level, since meta-metamodels conform to themselves.

A typical example of a metamodel-defined DSL is ATL (Jouault and Kurtev, 2006b), which is a transformation language. A large library of ATL transformations is available from the Eclipse metamodel open source library. The interested reader can consult the work by Jean Bézivin (2005) for a more complete and detailed introduction to these topics.

DSM often also includes the idea of code generation: automating the creation of executable source code directly from the DSM models. Being free from the manual creation and maintenance of source code implies significant improvements in developer productivity, reduction of defects and errors in programs, and a better resulting quality. Moreover, working with models of the problem domain instead of models of the code raises the level of abstraction, hiding unnecessary complexity and implementation-specific details, while putting the emphasis on already familiar terminology.

A DSM environment may be thought of as a metamodelling tool, i.e., a modelling tool used to define a modelling tool or CASE tool. The domain expert only needs to specify the domain specific constructs and rules, and the DSM environment provides a modelling tool tailored for the target domain. The resulting tool may either work within the DSM environment, or less commonly be produced as a separate stand-alone program. Using a DSM environment can significantly lower the cost of obtaining tool support for a DSM language, since a well-designed DSM environment will automate the creation of program parts that are costly to build from scratch, such as domain-specific editors, browsers and components.

Examples of DSM environments include commercial ones such as MetaEdit+; open source environments, such as the Generic Eclipse Modelling System; or academic ones such as the Generic Modelling Environment (GME, http://www.isis.vanderbilt.edu/projects/gme/). The increasing popularity of DSM has led to DSM frameworks being added to existing integrated development environments, such as the Eclipse Modelling Project (EMP) and Microsoft's DSL Tools for Software Factories.

3. MDA

One of the best known MDSD initiatives is called Model-Driven Architecture (MDA®), which is an approach to software development produced and maintained by the OMG, a consortium that produces and maintains computer industry specifications for interoperable enterprise applications. MDA is a registered trademark of the OMG, together with its related acronym, Model-Driven Development (MDD), another OMG trademark.

The goal of MDA is one that is often sought: to separate business and application logic from its underlying execution platform technology so that (1) changes in the underlying platform do not affect existing applications; and (2) business logic can evolve independently from the underlying technology. A tool that implements the MDA concepts will allow developers to produce models of the application and business logic, and also generate code for a target platform by means of transformations.

The major benefit of this approach is that it raises the level of abstraction in software development. Instead of writing platform-specific code in some high-level language, software developers focus on developing models that are specific to the application domain but independent of the platform. In a nutshell, MDA is a broad conceptual framework that describes an overall approach to software development.

MDA is not to be confused with MDSD. MDA is the OMG implementation of MDSD, using the set of tools and standards defined by OMG. These OMG standards include UML® (Unified Modelling Language), MOF (Meta-Object Facility), XMI (XML Metadata Interchange), and MOF/QVT (Query/View/Transformations), among others. All these standards can be obtained from the OMG's Web site (www.omg.org).

3.1 The MDA framework

The MDA framework is basically organized around the so-called Platform Independent Models (PIMs) and Platform Specific Models (PSMs), and on the model transformations between them. The PIM is a specification of a system in terms of domain concepts. These domain concepts exhibit a specified degree of independence of different platforms (e.g. CORBA, .NET, and J2EE). The system can then be compiled using any of those platforms as target by transforming the PIM to a platform specific model (PSM). Thus, the PSM specifies how the system uses a particular type of platform. Finally, the application's code is considered a form of PSM (at the lowest level).

In MDA, a platform is a set of subsystems and technologies that provides a set of functionality through interfaces and specified usage patterns, which

any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented (Miller and Mukerji, 2003). As in MDSD, each model in MDA conforms to a metamodel, which in MDA can be defined using MOF.

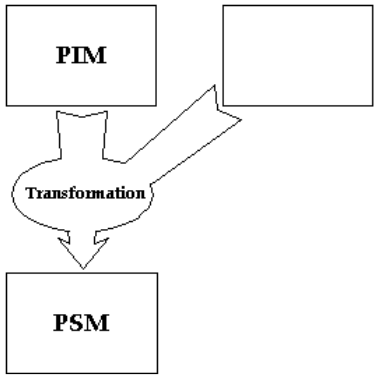

Figure 12.1. The MDA pattern

In addition to models, transformations are also at the heart of MDA. Model transformation is the process of converting one model to another model of the same system (see Figure 12.1). Such transformations can be done following many ways: using types, marks, templates, etc. In MDA, software development becomes an iterative model transformation process: each step transforms one PIM of the system at one level into one PSM at the next level, until a final system implementation is reached, with the particularity that each PSM of a transformation can become the PIM of the next transformation (within another level of abstraction). In this context, the implementation is just another model, which provides all the information necessary to construct the system and to put it into operation.

3.2 OMG approaches for defining DSLs

Both PIMs and PSMs are models, and therefore defined using modelling languages. Although in theory MDA's models can be defined using any modelling language, OMG strongly suggests that models are specified using UML or any other MOF-compliant language (i.e., whose meta-metamodel is MOF). This interest for being MOF and UML-compliant arises from the increasing need to be able to interoperate with other notations and tools, and to exchange data and models, thus facilitating and improving reuse.

OMG defines three main possible approaches for defining domain-specific languages. The first solution is to develop a metamodel that is able to represent the relevant domain concepts. This means creating a new domain language, an alternative to UML, using the MOF metamodelling facilities provided by OMG for defining object-based visual languages (i.e., the same mechanisms that have been used for defining UML and its metamodel). In this way, the syntax and semantics of the elements of the new language are defined to faithfully match the domain's specific characteristics. The problem is that standard UML tools will not be able to deal with such a new language (to edit models that conforms to the metamodel, compile them, etc.). This approach is the one followed by languages such as the CWM (Common Warehouse Metamodel) or the W2000 (Baresi *et al.*, 2006b) notations, since the semantics of some of these languages' constructs do not match the semantics of the corresponding UML model elements.

The second and third solutions are based on extending UML. Extensions of the UML can be either heavyweight or lightweight. The difference between lightweight and heavyweight extensions comes from the way in which they extend the UML metamodel. Heavyweight extensions are based on a modified UML metamodel with the implication that the original semantics of modelling elements is changed and therefore the extension might no longer be compatible with UML tools.

Lightweight extensions are called UML profiles and are based on the extension mechanisms provided by UML (OMG, 2005b; Fuentes and Vallecillo, 2004) (stereotypes, tag definitions, and constraints) for specializing its metaclasses, but without breaking their original semantics. UML profiles may impose new restrictions on the extended metaclasses, but they should respect the UML metamodel, without modifying the original semantics of the UML elements (i.e., the basic features of UML classes, associations, properties, etc., will remain the same, only new constraints can be added to the original elements). Syntactic sugar can also be defined in a profile, in terms of icons and symbols for the newly defined elements. One of the major benefits of profiles is that they can be handled in a natural way by UML tools.

In UML profiles, stereotypes define particularizations of given UML elements, adding them some semantics. For instance, we can define the stereotype <<persistent>> that extends UML classes to represent persistent elements in a particular domain. Tag definitions specify the possible attributes of stereotypes (e.g., the name of the table where the persistent element should be stored). Finally, constraints define the domain rules that the stereotyped UML elements should obey in order to make up correct models (e.g., suppose that we do not want abstract classes to be stereotyped

as persistent). Figure 12.2 graphically shows the UML specification of this example stereotype.

Figure 12.2. An example of a UML 2.0 stereotype specification

Constraints on stereotypes are normally specified using OCL (Object Constraint Language) (OMG, 2006), whose current version (2.0) is fully aligned with UML. Constraints can be either directly attached to the modelling elements (as shown in the Figure), or separately specified, and then be related to the element to which they apply by identifying their context:

```
context Persistent inv:
   self.baseClass.isAbstract = false
```

Perhaps the best known example of customizing UML for a specific domain is SysML, a DSL for systems engineering (www.sysml.org). In addition, there is a whole set of UML profiles that customize UML to deal with the specific concepts required in several relevant application domains (e.g., real-time, business process modelling, finance, etc.) or implementation technologies (such as .NET, J2EE, or CORBA).

Probably, the main advantage of UML profiles is not the extension of the UML metamodel (which is already too large and complex to be used in full), but that they allow “restricting” the set of UML elements that need to be used in a given domain, particularizing the semantics of those elements in order to capture the semantics and structuring rules of the domain-specific elements they represent. It is important to repeat that such a particularization can only be done by refinement, and without changing the original semantics of UML elements.

Finally, meta-transformations which transform back and forth from the profile definition to the metamodel definition can also be specified, as shown in Figure 12.3.

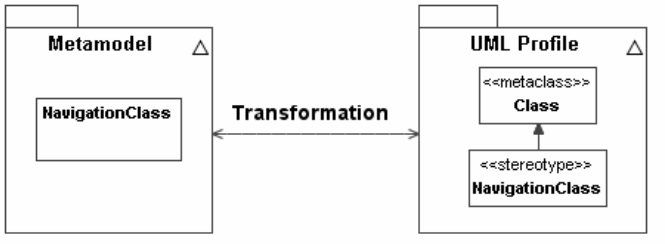

Figure 12.3. Example of transformation between a "profileable" metamodel and a profile

3.3 Model transformations

A model transformation can be viewed as a transformation between two models, which describes how elements in the source model are converted into elements in the target model. This is done by relating the appropriate metamodel elements in the source and target metamodels, and defining constraints and guards on such relations (e.g., the preconditions on the transformation to take place). It is important to notice that model transformations are also models, and therefore they conform to a metamodel that describes the language in which they are expressed.

MDA describes a wide variety of models and transformations between models. Whilst there are many kinds of transformations, they can fit broadly into two main categories:

- Vertical mappings (or *refinements*), which relate system models at different levels of abstraction—such as PIM to PSM mappings, or reverse-engineering mappings. Until now, vertical transformations have in most cases been developed within modelling tools using Web tool-specific proprietary languages. For the same reason that domain know-how should not be tied to a particular platform, it is thus critical that model transformations are not dependent of a given CASE tool.
- Horizontal mappings, which relate or integrate models covering different aspects or domains within a system, but at the same level of abstraction. Horizontal mappings maintain the consistency between levels guarantying that an entity needs to be consistent with what is said about the same entity in any other specification at the same level of abstraction. This includes the consistency of that entity's properties, structure and behaviour.

In MDA, OMG proposes MOF-QVT (Query/View/Transformation) (OMG, 2005a) as the standard language for specifying model

transformations. Many other model transformation languages, like VIATRA by the University of Budapest, ATL by INRIA, RubyTL (Sánchez and García-Molina, 2006) by the University of Murcia, etc., are also available, with different levels of compliance to the QVT standard (Jouault and Kurtev, 2006a). The interested reader can visit the "Model Transformation" Web site (www.model-transformation.org) for a complete listing of model transformation languages and tools.

4. MODEL-DRIVEN WEB ENGINNERING PROPOSALS

As mentioned in the introduction, Web Engineering is a specific domain in which MDSD can be successfully applied, due to its particular characteristics: there is a precise set of concerns that need to be addressed (navigation, presentation, business processes, etc.); the basic kinds of Web applications is well known (Kappel *et al.*, 2006) (document-centric, transactional, workflow-based, collaborative, etc.); and the set of architectural patterns and structural features used in Web systems is reduced and precisely defined. In fact, existing model-based Web Engineering approaches—most of which have been described in this book—already provide excellent methodologies and tools for the design and development of most kinds of Web applications.

These approaches come basically from two main areas. First, a few proposals are based on hypermedia design methods, introducing the required expressiveness and mechanisms to capture relevant Web-specific elements, such as navigation. Prominent examples of these initiatives are HDM (Garzotto *et al.*, 1993), RMM (Frasincar, 2001), WebML (Ceri *et al.*, 2002), W2000 (Baresi *et al.*, 2006b), WSDM (De Troyer and Leune, 1998), Hera (Vdovjak *et al.*, 2003) and Webile (Di Ruscio, 2004), the majority of which are based on the classic E/R model, or on extensions of it. Another group of more recent approaches emerged as extensions of conventional object-oriented development techniques, adapting them to cope with the particular characteristics of Web systems. In this group we can find methods such as EORM (Lange, 1994), OOHDM (Schwabe *et al.*, 1999), UWE (Koch, 2001), OOWS (Pastor *et al.*, 2006), OO-Method (Pastor *et al.*, 2001), OO-H (Gómez and Cachero, 2003) or MIDAS (De Castro *et al.*, 2006).

These proposals are model-driven because they address the different concerns involved in the design and development of a Web application using separate models (such as content, navigation and presentation), and then are supported by model compilers that produce most of the application's Web pages and logic right from the original models. Furthermore, most of them

count with development processes that support their notations and tools, and have been successfully used in commercial environments for building many different kinds of Web systems. And although all methodologies adopt different notations and propose their own constructs, they all share a common ground of concepts—and thus they might be considered as somehow based on a common metamodel, as suggested by N. Koch and A. Kraus (Koch and Kraus, 2003).

However, as the complexity of Web applications grows (to be able to deliver, e.g., large e-commerce, e-learning, or e-government applications), and new requirements are imposed on Web systems, most of these proposals are showing some limitations:

- They are usually tied to particular architectural styles and technologies, i.e., do not allow the parameterizable construction of Web applications using different platform technologies and architectural styles—they typically build client-server applications only, and based on very specific platform technologies (PHP, ASP, EJB or JSP). The problem is that these architectural styles and target technologies are no longer relevant when, for example, mobility and nomadic features are required for some types of Web applications.
- Most of these proposals were originally conceived to deal with particular kinds of Web applications (such as Web Information Systems, Hypermedia Applications, or Adaptive Web Applications), so they deal with a fixed set of common concerns (navigation, presentation, etc.). Therefore they are very good at modelling certain aspects, but very weak at modelling others. In addition, they are difficult to extend to model further aspects (such as internal processes, distribution, and some other extra-functional concerns) in a natural, modular and independent way.

Finally, Web applications currently need to interoperate with other external systems. This requires their integration with third party Web-services, portals, and also with legacy systems—meaning, among other things, that their processes, choreography, and part of their business logic, must be explicitly available for integration with these external systems (Moreno and Vallecillo, 2005a). Not all MDWE proposals address this issue at the model level; the integration is mostly achieved at the implementation level.

Solving all these limitations is not a trivial task. We are currently observing how some Web Engineering proposals are evolving to cope with some of these issues. For instance, some of them are developing extensions to address more and more aspects. Examples include UWE and OO-H, which have incorporated a process model into their original approaches (Koch *et al.*, 2004), and are working to deal with the architectural style of

the final application, too (Cáceres *et al.*, 2006). WebML has also evolved to be able to deal with legacy systems, and for context-awareness (Ceri *et al.*, 2007). The problem with these incremental extensions is that, unless their efforts to include new concerns are made in a very well organized and interoperable manner, we may end up with proposals that have grown by adding too many new features in an unnatural and artificial way, and therefore may become too complex and brittle.

Another problem that some of these proposals are also facing is their use proprietary notations and tools. This forces customers and developers to buy and use "yet-another" modelling tool (with the learning costs and efforts involved in the process) if they want to take advantage of them. Even worse, these proprietary tools do not interoperate with the rest of the tools being used by the customer, which forces him/her to work with a whole set of isolated development environments, each one different (and incompatible) with the rest—something that the customer is not going to tolerate.

Thus, we are witnessing how the Web Engineering community considers the use of standard UML notation, techniques and supporting tools for modelling Web systems, including the adaptation of their own modelling languages, representation diagrams and development processes to UML. There is a need to be able to be compatible and interoperate with other notations and tools, and to seamlessly exchange data and models with them. This is the case for instance of WebML, which is defining UML-based representations of its modelling language so that the WebML notation and its development process can be smoothly integrated into standard UML development environments (Moreno *et al.*, 2006; Schauerhuber *et al.*, 2006).

The advent of the Model Driven Architecture (MDA) initiative may also bring significant benefits here, and also help to address most of the limitations cited above in a natural way. As mentioned in the preceding section, MDA provides an approach for organizing the design of an application into separate models so that portability, interoperability and reusability can be achieved through architectural separation of concerns. In addition, the new modelling notation UML 2.0 incorporates a whole new set of diagrams and concepts which are more appropriate for modelling the specific structure and behaviour of software systems, and in particular of Web applications (e.g., the new structuring mechanisms, or the improved specification and semantics of state machines and activities).

Of course, the use of UML and MDA for Model-Driven Web Engineering is not free from problems. As any other initiative, it brings along both benefits and drawbacks, and also counts with both supporters and detractors. The next two sections are dedicated to explain these ideas in detail.

5. MDA-BASED WEB ENGINEERING

MDA provides several interesting opportunities to improve current Web Engineering approaches, helping them to overcome some of the limitations cited above.

5.1 Becoming UML and MOF-compliant

As previously mentioned, there is an increasing need to be able to interoperate and be compatible with other notations and tools, and to integrate with already existing modelling environments—in particular with the UML tools that nowadays are commonplace in many customer settings. Of course, there are other DSM environments already coming—some of them probably much better than those supporting the UML notation—but the problem is that they have not reached the level of acceptance and are not as spread as UML modelling tools are today. And we are faced with the need to be able to offer a solution to our customers today.

In this sense, a very promising approach is the definition of UML profiles for representing proprietary Web Engineering modelling languages. This is the case of WebML, which has recently defined a metamodel and a UML Profile (Moreno *et al.*, 2006; Schauerhuber *et al.*, 2006) for its notation. This allows the WebML language and its development process (supported by the WebRatio tool) to be smoothly integrated with standard UML development environments.

In addition, counting on a metamodel for WebML will allow its integration with other MDA tools as soon as they are available (editors, validators, metric evaluators,…) and also with other MDSD approaches and tools (using model transformations that allow the conversion of MOF-metamodels to other metamodelling approaches, such as KM3 or Ecore).

5.2 Organizing models according to the MDA principles

We are also witnessing how other approaches that were originally UML-based are making use of the new MDA principles to reorganize their models in a modular manner, in such a way that each model focuses on one specific concern, and then formulating their development processes in terms of model transformations and model merges.

Probably the most representative example is UWE, which has successfully re-structured its original set of models (which represented the different concerns involved in the design and development of a Web application) in terms of metamodels, and the UWE development process in terms of transformations between them (Koch, 2006; Kraus, 2007). This has

significantly enhanced the original proposal with better modularity, expressiveness and re-use. Furthermore, the use of specification techniques for the transformations will allow UWE to redefine and improve many of the aspects of its development process, especially those that were originally hard-coded in the UWE supporting CASE tool, in order to benefit from model transformation rules defined at a higher abstraction level, e.g., using graph transformations or transformation languages.

Another interesting outcome of the work done by the UWE group when adopting the MDA principles into their proposal is the analysis of the models (and model transformations) that comprise the MDSD process for Web applications, focusing on the classification of the model transformations in terms of type, complexity, number of source models, involvement of marking models, implementation techniques and execution type (Koch, 2006). This analysis could be very useful to other model-based Web Engineering methods if they decide to reformulate their proposals in terms of independent models and transformations between them. Other proposals, such as MIDAS, have also started to adopt such an approach by specifying the development process of Web Information Systems in terms of (meta)models and transformations between them (Cáceres *et al.*, 2006).

5.3 Adding new concerns

That reformulation of model-based Web Engineering proposals is also proving other benefits, such as the modular addition of further aspects into their designs. Most of these concerns were not contemplated originally, and integrating them was difficult because of the (usually ad-hoc) internal structure of their supporting processes and tools.

One representative example is OO-H, whose authors realized that they had to be able to deliver Web applications with different software architectures and to different platforms, depending on the customers' specific requirements—in this case the customers were the ones demanding such features. The OO-H team managed to successfully reformulate part of their internal structure and methods, making the representation of the software architecture of the system a separate concern that could be captured as a separate model, and then merged (using QVT transformations) with the rest of the models of the system (such as navigation, presentation, etc.) (Meliá and Gómez, 2006).

UWE and OO-H have also investigated the explicit representation of the business processes of a Web application, as separate models (Koch *et al.*, 2004). Their joint findings are very encouraging, because they managed to define a common way for modelling them for both proposals. This shows that re-use of metamodels across Web Engineering proposals is feasible.

Finally, UWE has also showed recently how other concerns, such as the user requirements (Koch *et al.*, 2006), can be expressed as UML models and connected to the approach. This is one of the benefits they have obtained once they have fully re-organized their proposal as a set of separate models, related through model transformations (Kraus, 2007).

All these findings support the thesis that a common metamodel is possible for Web Engineering, as originally proposed by Koch and Kraus (2003). Furthermore, in the next section we will see how the existence of a common metamodel could allow the definition of a framework for building Web applications, which in the context of the MDA would also enable the exchange of models and tools between MDWE proposals.

6. WEI: A MODEL-BASED FRAMEWORK FOR BUILDING WEB APPLICATIONS

In this section we shall identify a general set of common concerns involved in the development of Web applications, and present a Model-Driven Web Architectural Framework (WEI) for organizing and relating the different models that represent these concerns. Each WEI model focuses on one particular concern (navigation, presentation, architectural style, distribution) and at different levels of abstraction (platform-independent, platform-specific). The set of metamodels that define such models can be considered as a common metamodel for MDWE.

WEI is also supported by a development methodology for building Web applications, which conforms to the MDA principles—in the sense that it is defined in terms of models and the relationships between them, so transformations can be easily formalized amongst the models until the final implementation is reached.

6.1 Identifying reference models for Web applications

In general, the kinds of concerns involved in the development of a Web application will directly depend on the type of Web application being designed and also on the project requirements. Web applications have already been classified by complexity and development history (Kappel *et al.*, 2006):

1. *Document centric Web sites*, which are hierarchical collections of static HTML documents (basically, plain text and images) that offer read-only information based on a set of structured content, navigation patterns and presentation characteristics designed and stored a-priori. The simplicity

and stability of these systems limits the scope of Web modelling to three models: a **user interface structure** model that deals with the content of the information delivered to the client, a **navigation** model that points out the network of paths within the Web application and a **presentation** model that refers to the visual elements that comprise the Web pages.

2. *Transactional Web applications*, which incorporate support for persistent data store, information location, concurrency control, failure and configuration management. In addition to the navigation aspects of any hypermedia application, development of transactional Web applications implies the need for an effective **information structure** model, which is capable of capturing the processes of inserting, updating and deleting data, and also a **distribution** model which enables the establishment of alternatives for carrying out transactions. A clearer separation between data design, behavioural aspects of the application, and from the user interface concerns is required.
3. *Interactive Web applications*, which are browser-based applications that allow dynamic content of Web pages, hence providing users with personalized information. This feature requires a **process** model that describes how business classes manage the information stored (i.e., the elements of the information structure model), and also requires that the navigation and presentation models are parameterizable to provide tailor-made information to individual users according to their preferences, goals and knowledge. Furthermore, this type of application puts emphasis on modelling not only the information structure itself and its future consumers (i.e. the **users** model), but also the relationships or bridges between the information structure model, navigation model and **business** model.
4. *Workflow-based Web applications*, which provide support for modelling structured business processes, activity flows, business rules, interactions among actors, roles, and a high-performance infrastructure for data storage (content management). Information is needed not only for the system actors but also for its processes. For this kind of Web applications, as a minimum the following models are required: a user interface structure model, a navigation model, a presentation model, an information structure model, a business model (i.e., the description of how functionality is encapsulated into business components and services), a process model (with a description of the behaviour of the internal processes) and a **software architecture** model identifying the subsystems, components and connectors (software and hardware) the application should have.
5. *Collaborative Web applications*, which are those executed by different groups of users that access Web resources to accomplish a specific task. They entail a modelling decomposition of the Web application design into **views** or workspaces based on different **user roles**. For each group

of users, the functional requirements, task and activities to be performed must be specified. These issues involve **modularity** and **distribution** requirements on the process model. Finally, the information assets to be manipulated by views must be also modelled.

6. *Portal-oriented Web applications*, which integrate resources (data, applications, and services) from different sources in a single point. From an end-user perspective, a portal is a Web site with pages that are organized by some form of navigation. Pages can either display static HTML content or complex Web Services. Personalization, behaviour tracking of users as well as message flows in Web service collaborations are extremely relevant in portal-oriented Web applications. Therefore, a **choreography** model needs to express the expected behaviour of both the system processes and the **external services** in order to check their compatibility and interoperability to compose them to build the portal aggregated.
7. *Ubiquitous Web applications*, which need to be accessible at any time, from anywhere, and in any media, i.e., they must run on a variety of platforms, including mobile phones, personal digital assistants (PDAs), desktop computers, etc. This implies that their **presentation** and **navigational** models should be **adaptable** not only to different kinds of users, but also to different kinds of platforms and contexts. Consequently, this kind of application requires modelling the separation between **platform-independent** and **platform-specific** concerns.

Based on the set of concerns identified above, each one represented by one model, we have built an architectural framework for model-driven Web application development (WEI). Its basic structure is depicted in Figure 12.4. It is organized in three main layers (User Interface, Business Logic and Data), each one corresponding to a viewpoint. In turn, each layer is composed of a set of models, which specify the entities relevant to each concern.

Far from being “yet another Web methodology”, the aims of WEI can be summarized as follows:

(a) to be able to represent, in terms of models and relationships between them, the concerns required for designing and developing Web applications—following an architectural separation of concerns as prescribed by MDA;
(b) to integrate and harmonize the models and practices proposed by existing approaches, addressing their concerns;
(c) to be extensible so new concerns could be easily added;
(d) to provide as a common framework (and metamodel) in which current proposals could be integrated and formulated in terms of the MDA

principles, hence allowing them to smoothly interoperate (by defining, e.g., interoperability bridges between compatible models coming from different proposals, whenever this is possible) and complement each other, share tools, etc.

At a high architectural design level, the whole WEI concept space is captured by thirteen metamodels, organized in three main packages as shown in Figure 12.4. It is important to note that the models that comprise the framework have not been arbitrarily chosen, but based on the concerns covered by existing Web Engineering proposals (see also Table 12.1 later on) and our previous experience with the development of large distributed applications.

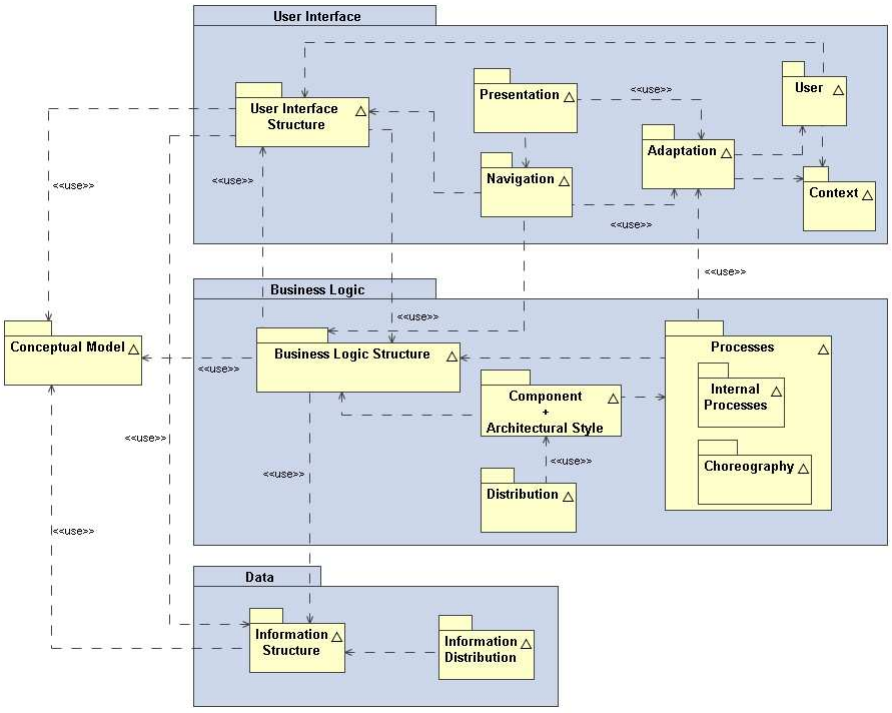

Figure 12.4. Models representing the concerns involved in the development of Web applications

At the bottom level, the **Data Structure** package describes the organization of the information managed by the application (by means of, e.g., a database system) and provides a mechanism for storing it persistently. Information is depicted in terms of the data elements that constitute its

information base and the semantic relationships between these elements. This level is organized in two models:

(i) The **Information Structure** model deals with the information that has to be made persistent, i.e., stored in a database.
(ii) The **Information Distribution** model describes the distribution and replication of the data being modelled, since information can be fragmented in *Nodes* or replicated in different *Locations*.

Then, the **User Interface** focuses on the facilities provided to the end user for accessing and navigating through the information managed by the application, and how this information is presented depending on the context and the user profile. The User interface level is responsible for accepting persistent, processed or structured data from the Process and Data viewpoints, in order to interact with the end user and deliver the application contents in a suitable format. Originally, Web applications were specifically conceived to deal mainly with navigation and presentation concerns, but currently they also need to address other relevant issues:

(i) The **User Interface Structure** model encapsulates the information that the rest of the models at this level have about the information handled by the system (i.e., it is the *view* of such information from this viewpoint).
(ii) The **Navigation** model represents the application navigational requirements in terms of Access Structures that can be accessed via Navigational Links.
(iii) Navigational objects are not directly perceived by the user; rather, they are accessed via the **Presentation** model. This model captures the presentational requirements in terms of a set of *PresentationUnits*.
(iv) The **User** model describes and manages the user characteristics with the purpose of adapting the content and the presentation to the users' needs and preferences.
(v) The **Context** model deals with *Device*, *Network*, *Location* and *Time* aspects, and describes the environment of the application. These are needed to determine how to achieve the required customization.
(vi) The **Adaptation** model captures context features and user preferences to obtain the appropriate Web content characteristics (e.g., the number of embedded objects in a Web page, the dimension of the base-Web page without components, or the total dimension of the embedded components). Adaptation policies are usually specified in terms of ECA rules.

Finally, the **Business Logic** package encapsulates the business logic of the application, i.e., how the information is processed, and how the application interacts with other computerized systems.

(i) The **Business Logic Structure** model describes the major classes or component types representing services in the system (*BusinessProcessInformation*), their attributes (*Attributes*), the signature of their operations (*Signature*), and the relationships between them (*Association*). The design of the Structure model is driven by the needs of the processes that implement the business logic of the system, taking into account the tasks that users can perform.

(ii) The **Internal Processes** model specifies the precise behaviour of every *BusinessProcessInformation* or component as well as the set of activities that are executed in order to achieve a business objective. For a complete description of a business process, apart from the Structure model, we need information related to the *Activities* carried out by the *BusinessProcessInformation*, expressing their behaviour and the *Flows* that pass around objects or data.

(iii) The **Choreography** model defines the valid sequences of messages and interactions that the different objects of the system may exchange. The choreography may be individually oriented, specifying the contract a component exhibits to other components (*PartialChoreography*) or, it may be globally oriented, specifying the flow of messages within a global composition (*GlobalChoreography*).

(iv) The **Distribution** model describes how its basic entities, the *Nodes*, are connected by means of point to point connections or *Links*. While the Information Distribution model of the Data layer specifies the distribution of the data, this model describes the distribution of the processes that achieve the business logic of the system.

(v) The **Component+Architectural Style** model defines the fundamental organization of a system in terms of its components, their relationships, and the principles guiding its design and evolution, i.e., how functionality is encapsulated into business components and services.

The emphasis in each of these levels will depend on the kind of Web application being modelled (data-intensive, user-interface oriented, etc.)

A central model of the WEI framework is the **Conceptual Model**, which can be used for both specifying the basic structure and contents on the Web application (so the rest of the "views" can relate to the elements of that model), and also to maintain the consistency of the model specifications establishing how the different viewpoints merge and complement each other.

Please note that, in addition to the models, the framework predefines some dependencies between the models which determine those cases in

which the definition of a model requires the previous specification of some other models. At a different level, the dependencies may also imply how the framework instantiation process should be carried out. Furthermore, these dependencies also specify correspondences between the elements from different models of the framework, especially when they may have been independently developed by different parties, or when they represent the system from different viewpoints, and therefore the same element is specified in different ways in different models (each one offering a partial view of the whole). In these cases, correspondences between model elements may be also subject to certain consistency rules, which check that the views do not impose contradictory requirements on the elements they share.

6.2 Modelling these concerns

In order to formally define the framework, we have built a MOF metamodel for each model, which describes its entities and their relationships (http://www.lcc.uma.es/~nathalie/WEI/). MOF was selected as metamodelling language because our interest in being MDA-compliant. Other alternatives were of course possible (using, e.g., KM3 or Ecore) but it was important for us to try to use OMG's notations and tools, to exercise the MDA approach. MagicDraw was selected as modelling tool. The selection of a UML tool is something really important, because they do not interoperate well and therefore the tool you use may greatly condition your project.

But the metamodels are just one part of the puzzle. Unlike other approaches, OMG does not provide a solution for directly building correct models from metamodels. Instead, you have to define your own DSL associated to these metamodels. In our case we defined light-weight extensions of UML, i.e., UML profiles, for representing these models (Moreno *et al.*, 2005). As an example of it, Figure 12.5 shows the profile for the WEI presentation metamodel.

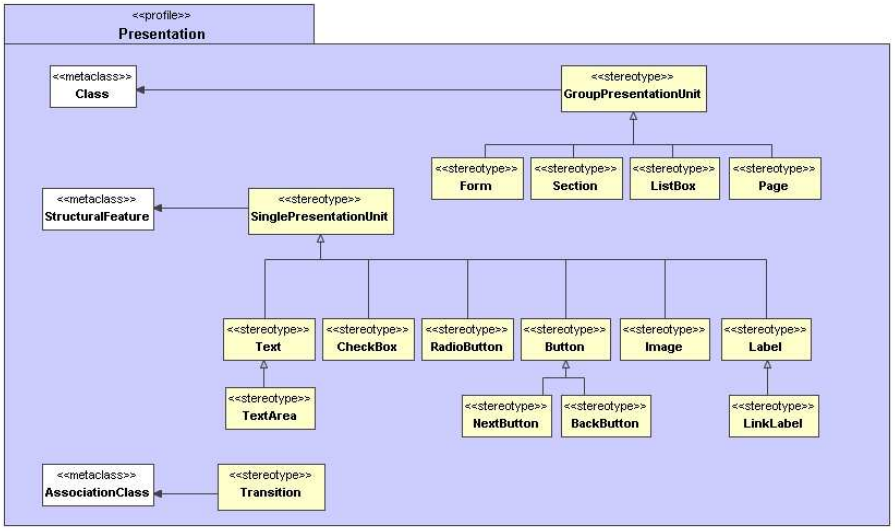

Figure 12.5. The WEI Presentation profile

6.3 How the framework is used

WEI can be instantiated both to build Web applications from scratch, and to build Web applications based on existing models (including those defined using other methodologies, e.g., UWE, WebML or OOH).

6.3.1 Building applications from scratch with WEI

The straightforward application of the framework in the context of MDA to develop a Web system from scratch has already been documented in detail (Moreno *et al.*, 2005a; Moreno *et al.*, 2005b; Moreno and Vallecillo, 2005c), and successfully applied to define and implement several kinds of Web applications such as the *Conference Review System* or the *Travel Agency Application*.

As a brief summary, the WEI methodology process involves the definition of at least three PIMs, each one corresponding to a viewpoint as illustrated in Figure 12.6(b). Each PIM is composed of the set of models described in the previous section, and is developed following the process depicted in Figure 12.6(a).

Once we have the three top-level PIMs are appropriately defined, we need to mark them using the appropriate profile(s) for the target platform(s) and technologies. Once marked, we need to follow the MDA transformation process from PIMs to PSMs, applying a set of mapping rules (one for each

mark and for each marked element). The result of the application of such mapping rules are a set of UML models of the application according to the target technologies (e.g. Java, JSP, Oracle, etc.). Finally, the PSMs are translated to code applying a transformation process again.

It is important to note that *bridges* should be specified between the three PIMs and between their corresponding PSMs, and for which transformations are also required. Bridges are the key elements to maintain consistency between the different models at the same level of abstraction, and to be able to provide links between them. A very interesting work by the group of Alfonso Pierantonio at the University of L'Aquila (Chicchetti *et al.*, 2006) shows how model weaving can be effectively used to specify and implement such bridges, being able to connect the different artifacts and models produced during the development of Web applications—in particular the models describing the data, navigation, and presentation aspects, whose connections are usually defined in an ad-hoc manner, and their consistency manually maintained. Although their work is carried out using non-OMG notations and standards, it can be easily ported to the MDA context, using MOF metamodels and QVT transformations for establishing correspondences between elements from different views.

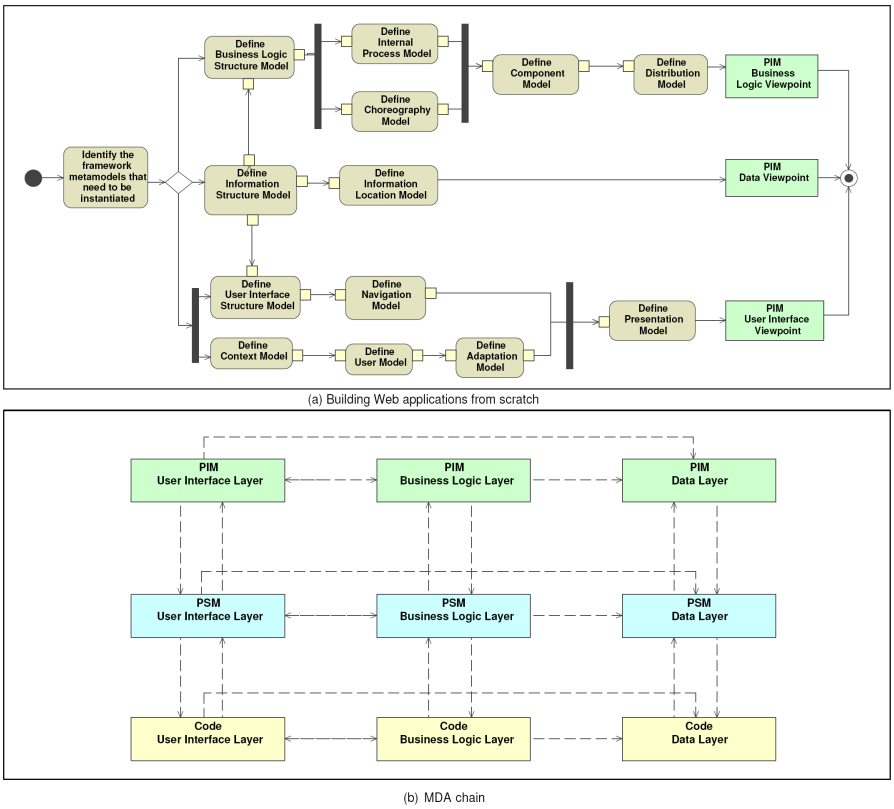

Figure 12.6. The WEI process

6.3.2 Designing Web applications by reusing models from other methodologies

One of the major advantages of our proposal is its ability to design and implement applications reusing both models and tools (e.g., model compilers) defined by other Web methodologies. Thus, a Web application developer could use, for instance, UWE or OO-H for designing the models of the User Interface layer, and WebML for designing the Data layer or vice-versa. Furthermore, models could be already defined for other applications and reused here for building fast prototypes.

Reusing models conforming to other Web methodologies requires the definition of interoperability bridges between “compatible” models coming from different methodologies and the appropriate models of our framework. Usually, the source and target entities defined in different approaches do not differ much. In addition, neither the models nor the entities described in our

framework were arbitrarily chosen: instead, they try to generalize the entities and models defined by most Web Engineering proposals (see Table 12.1). Thus, the interoperability bridges between models from different proposals are a priori feasible and even quite straightforward using WEI as a reference framework.

Layer	Model	OOHDM	W2000	UWE	WebML	WSDSM	OOWS	OOH
User Inter-face	Structure	√	~	√	~	~	√	√
	User	~		√	√	√	√	
	Context	~		√	√			
	Adaptation			√	√		√	
	Navigation	√	√	√	√	√	√	√
	Present.	√	√	√	√	√	√	√
Busi-ness Logic	Structure		~	~		√	√	~
	Processes		√	√			√	√
	Choreogr.						√	
	Architect.			√	√			√
	Distribution							
Data	Inf.Struct.	√	√	√	√	√		√
	Inf.Distrib.							

Table 12.1. Concerns and models covered by current Web Engineering proposals

There are however some issues that need to be addressed, which are similar to the traditional problems that appear when integrating models that represent different views of the same system. In the first place we may find models using different names to refer to the same elements. Second, we may find that one model may assume the existence of other models that either provide some services (e.g., the precise behaviour that needs to be executed when a navigation link is traversed) or represent external systems or legacy applications that our Web system should be able to work with (by, e.g., exchanging data o invoking services). Third, the majority of Web Engineering proposals apply (almost the same) separation of concerns but the problem is that their levels of abstraction and granularity do not always coincide. Fourth, some of the models that we want to reuse may deal with more than one of our framework concerns. And finally, we may find some aspects and concerns that have not been modelled, because they are implicitly assumed in the proposals' models (the most typical example is behaviour).

The way in which we address the first four issues is by specifying bridges (either correspondences or transformations) between the elements living in different models. Such bridges have been defined using QVT relations. The last issue, i.e., the lack of models for representing some concerns, needs to

be addressed by the explicit specification of such elements, in order to supply the "missing" information. This case currently happens when models to be re-used come from methodologies which do not have all their information explicitly modelled, but hard-wired into their supporting CASE tools. Thus, the models to be re-used assume some information and semantics which is not available if we try to use them in a different environment. This problem is alleviated by the explicit representation of all concerns in the WEI framework, because all the information has to be supplied there.

7. ISSUES AND CHALLENGES FOR MDWE AND MDA

So far we have discussed how MDA and its related concepts and mechanisms can help in the effective design and development of Web applications. This section describes the major challenges faced by the introduction of MDA in the Web Engineering domain.

7.1 Maturity of MDA standards and tools

One of the major problems that any person approaching MDA discovers is the lack of maturity of the current standards and tools. For example, some standards considered key to MDA are not currently supported by tools, and some others have not even been finalized. Probably the most representative example is QVT, for which there is not a complete implementation available as of today. This is really frustrating, and needs to be urgently addressed in order to avoid the dissatisfaction it produces to potential users.

7.2 Lack of interoperability between UML tools

Despite the interoperability goals of the OMG, current UML modelling tools cannot properly interoperate, and exchanging models and diagrams between them is almost impossible. XMI is supposed to provide the solution to this problem, but most UML tool vendors fail to generate fully XMI-compliant specifications of the models they produce. What we currently see is that most vendors add proprietary extensions to the XMI tags, which cannot be understood by other tools. This is another sign of the current immature status of the MDA initiative, which we expect can be resolved soon (otherwise the vendors may kill this opportunity with their incompatibilities).

7.3 Need to improve the support for DSLs

As mentioned above, UML profiles are a very interesting option to define DSLs, not only because they are relatively simple to define, but also because once defined they can be (in theory) used by any UML tool to produce models that conform to that profile.

The current situation is not so bright, however. Actually, most UML tools provide support for defining UML profiles (in terms of their corresponding stereotypes, tag definitions and constraints), but fail to be able to guarantee the constraints on the models because they do not support OCL checks. Therefore, you can specify a UML profile that represents a given application domain (that is, a DSL for that domain), but then there is no way of checking that the models that users produce respect the structuring rules of that DSL, i.e., users can easily create wrong models. It is similar to defining a language but without providing a compiler that could check the grammar of the programs produced.

Another improvement that is also required is a better support for relating MOF metamodels with profiles, i.e., to map the metamodel of a DSL to its corresponding profile, as suggested in Section 3.2. This would allow importing metamodels from other sources, and then being able to use standard UML tools to easily draw models that conform to them. There are some academic proposals in this respect (Abouzahra *et al.*, 2005), although this kind of mechanisms should be implemented in most UML tools as part of their profiling facilities.

7.4 The complexity of UML

The size and technical complexity of UML has been held responsible for hampering its wide adoption in many industrial environments. UML is a general-purpose modelling language for software-intensive systems, which is designed to support many kinds of applications. Consequently, in contrast to specific DSM languages, UML is used for a wide variety of purposes across a broad range of domains. Thus, it counts with many modelling elements and diagrams, and even provides support to cope with different semantic variants, through the *semantic variation points* defined for some of its elements. This mechanism increases the potential adoption of UML in many different kinds of environments, but at the high cost of increasing its complexity and introducing lack of focus and precision (“maximizing reuse minimizes use”). This kind of mechanisms has also a strong impact on the learning curve of UML, and on the efforts required by system modellers to master and effectively use the UML notation.

7.5 The ways in which modellers work

Many of today's modellers are still casual in their approach; MDSD (and in particular MDA) requires increased rigor to produce models which are amenable to automatic generation of code. This means that users need to be very precise when designing their models—which in MDA implies plenty of training in UML modelling.

Please notice that this issue and the previous one could be greatly alleviated by the use of UML profiles which restricted the set of UML elements that can be used to model a domain-specific application, and only allowed users to draw correct models with regard to the DSL metamodel (i.e., the profile). This is why very compact, precise and specific UML-based DSLs, with a reduced number of elements and strong structuring rules are being perceived as key factor to the success of MDSD (Bézivin *et al.*, 2005). However, current UML tools do not provide complete support for UML profiles (including the validation of their OCL constraints) as mentioned above. In addition, the use that average modellers make of UML stereotypes and profiles is not always correct, especially because this extension mechanism is not as simple as it might seem at first sight. Different studies have tried to analyse the way in which stereotypes are currently used, and the most common mistakes made by modellers when defining and using them (Atkinson *et al.*, 2003; Henderson-Sellers and González-Pérez, 2006).

Another tendency that we also perceive in normal modellers is the use of DSLs that support agile methodologies and rapid prototyping for designing and developing Web applications. For instance, the use of Ruby is gaining acceptance in many areas (Schwabe, 2006), and the experiences show that the increase in development performance and reduction in costs might be worth its use, especially when combined with frameworks such as Rails (Thomas *et al.*, 2006).

7.6 MDA is not just about modelling

It is unrealistic to expect 100% code generation for every computing problem, and no vendor today can realistically offer a complete MDA solution. Thus, if you expect too much of MDA, it will fail. What MDA offers is just a way of approaching the design and development of systems, using a set of standard notations and tools to achieve interoperability and reuse across vendors, and platform independence. But to realize the full benefits of MDA, organizations should not just introduce some modelling practices in their development processes; they must support the full software lifecycle development process, from analysis and requirements management

through design, development, implementation, deployment, and maintenance. Otherwise the full advantages of MDA will be lost.

7.7 Modelling further concerns

Finally, and specially in the case of more data-intensive Web applications (usually called Web-based Information Systems) we see a trend towards the incorporation of emerging initiatives like the Semantic Web, with supporting technologies such as (Semantic) Web Services, and (Semantic) Web Rule Languages, which aim at fostering application interoperability. Semantic Web languages (like RDF(S) or OWL) facilitate the description of models for such domains. However, the integration of all these models with the rest of the model-based Web Engineering approaches is still unresolved. This is not only a problem for MDA, but for any MDSD approach.

Further concerns, such as user requirements, as well as the role that the Computation Independent Model (CIM) defined by MDA plays in MDWE, need to be investigated too.

8. CONCLUSIONS

In this chapter we have presented an overview of the current state of Model-Driven Software Development, and of Model-Driven Web Engineering in particular, especially in the context of MDA. We have analyzed which are the key concepts and mechanisms that these approaches provide, and how the development of Web systems can benefit from them. Apart from introducing the advantages and opportunities that MDA can bring to MDWE, we have also discussed the current problems and threats that MDA faces for its successful adoption in industrial settings. Addressing and resolving them properly is possibly the major challenge for MDA nowadays.

In summary, we have seen that there is a real need to integrate with UML environments, which are the ones currently demanded in many customer settings nowadays, and that MDA can help re-formulating and re-organizing current Web Engineering proposals in terms of models and transformations between them. MDWE can significantly benefit from the facts that each model can address a concern, that these concerns can be explicitly represented, and that they can be specified in a platform independent manner—hence achieving the modularity, portability, reusability and interoperability required for any competitive Web Engineering proposal. MDWE solutions cannot survive isolated any longer, they need to interoperate among themselves and be integrated into the customers'

development environments. And these are precisely the issues that MDA can help them address in a very successful way.

9. ACKNOWLEDGEMENTS

We would like to acknowledge the work of many MDSD, MDA and MDWE experts who have been involved in investigating and addressing the problems of model-Web Web Engineering. Although the views in this paper are the authors' solely responsibility, they could not have been formulated without the many long and clarifying discussions with these experts. In particular we would like to thank Nora Koch, Jaime Gómez, Vicente Pelechano, Piero Fraternali, Oscar Pastor, Daniel Schwabe, Gustavo Rossi, Geert-Jan Houben, Joaquin Miller, Jean Bézivin, Alfonso Pierantonio, Bryan Wood, and many others too numerous to be named here. We would also like to thank both the organizers and the participants of the past editions of the Model-driven Web Engineering (MDWE) workshop at the last ICWE conferences, where some of the issues presented here were originally raised and discussed.

This work has been supported by Spanish Projects TIN2005-25886-E and TIN2005-09405-C02-01.

10. REFERENCES

Abouzahra, A., Bézivin, J., Del Fabro, M.D. and Jouault, F. 2005. A Practical Approach to Bridging Domain Specific Languages with UML profiles, in: Proc. of the Best Practices for Model Driven Software Development at OOPSLA'05, San Diego, California, USA.

Atkinson, C., Kühne, T. And Henderson-Sellers, B. 2003. Systematic stereotype usage. Software and Systems Modelling, 2(3) pp. 153–163.

Baresi, L., Colazzo, S., Mainetti, L. and Morasca, S. 2006a. Model–Based Web Application Development, in: Web Engineering, Germany, Springer-Verlag, pp. 303–334.

Baresi, L., Colazzo, S., Mainetti, L. and Morasca, S. 2006b. W2000: A Modelling Notation For Complex Web Applications, in: Web Engineering: Theory and Practice of Metrics and Measurement for Web Development, Springer-Verlag, pp. 335–408.

Bézivin, J., Jouault, F., Rosenthal, P. and Valduriez, P. 2005. Modelling in the Large and Modelling in the Small, in: Proc. of the European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004, Springer-Verlag, LNCS 3599, pp. 33–46.

Bézivin, J. 2005. On the Unification Power of Models. Software and Systems Modelling (SoSym), Springer Verlag, 4(2) pp.171–188.

Cáceres, P., De Castro, V., Vara, J.M. and Marcos, E. 2006. Model transformations for hypertext modelling on web information systems, in: Proc. of the ACM/SAC 2006 Track on Model Transformations (MT2006), Dijon, France pp. 1256–1261.

Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S. And Matera, M. 2002. Designing Data-Intensive Web Applications, Morgan Kaufmann.

Ceri, S., Daniel, F., Matera, M. and Facca, F. 2007. Model-driven Development of Context-Aware Web Applications. ACM Trans. on Internet Technology (TOIT), 7(1).

Chicchetti, A., Di Ruscio, D. and Pierantonio, A. 2006. Weaving Concerns in Model-Based Development of Data-Intensive Web Applications, in: Proc. of the ACM/SAC 2006 Track on Model Transformations (MT2006), Dijon, France, pp. 1256–1261.

De Castro, V., Marcos, E. and López Sanz, M. 2006. A Model Driven Method for Service Composition Modelling: A Case Study, Int. Journal of Web Engineering and Technology, 2(4) pp. 335–353.

De Troyer, O. and Leune, C. J. 1998. WSDM: A User Centered Design Method For Web Sites, in: Proc. of the 7th International Conference on World Wide Web, Amsterdam, The Netherlands, Elsevier Science Publishers B. V., pp. 85–94.

Di Ruscio, D., Muccini, H. and Pierantonio, A. 2004. A Data Modelling Approach to Web Application Synthesis, International Journal of Web Engineering and Technology, 1(3) pp. 320–337.

Frasincar, F., Houben, G. and Vdovjak, R. 2001. An RMM–Based Methodology for Hypermedia Presentation Design, in: Proc. of the 5th East European Conference on Advances in Databases and Information Systems ADBIS '01, London, UK, Springer-Verlag, pp. 323–337.

Fuentes, L. and Vallecillo, A. 2004. An Introduction to UML Profiles. UPGRADE, The European Journal for the Informatics Professional, 5(2) pp. 5–13.

Garzotto, F., Paolini, P. and Schwabe, D. 1993. HDM - A Model-based Approach to Hypertext Application Design, ACM Transactions on Information Systems, 11(1) pp. 1–26.

Gómez, J. and Cachero, C. 2003. OO-H Method: Extending UML to Model Web Interfaces, Idea Group Publishing, pp. 144–173.

Greenfield, J. and Short, K. 2004. Software Factories: Assembling Applications with Patterns, Frameworks, Models & Tools, Wiley Publishing, Inc.

Henderson-Sellers, B. and González-Pérez, C. 2006. Uses and abuses of the stereotype mechanism in UML 1.x and 2.0, in: Proc. of MODELS 2006. Italy.

Jouault, F. and Kurtev, I. 2006a. On the Architectural Alignment of ATL and QVT, in: Proc. of the ACM Symposium on Applied Computing, Dijon, France. ACM Press.

Jouault, F. and Kurtev, I. 2006b. Transforming models with ATL, in: Proc. of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica. Springer-Verlag, LNCS 3844, pp. 128–138.

Kappel, G., Pröll, B., Reich, S. and Retschitzegger, W. 2006. Web Engineering – The Discipline of Systematic Development of Web Applications, Wiley & Sons.

Koch, N. 2001. Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modelling Techniques and Development Process, Softwaretechnik- Trends, 21(1).

Koch, N. and Kraus, A. 2003. Towards A Common Metamodel For The Development of Web Applications, in: Proc. of 3rd International Conference on Web Engineering, ICWE 2003. Springer-Verlag, LNCS 2722, pp. 497–506.

Koch, N., Kraus, A., Cachero, C. and Meliá, S. 2004. Integration of Business Processes in Web Applications Models, Journal of Web Engineering (JWE), 3(1) pp. 22–49.

Koch, N., Zhang, G. and Escalona, M.J. 2006. Model Transformations from Requirements to Web System Design, in: Proc. of the 6th International Conference on Web Engineering, ICWE 2006, Palo Alto, USA, ACM Press, pp. 281–288.

Koch, N. 2006. Transformations Techniques in the Model-Driven Development Process of UWE, in: Proc. of the 2nd Model-Driven Web Engineering Workshop, MDWE 2006, Palo Alto, California.

Kraus, A. 2007. Model Driven Software Engineering for Web Applications, PhD Thesis. Institut für Informatik, Ludwig-Maximilians-Universität München.

Lange, D. B. 1994. An Object-Oriented Design Method For Hypermedia Information Systems, in: Proc. of 27th Annual Hawaii International Conference on System Sciences (HICSS-27), Maui, Hawaii. IEEE Computer Society, pp. 366–375.

Meliá, S. and Gómez, J. 2006. The WebSA Approach: Applying Model-Driven Engineering To Web Applications, Journal of Web Engineering (JWE), 5(2) pp. 121–149.

Miller, J. and Mukerji, J. 2003. The MDA Guide. Draft v. 2.0, OMG doc. ab/2003-01-03.

Moreno, N. and Vallecillo, A. 2005a. A Model-Based Approach For Integrating Third Party Systems With Web Applications, in: Proc. Proc. of 5th International Conference on Web Engineering, ICWE 2005, Springer-Verlag, LNCS 3579, pp. 441–452.

Moreno, N., Romero, J.R. and Vallecillo, A. 2005b. Incorporating Cooperative Portlets in Web Application Development, in: Proc. of the first Model-Driven Web Engineering Workshop, MDWE 2005. Sydney, Australia, pp. 70–79.

Moreno, N. and Vallecillo, A. 2005c. Modelling Interactions between Web Applications and Third Party Systems, in: Proc. of IWWOST 2005. Porto, Portugal, pp. 441–452.

Moreno, N., Fraternalli, P. and Vallecillo, A. 2006. A UML 2.0 Profile for WebML Modelling, in: Proc. of the 2nd Model-Driven Web Engineering Workshop, MDWE 2006, Palo Alto, California.

OMG. 2005a. MOF QVT Final Adopted Specification, OMG doc. ptc/05-11-01.

OMG. 2005b. UML 2.0 Superstructure Specification v. 2.0, OMG doc. formal/05-07-04.

OMG. 2006. OCL 2.0, OMG doc. ptc/06-05-01.

Pastor, O., Gómez, J., Insfran, E. and Pelechano, V. 2001. The OO-Method Approach for Information Systems Modelling: from Object-Oriented Conceptual Modelling To Automated Programming, Information Systems, 26(7) pp. 507–534.

Pastor, O., Fons, J., Abrahao, S. and Pelechado, V. 2006. Conceptual Modelling of Web Applications: the OOWS approach. Web Engineering. Eds. Emilia Mendes and Nile Mosley. Springer, pp. 277–302.

Sánchez, J. and García-Molina, J. 2006. A plugin-based language to experiment with model transformation, in: Proc. of the 9th International Conference MoDELS 2006, Genova, Italy. Springer-Verlag, LNCS 4199, pp. 336–350.

Schauerhuber, A., Wimmer, M. and Kapsammer, E. 2006. Bridging existing Web Modelling Languages to Model- Driven Engineering: A Metamodel for WebML, in: Proc. of the 2nd Model-Driven Web Engineering Workshop, MDWE 2006, Palo Alto, California.

Schwabe, D., Pontes, R. A., Moura, I. 1999. OOHDMWeb: An Environment for Implementation of Hypermedia Applications in the WWW, SigWEB Newsletter, 8(2).

Schwabe, D. 2006. Rapid Prototyping of Web Applications combining Domain Specific Languages and Model Driven Design, in: Proc. of the sixth International Conference on Web Engineering, ICWE 2006, Palo Alto, California. ACM Press.

Thomas, D. and Heinemeier, D. 2006. Agile Web Development with Rails: A Pragmatic Guide. Second Edition, Pragmatic Bookshelf.

Vdovjak, R., Frasincar, F., Houben, G. and Barna, P. 2003. Engineering Semantic Web Information Systems In Hera, Journal of Web Engineering (JWE), 2(1-2) pp. 3–26.