Towards Interoperable Web Engineering Methods

Nathalie Moreno and Antonio Vallecillo
Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga, ETSI Informatica, Campus de Teatinos, 29071, Malaga, Spain. E-mail: {vergara, av}@lcc.uma.es

Current model-driven Web engineering approaches provide methods and compilers for the effective design and development of Web applications. However, these proposals also have some limitations, especially when it comes to exchanging model specifications or adding further concerns such as architectural styles, technology independence, or distribution. One solution to these issues is based on the possibility of making Web proposals interoperate, being able to complement each other, and to exchange models between their tools. We analyze how a common reference model shared by Web engineering proposals can be effectively used to achieve the desired interoperability. We also examine how such a common reference model can be used to combine models coming from different proposals, and discuss the problems that can occur when integrating these separate models. Finally, we show how high-level model transformations allow to efficiently solving these problems.

Introduction

Over the last two decades, a significant number of approaches for Web applications development have been proposed to help developers master the complexity of Web application design and development. Prominent examples include Web Modeling Language (WebML; Ceri et al., 2002), Unified Modeling Language-based (UML) Web Engineering (UWE; Koch, 2001), Object-Oriented Hypermedia Design Method (OOHDM; Baresi, Colazzo, Mainetti, & Morasca, 2006a), W2000 (Baresi, Colazzo, Mainetti, & Morasca, 2006b), or Object-Oriented Hypermedia Method (OO-H; Gómez & Cachero, 2003), among others. Although most of them were developed in academic contexts, they have also been successfully applied in several industrial settings, including e.g., ACER in Europe, the Middle East, Africa and Pan American regions, several banking institutions across Spain, etc.

These proposals are model-based because each of them follows a particular separation of concerns for structuring a system specification into separate viewpoints and adopts a specific notation for describing the each of the viewpoints in terms of models. Furthermore, each initiative provides its own proprietary tool and storage format, normally incompatible with the rest of the notations and tools being used by the other proposals. However, in this context, customers increasingly demand modeling solutions that could interoperate with their current modeling notations and environments, such as the UML, Eclipse, etc.

In addition, most of these Model-Based Web Engineering (MDWE) proposals do not fully exploit all the potential benefits of Model-Driven Engineering (MDE), e.g., complete platform independence or tool interoperability. Furthermore, they also have some limitations, especially when it comes to exchanging models or expressing further concerns, such as architectural styles or distribution. One possible solution to these problems is based on the use of synergies, offering the possibility of making these proposals interoperate, being able to complement each other, and exchanging models between the different existing tools. This would allow each approach to complement its weakest points with the notations, models, or tools from other proposals, hence offering better services and support to customers.

The Institute of Electrical and Electronics Engineers (IEEE) defines interoperability as "the ability of two or more systems or components to exchange information and to use the information that has been exchanged" (Institute of Electrical and Electronics Engineers [IEEE], 1990). One way to achieve the required interoperability between MDWE proposals is by defining a *common reference model* that will enable the conceptual integration of Web engineering approaches (i.e., integration focused on the use of common concepts, artefacts, and models and model-relationships shared by Web modeling proposals) and in which most development methods can be naturally embedded. Such a unified approach will leverage the strengths of the existing proposals by extending their individual models and tools with effective interoperability bridges (transformations) that would allow their seamless integration. Furthermore, in terms of development, it would allow designers to work on a combination of models (not necessarily defined using the same notation or

Received July 2, 2007; revised December 11, 2007; accepted December 11, 2007

 ● Published online 14 March 2008 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/asi.20811

terminology) with well-established interrelations, which could be processed now in a uniform way, e.g., for their visualization, storage, or transformation.

In this article, we present the Web Engineering Interoperability (WEI) common reference metamodel for model-driven Web engineering. WEI has been developed based on our previous joint work with other Web engineering groups, including the OO-H, WebML and UWE teams. With them we have been working on the incorporation of further modeling concerns to their proposals, defining metamodels and profiles for their notations, and restructuring their approaches according to the Model Driven Software Development (MDSD) principles. Based on these experiences, the need for a common reference model that could allow them to interoperate was clear. The resulting reference model, WEI, is also expected to serve as a key input to the emerging MDWEnet initiative (Vallecillo et al., 2007), whose goal is to improve current practices and tools for the model-driven development of Web applications for better interoperability.

Furthermore, the Model Driven Architecture (MDA) initiative (Miller & Mukerji, 2003; Object Management Group, 2001) and their associated standards, technologies, and tools play a cornerstone role in WEI. In fact, interoperability is one of the major goals of MDA, and it has inspired a new approach for organizing the design of an application into different models so that portability, interoperability, and reusability can be obtained through architectural separation of concerns. MDA covers a wide spectrum of topics and issues ranging from Meta-Object Facility-based (MOF) metamodels to UML profiles, model transformations, and modeling languages. Thus, following the MDA guidelines, we will examine how different modeling approaches and tools can be combined and complemented in the context of WEI to support and facilitate the development of a single Web application. In this sense, the required *interoperability bridges* can be naturally defined as straightforward model-to-model transformations. These interoperability bridges will allow resolving different kinds of problems including, e.g., interoperability conflicts such as those related to syntactical heterogeneities, or the integration of models which represent different concerns of the same system. Finally, we are aware of the current lack of mature tools to support MDA and some of its constituent standards, such as the Object Constraint Language (OCL) or Query/View/Transformation (QVT). Until these tools become available, we have implemented a tool called "GlueWeb" to support and validate some of the features of our proposal. GlueWeb implements a noncomplete OCL and QVT parser for a subset of these two languages and it is able to translate OCL constraints and QVT relations into the appropriate code in the final implementation.

The remainder of this document is structured as follows. After this introduction, the *A Generic Framework for Web Applications* section describes the WEI metamodel, a generic framework for Web application development that can be considered as a common reference metamodel for Web engineering proposals. Then, the *Designing Web Applications by Reusing Models From Other Methodologies* section analyzes a set of problems encountered when interoperating models coming from different methodologies. These problems are addressed, allowing us to instantiate WEI to match and generate a "complete" specification of complex Web systems by reusing preexisting models. After that, the *WEI Tool Support* section studies the practical viability of assisting the MDA-based WEI development process by reusing available design and code generation tools. Next, the benefits of our approach are illustrated, as a proof of concept, with the well-known Web application example of the *Conference Review System*. This system is designed and developed in the *Proof of Concept: The Conference Review System* section, reusing different models and tools coming from different MDWE approaches, namely OO-H and WebML. Then, the *Related Work* section relates our work to other similar proposals and, finally, the *Conclusions* section draws some conclusions and outlines some future research activities.

A Generic Framework for Web Applications

The unification of several Web modeling languages into a common metamodel may form the basis of a generic framework that can be instantiated for different choices of notations. In fact, we can also provide a new notation and methodology to that common metamodel, resulting a new proposal with many advantages from the perspective of interoperability. In addition, we can extend the common metamodel to cover advanced requirements and features that current Web design methods support weakly (e.g., the integration of third-party systems or modeling behavioral features). The following subsections describe these issues in more detail.

The WEI Architecture

WEI is a model-driven Web engineering architectural framework for organizing the models that address the different concerns of Web application development. In the context of this article, the term "framework" is used to refer to an architectural design describing the major artefacts or models of a system and the way they interact. Thus, at a high architectural design level, the whole WEI concept space is captured by means of three levels or viewpoints, which comprise up to 13 models (see Figure 1). Of course, while no more than five or six models are normally used to represent the various facets of a system by the existing proposals, WEI defines 13 models not only for completeness but also to facilitate the integration and reusability of external designs. This does not mean that all the 13 models need to be defined for every single Web application design. The emphasis of each level will depend on the kind of Web application being modeled and on the particular project requirements, as we shall see later.

The central element of our architecture is the *Conceptual model*, which can be used for specifying the basic structure and content of the Web application as well as to maintain the consistency of the model specifications, establishing the correspondences between the different models. This Conceptual model plays a relevant role when the Web application

DOI: 10.1002/asi

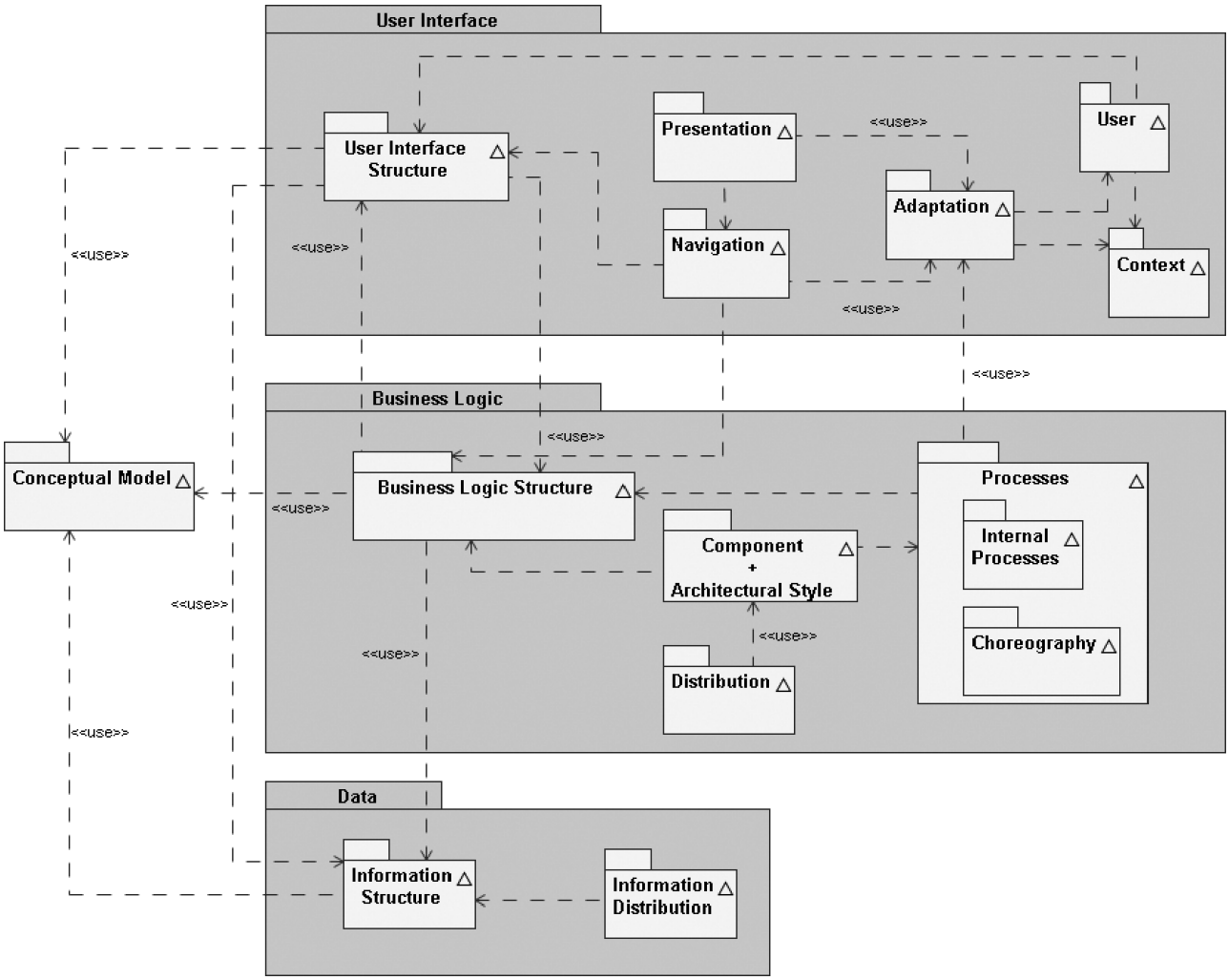


FIG. 1. Models representing the different concerns involved in the development of Web applications.

design integrates third party system models using heterogeneous notations, terminology, or even different platforms, as will be explained in more detail in *Designing Web Applications by Reusing Models from other Methodologies* section.

Leaving out the conceptual model, WEI models are organized into three main packages corresponding to three different viewpoints of the same system: the *Data Structure*, the *User Interface*, and the *Business Logic* viewpoint.

- The *Data structure* package describes the organization of the information managed by the application by means of, e.g., database systems or Lightweight Directory Access Protocol (LDAP) directories, and provides a mechanism for storing information persistently. This level is organized in two models: *Information Structure* and *Information Distribution*.
- 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. Originally, Web applications were specifically conceived to deal with *Navigation* and *Presentation* concerns, but currently they also need to address other relevant models or concerns at this level, such as *Adaptation*, *User,* or *Context*.
- 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. This level is organized in six models: *Business Logic Structure*, *Internal Processes*, *Choreography*, *Component*, *Distribution,* and *Architectural style*.

Each of these metamodels constitutes a part of the common reference metamodel. They have not been arbitrarily chosen; they are based on the concerns covered by existing Web engineering proposals and our previous experience with the development of large distributed applications. Consequently, there is a correspondence between WEI concerns and those established by other modeling approaches. This correspondence is summarized in Table 1.

In addition to the models, the framework predefines some dependencies between them, 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

DOI: 10.1002/asi

TABLE 1. Concerns and models covered by Web engineering methodologies.

Layer	WEI model	OOHDM	W2000	UWE	WebML	WSDM	OO-H	WAE
User interface	UI structure	√	∼	√	∼	∼	√	
	User	∼		√	√	√		
	Context	∼		√	√			
	Adaptation			√	√			
	Navigation	√	√	√	√	√	√	√
	Presentation	√	√	√	√	√	√	√
Business logic	Structure	√	∼	∼		√	√	
	Int. processes	√	√	√			∼	
	Choreography							
	Architecture			√	√		√	√
	Distribution							√
Data	Inf. structure	√	√	√	√		√	
	Inf. distribution							

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.

Modeling WEI Concerns

In order to formally define the framework, we have built a MOF metamodel for each model, which describes its entities and their relationships. (These metamodels are available at http://www.lcc.uma.es/~nathalie/WEI/.) MOF was selected as metamodeling language because of 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 and see whether it was indeed valid or not. As an example of these metamodels, Figure 2 describes the WEI presentation metamodel.

However WEI metamodels provide just the abstract syntax for the domain concepts. Unlike other approaches, OMG does not provide a solution for directly building correct models from metamodels. Instead, designers have to define their own Domain Specific Language (DSL) associated with these metamodels. OMG offers three possible approaches for defining domain-specific languages: (a) to specify a new

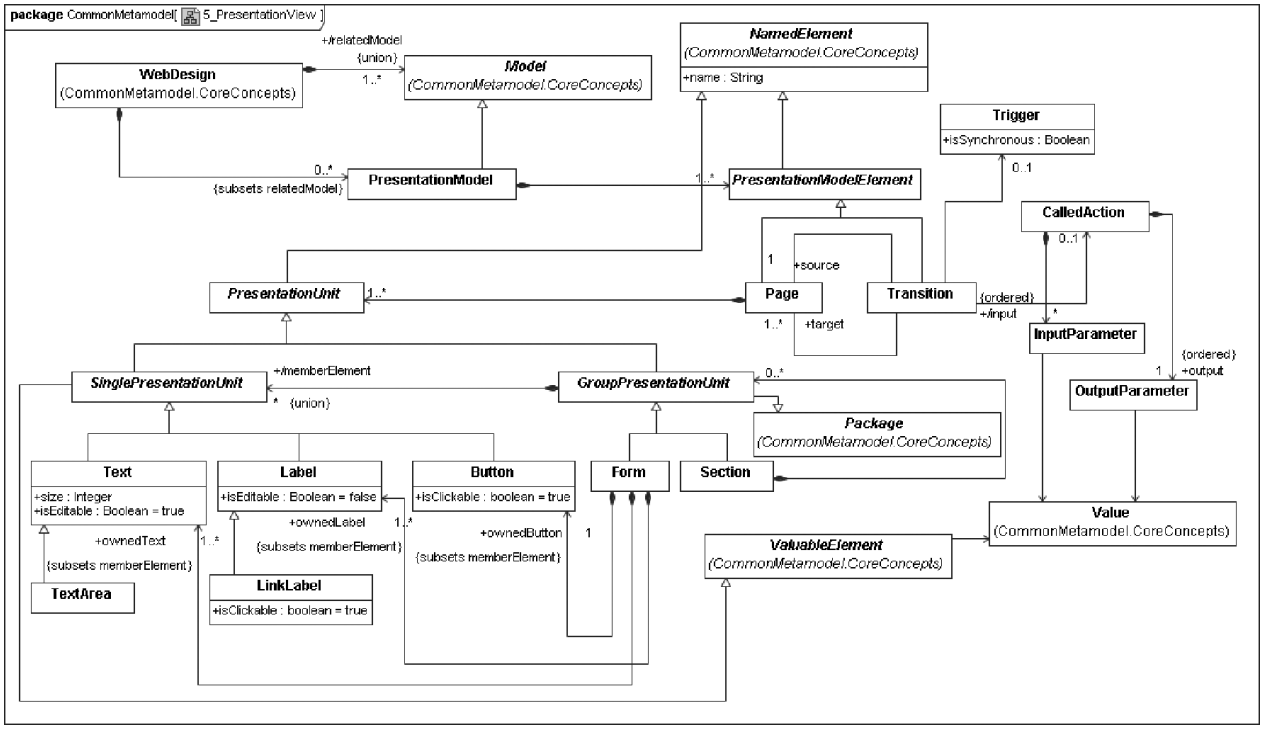


FIG. 2. An excerpt of the metamodel for the WEI presentation concern.

DOI: 10.1002/asi

domain language, an alternative to UML, using the MOF metamodeling facilities provided by OMG for defining object-based visual languages; (b) to extend the UML metamodel (heavyweight extension), modifying the original UML metamodel and its semantics; (c) to define a UML profile (lightweight extension) based on the extension mechanisms provided by UML (OMG, 2005b; Fuentes & Vallecillo, 2004) (stereotypes, tag definitions, and constraints) for specializing UML metaclasses but without breaking their original semantics.

The problem with the first and second solution is that standard UML tools will not be able to deal with such a new language (to edit models that conform to the metamodel, compile them, etc.). In consequence, we defined lightweight extensions of UML, i.e., UML profiles, for representing these models. These profiles provide the concrete syntax for each of the WEI metamodels, i.e., WEI concepts have been represented as stereotyped UML elements, using the standard extension mechanisms provided by UML for developing new domain-specific languages. WEI stereotypes are more than mere notational variations of the concrete syntax or visual representation of the UML language elements. From the Web application developer viewpoint, WEI stereotypes are first-class members in the UML language that extend UML with a new vocabulary and semantic restrictions on the added language elements.

As an example, Figure 3 shows the profile for the WEI presentation model and its associated stereotypes, using the notation prescribed by UML to specify profiles. Each stereotype definition specifies the UML 2 metaclass extended, the tag definitions (described as properties of the stereotype), and constraints (described as constraints on the stereotype). The whole set of profiles and their complete description can be found at the WEI Web site, http://www.lcc.uma.es/~nathalie/WEI/.

WEI profiles have been successfully applied to define and implement several kinds of Web applications, such as a *Conference Review System* or a *Travel Agency*, as documented in detail in Moreno and Vallecillo, 2005a, 2005b, and 2005c. Profiles were also designed to support the current trends in the development of distributed Web applications based on the principles of reusing and assembling preproduced components, such as Web services or portlets, in order to reduce development costs and efforts while improving software quality. Currently, this particular capability of WEI profiles is being exploited to generate rapid prototypes of rich user interfaces on top of Web services and existing business applications, in projects that require different kinds of front-ends to interact with the underlying business logic. In particular, this strategy is being applied to integrate three Maude tools, such as CiME (http://cime.lri.fr/), MuTerm (http://www.dsic.upv. es/~slucas/csr/termination/muterm/), and AProVE

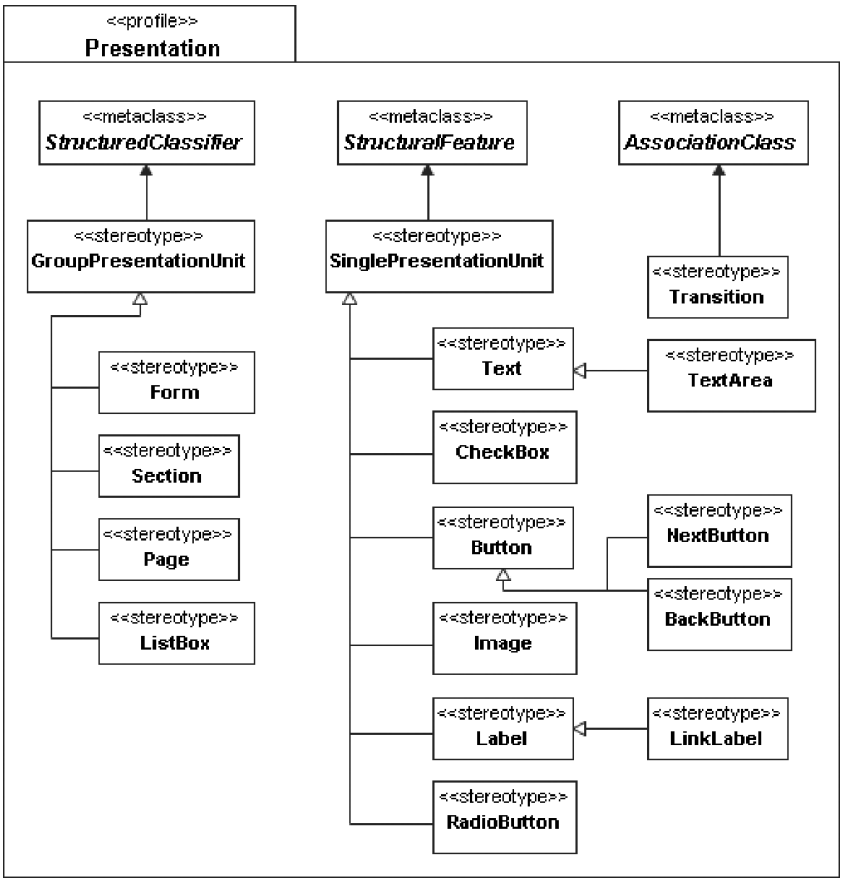


FIG. 3. Profile for the WEI presentation model.

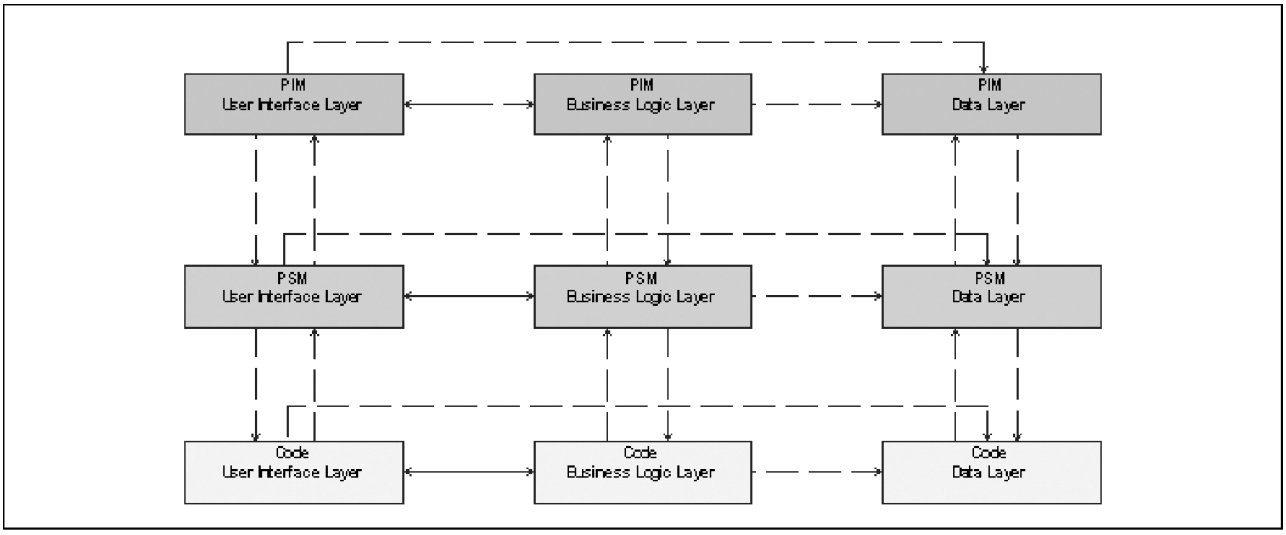
DOI: 10.1002/asi

(http://www-i2.informatik.rwth-aachen.de/AProVE/), into one unified, rich user interface.

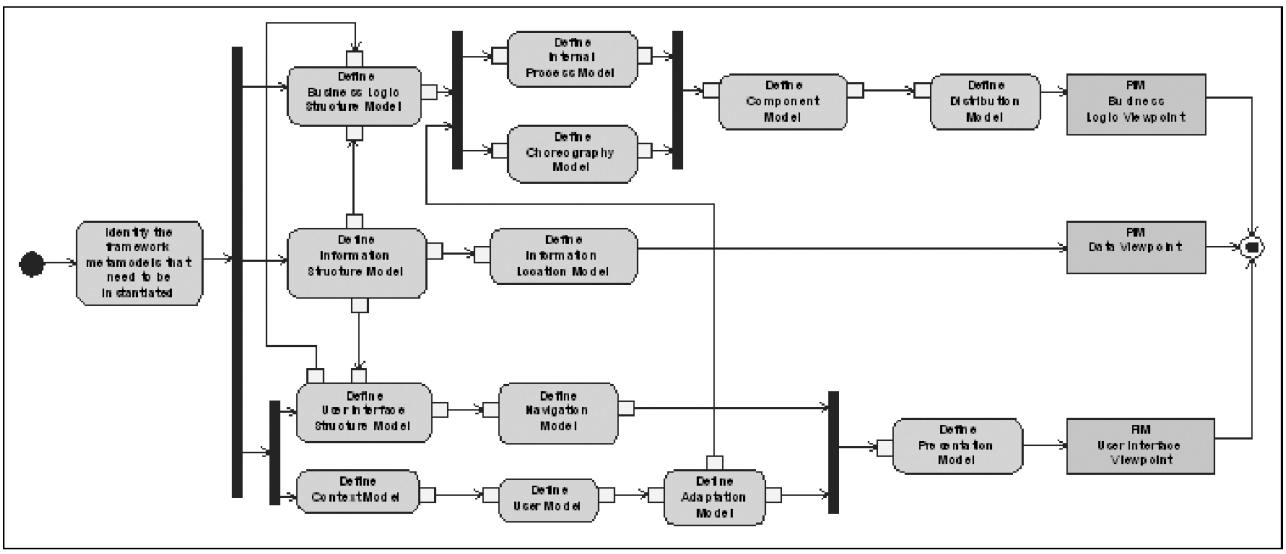
The WEI Methodology

In short, the WEI development process involves the definition of at least three Platform Independent Models, each one corresponding to a viewpoint as illustrated in Figure 4a. How to develop each PIM is graphically represented by the activity diagram depicted in Figure 4b in which activities can have inputs and outputs (i.e., pins in the UML terminology, which represents input and output models in our context). By means of object flow edges, we show how the model provided by some action is received, refined, and enriched with new information by another activity to produce an output model. Currently, this process is only partially automated because there are certain steps that require manual intervention and some decisions to be made by the software engineer (for example, how to combine and merge the internal processes and choreography models into a single model). Therefore, Figure 4b should be interpreted as a set of systematic guidelines to develop Web applications using WEI.

The transformation of our top-level Platform Independent Models to Platform Specific Models requires the definition of a set of dedicated mapping rules. Before applying the mappings, the designer has to mark the PIMs using the appropriate profile for the target platform and technology. The result of the application of such mapping rules is a set of UML 2.0 models of the application according to the target platform technologies (e.g. Java, JSP, Oracle, etc.). Finally, the PSMs are translated to code by applying a transformation process again, where special care should be taken with the bridges between the three PIMs and their corresponding PSMs, for which transformations are also required.



(a) MDA chain



(b) Building Web applications from scratch

FIG. 4. The WEI methodology process.

DOI: 10.1002/asi

The interested reader may refer to (Moreno and Vallecillo, 2005a, 2005b, and 2005c) for more details about the straightforward application of the framework in the context of MDA to develop Web systems. In this article, we focus on how to use this framework as a common reference metamodel to achieve interoperability between Web engineering proposals, so that the design of a Web application can be addressed by reusing models coming from different modeling approaches.

Designing Web Applications by Reusing Models From Other Methodologies

In addition to using WEI to build Web applications from scratch, one of the major advantages of our proposal is its ability to design and implement Web applications reusing existing models and tools (e.g., model compilers) from other methodologies. Thus, a Web application developer could use, for instance, UWE or OO-H notations to design the models of the *User Interface* layer and WebML to design the *Data* layer or vice versa. Furthermore, models already defined for other applications could be reused here for fast prototyping of Web applications.

Reusing models from other Web methodologies requires the definition of interoperability bridges between these models and the appropriate models of our framework (Table 1 summarized the concerns and models covered by Web engineering methodologies with regard to WEI). Usually, the concepts and artefacts used by existing Web modeling languages do not differ much. In addition, neither the models nor the concepts described in our framework were arbitrarily chosen; instead, they try to generalize the concepts and models defined by most Web engineering proposals. Thus, the interoperability bridges between models from different approaches are feasible a priori and even quite straightforward using WEI as a reference metamodel.

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 element. Second, we may find that one model may assume the existence of other models that either provide some services or operations (e.g., the precise behavior 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, for example, exchanging data or 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 explicitly modeled by one of the proposals because they are implicitly assumed (the most typical example is behavior).

The way in which we address the first four issues is by specifying bridges between the elements living in different models. Such bridges will normally be specified in terms of model transformations (e.g., QVT relations), as described in the following paragraphs. 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 happens when the models to be reused come from methodologies that do not have all their information explicitly modeled but hard-wired into their supporting CASE tools. Thus, the models to be reused assume the existence of some information and semantics that 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. We show an example of this and how to deal with it in *Proof of Concept: The Conference Review System* section.

Issue 1: Different Names for the Same Modeling Element

We cannot suppose that software developed by independent parties will assign the same names to the same entities. Hence, a common problem when relating separate views of a system is the use of different names for modeling elements that represent the same system element. These modeling elements need to be related by the appropriate correspondences.

UML 2.0 abstraction dependencies, possibly constrained by OCL statements (OMG, 2006), are the natural mechanisms provided by UML to represent a relationship that relates two elements or sets of elements that represent the same concept at different levels of abstraction. However, UML dependencies cannot store all the required information about the correspondence they represent and cannot guarantee the enforcement or propagation of information that is required in some situations (e.g., in change or evolution management).

The solution adopted here is the use of QVT relations (OMG, 2005a) to represent the correspondences between elements in different models. In most cases, instead of directly relating the elements from two separate models, we go through the WEI *Conceptual* model. Although this might add certain complexity in some situations, it also facilitates the process when there is the need to specify correspondences between elements with more than two models. In fact, this strategy decreases significantly not only the number of possible conflicts due to semantic differences but also the number of model transformations to be developed (because going through WEI can bring the gap between both languages).

For instance, the following QVT relation solves a name mismatch between classes person and user of two WebML and OO-H models, and also between their corresponding attributes.

DOI: 10.1002/asi

```
relation USLPerson2ISUser{
  /* maps person to user */
  checkonly domain webml.WebML_Metamodel e1:
     WebMLEntity {
    name = 'Person',
    attribute = a1:WebMLAttribute{
         name = a1n,
         type = p1:WebMLDataType {name = p1n}}};
  enforce domain ooh.OO-H_Metamodel e2:
     OOHEntity {
    name = 'User',
    attribute = a2:OOHAttribute{
         name = a2n,
         type = p2:OOHDataType{name = p2n}}};
  where{
    sameNameEntity(e1,e2);
    sameNameAttribute(e1,e2);
    equivalentTypeAttribute(e1,e2);}
  }
```

Issue 2: Need for Explicit Bridges

A model may also implicitly assume the existence of another model, which provides some required services or operations. These other models may be local (for example, those models specifying the required behavior of the application) or may even represent external systems (e.g., external Web services or legacy applications).

In the proposal by Mellor and Balcer (2002), these kinds of correspondences are called explicit bridges, and they can be used to link application and intermediate abstraction domains to real domains that represent external systems and legacy software. More precisely, these bridges are represented as signals to and from external entities (of a model) or invocation of operations on such entities.

In the WEI context, the typical example is an element representing a *NavigationLink* in the *Navigation* model, which may require the invocation of a method of the *Business Logic Structure* model. Thus, *NavigationLinks* have to be related to *Transitions* located in time and space in the *Presentation* model and also have to be related with *Events* in the *Internal Processes* models triggering the execution of an associated behavior.

For illustration purposes let us consider a simplified example that shows an excerpt of a login page containing fields for a username and password and that should be able to go to the registered user screen. Aligning the Web page concept with the structured class concept of UML 2.0, Figure 5 specifies a Web page as a structured UML class with internal "parts," some of which can be references to external objects (i.e., not owned by the enclosing object) shown in the diagram by a dashed rectangle. One benefit of using a structured classifier in this case allows the internal structure (decomposition) of Web pages to be represented in a new and visual way. The manner in which WEI connects the *User Interface* and *Business Logic* viewpoints is shown in Figure 5. The bridge between the user interface and business logic is defined by the UML dependency relationship between the `LoginMenu` and `UserInformation` classes in that figure, with an added OCL expression in the conceptual model. This constraint establishes that the behavior of the method `login()` in the user interface side is provided by method `login()` of class UserInformation in the *Business Logic* side:

```
context LoginLink::
   login(uid:String,pwd:String):Boolean
      body: ui.login(uid,pwd)
```

Issue 3: Different Levels of Abstraction

As shown in Table 1, not all methodologies provide the explicit representation of all of our framework's models. For instance, UWE does not define an *Information Structure* model. However, UWE's *Content* model is used for both representing the structure of persistent data, and to provide support to other models such as *Navigation* and *Presentation* (which belong to WEI's *User Interface* level).

The most natural way to populate WEI's models with those coming from other approaches is by defining QVT relations that generate the appropriate model elements in all the corresponding WEI models from the model elements of other approaches.

QVT transformations can specify single direction and bidirectional transformations, verify that models are related in a specified way, and establish relationships between already preexisting models. Thus, when a transformation is invoked, it first checks whether the relations hold and, if not, it attempts to enforce them by creating, deleting, or modifying the target model.

Consider the example in Figure 6b that reports a simple WebML hypertext, containing a page entitled *recommended CDs*. This WebML page consists of an index unit defined for the Recommended_CD data model entity, which shows a list of most popular CDs of the previous month and a data unit also defined over the CD entity, which displays the details of the CD selected from the index. Two selectors (`[Year = System.year()], [Month = System.month()]`) are defined to restrict the selection only to the CDs of the current month and year. The arrow between the two units is a contextual link, carrying the parameter `CurrentCd`, containing the object identifier (`OID`) of the selected item. The data unit includes a parametric selector (`[OID5CurrentCD]`), which uses the input `OID` parameter to retrieve the data of the specific movie.

The corresponding model in the WEI framework is derived by applying a set of relations. In particular, the relation *WHDataUnit2MNNavigationUnit* establishes a relationship between *Data Units* in the WebML approach and *NavigationUnits* in WEI models, whereby every *Data Unit* in the WebML hypertext model generates one *NavigationUnit* with the same name, attributes and constraints as its native WebML *Data Unit.*

DOI: 10.1002/asi

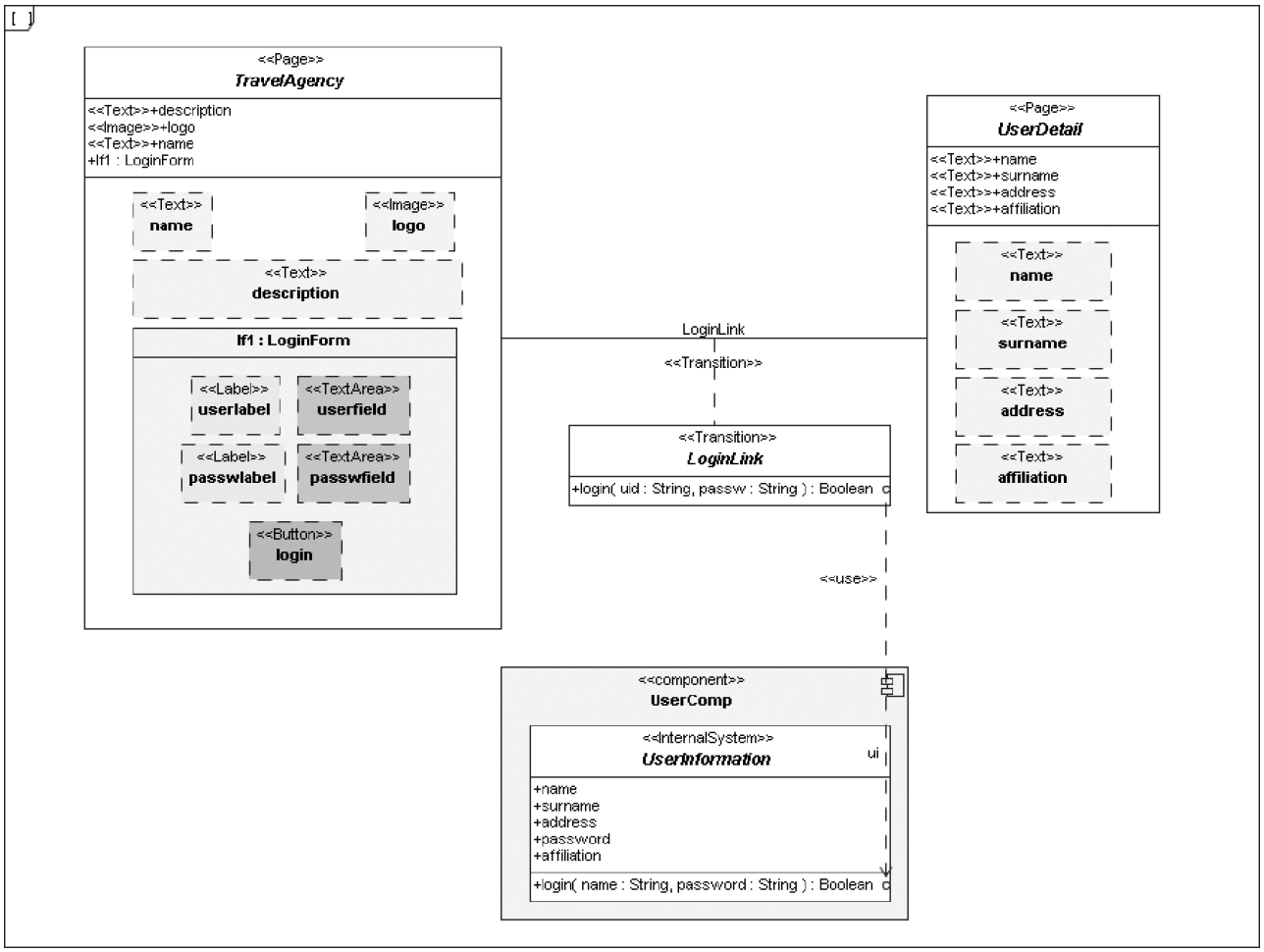


FIG. 5. An excerpt of the PIMs for the user interface and business logic viewpoints.

```
modeltype SC uses "webml.WebML_Metamodel"
modeltype TG uses " WEI.WEI_Metamodel"
transformation WebMLHypertext2WEINavigation ( mdsc:
SC, mdtg: TG) {
    top relation WHPage2MNGroupUnit{...}
    top relation WHIndexUnit2MNIndex{...}
    top relation WHContextualLink2MNAssociation-
         Class{...}
    top relation WHDataUnit2MNNavigationUnit{...}
    relation WHSelector2MNMethodParameter{...}
    ...
}
top relation WHDataUnit2MNNavigationUnit{
    /* maps WebML data units to WEI navigation units */
    checkonly domain mdsc du: ContextUnit{
        view = p:HypertextPackage{} ,
        type ='DataUnit',
        attribute = a:WebMLAttribute{
            name = a1n,
            type = p1:WebMLDataType{name = p1n},
        condition = s:SelectorCondition{
            cond = c1,
                type = Boolean},
            name = x
    }
    enforce domain mdtg nu: NavitationUnit{
        view = c: NavigationModel{},
        name = x,
    when {HypertextPackage2NavigationModel(p,c);}
    where {nu.stereotypedBy("NavigationUnit");
        sameAttributes(du,nu);
        sameConstraint(du,nu);
    }
}
```

Issue 4: Different Separation of Concerns

Many Web engineering approaches group several concerns in a unique view. This strategy produces more compact representations, although for medium/large applications, these views can be very complex and difficult to maintain. In these cases, it should be desirable to change the system model to improve its internal structure without altering its external behavior, that is, to a apply a refactoring process (Mens & Tourwe, 2004) that allows us to reorganize

DOI: 10.1002/asi

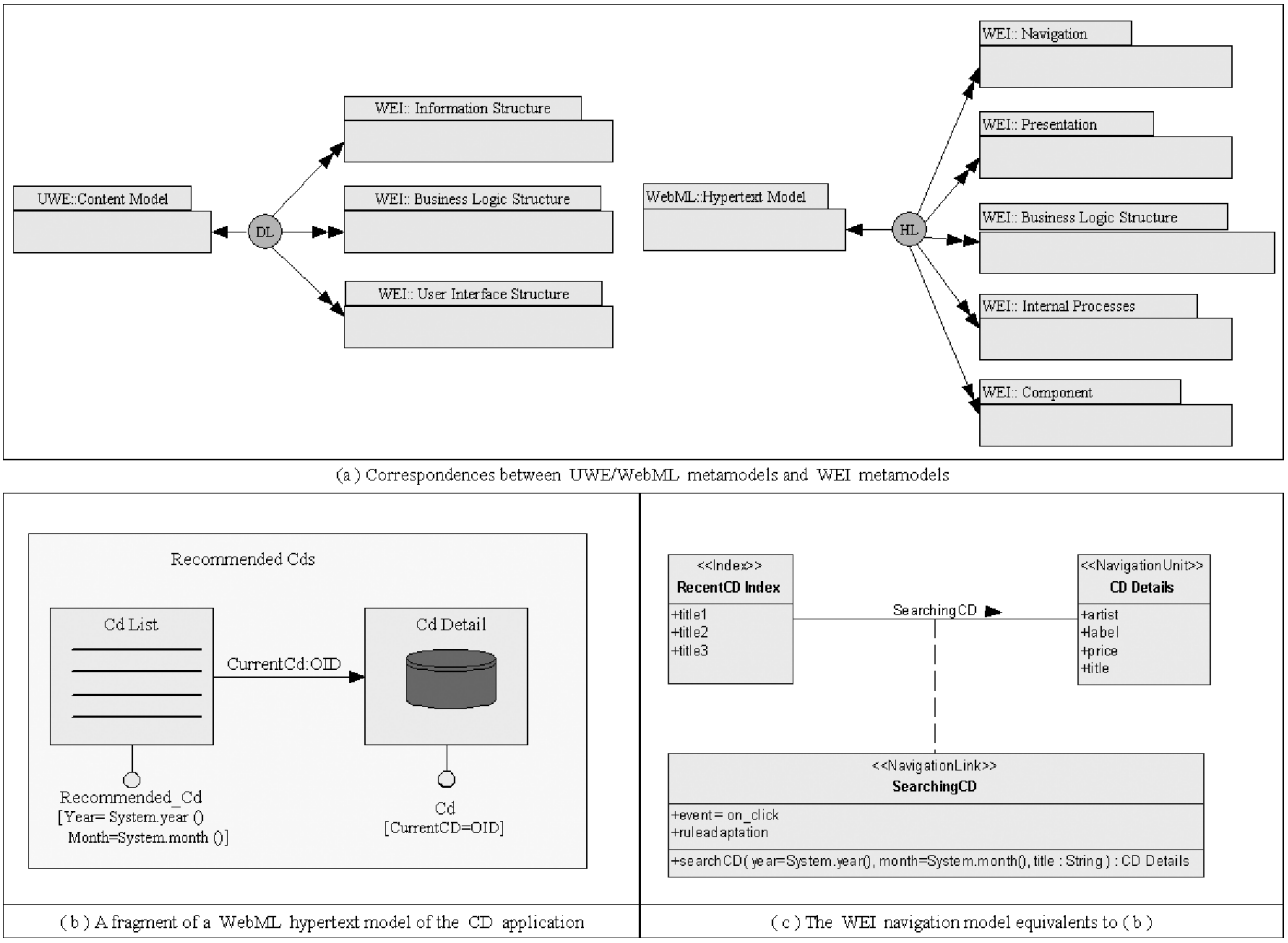


FIG. 6. Correspondences between UWE/WebML and WEI.

representations that mix different concerns, achieving both a greater modularization in accordance with our framework's separation of concerns and a reduction of the complexity and understanding of their representations. In WEI, the refactoring process is performed by interoperability bridges, which are also defined here in terms of QVT transformations.

To illustrate this issue, let us go back to Figure 6b. The reader can already infer from it that the WebML hypertext view encapsulates several WEI concerns. Thus, given a WebML hypertext view, the following QVT relation is defined to generate its equivalent WEI view and check that every WebML hypertext element has its counterpart in the corresponding *Navigation, Presentation, Business Logic, Internal Processes*, or *Component* models of the WEI framework, as shown pictorially in Figure 6a.

```
modeltype SC uses "webml.WebML_Metamodel"
modeltype TG uses "WEI.WEI_Metamodel"
transformation     WebMLHypertext2WEIBusiness
   (mdsc: SC, mdtg: TG) {
   top  relation  WHContextualLink2MBMethod{...}
   ...
}
```

```
transformation WebMLHypertext2WEIPresentation
   (mdsc: SC, mdtg: TG) {
   top relation WHPage2MPGroupUnit{...}
   ...
}
transformation    WebMLHypertext2WEIComponent
   (mdsc: SC, mdtg: TG) {
   top  relation  WHIndexUnit2MComponent{...}
   ...
}
...
```

WEI Tool Support

One of the most common shortcomings of many of the model-driven Web engineering proposals is the use of proprietary notations and tools for designing and developing Web applications. This issue can be addressed, for example, by adopting standard notations and tools, which aim at guaranteeing the interoperability between separate vendors and service providers. In particular, UML is an OMG standard widely used and adopted as a standard in the software industry for modeling software, and therefore we decided to use it

DOI: 10.1002/asi

as a basis for our proposal. Furthermore, MDA and its supporting tools provide the separation of concerns and interoperability mechanisms that we need to integrate models and tools coming from different proposals.

In the next subsections, we explain how to assist the WEI development process in order to generate the design and the implementation of a Web application. We analyze how well current UML and MDA-based CASE tools fit into our methodology requirements.

UML-Based Tools to Support WEI Applications Design

One advantage of WEI over other Web methodologies is that all standard UML modeling tools that support UML profiles or UML extension mechanisms can be used to create WEI models of Web applications. Therefore, we have not developed a specific tool to draw the UML diagrams of an application. Instead, software developers can use any available UML editor able to generate the corresponding WEI models (e.g., MagicDraw or Eclipse UML2).

Due to the fact that most code generation tools use XMI as the interchange and storage format, it is desirable that the UML editor provides exporting facilities to this format. Choosing the most appropriate UML editor for our project is extremely important as they do not interoperate well, and consequently, this choice may contribute to the success or failure of the project.

Code Generation Support for WEI Applications Design

Once WEI models have been exported to the appropriate XMI format, we have studied two alternative paths for assisting the WEI code generation process: (a) taking advantage of existing MDA-based CASE tools and (b) reusing proprietary tools of other Web engineering methodologies.

Regarding the first option, there are at least 80 tools right now (including commercial, free and open source tools) that support one or more major features of the MDA, including design models, transformation rules, automatic transformation, mapping, integration, code generation, reverse engineering, and platforms support. If it is technically possible, these CASE tools will be further extended to automatically perform some steps of the WEI methodology process shown in Figure 4b (Moreno & Vallecillo, 2005c).

However, we are aware that most MDA tools are currently still immature. This is why we have explored other ways to produce code for the WEI application design (see the *Reusing Proprietary Tools of Other Web Engineering Methodologies* section). It should be noted that one of the advantages of achieving interoperability between Web engineering proposals is that both application models and tools from other approaches can be reused. Let us study the viability of reusing tools in the following subsections.

Reusing UML and MDA-based CASE tools. Although there are tools that are already based on MDA principles and produce code for different platforms, most of them are not Web-oriented in the sense that their compilers are not able to generate specific code for Web platforms. Moreover, we have seen that using certain MDA-based tools for automating the code generation process may imply one further transformation step because current MDA-based tools always rely upon some clearly defined preconditions (modeling conventions). This usually means that our annotated PIMs have to be restructured to adhere to the way in which these tools work. We hope this will change in the near future as tools are packaged as MDA-components, so they can be easily integrated into the MDA chain. But in the meantime, this is a drawback of current MDA tools.

The most important drawback however is the lack of a standard syntax for specifying actions in UML. There are currently (as of year 2007) clear semantics specified for UML action languages, but no syntax has been agreed for them yet. Instead, there is a variety of action languages, but they are incompatible and supported by proprietary tools. Examples include the BridgePoint Object Action Language (OAL; Mellor & Balcer, 2002), the Kennedy Carter Ltd. Action Specification Language (ASL; Raistrick, Francis, Wright, Carter, & Wilkie, 2006), the Shlaer-Mellor Action Language (SMALL; Shlaer & Mellor, 1992), the Xion language (Muller, Studer, Fondement, & Bézivin, 2005), etc. Each action language (and its supporting CASE tool) is focused on a certain set of requirements and concrete application domain, so we need to select the one that best fulfils our Web application requirements. Please notice that we need to make use of at least one of them because we explicitly need to represent behavior, in contrast to most Web engineering proposals, that implicitly specify it in their modeling entities and then hardwire such behavior into their code-generation tools.

Based on our experiences and findings, most of the existing action languages and tools are not Web-oriented, but they can be successfully used to model most of the behavioral parts of a Web application, such as the business logic layer, and then provide good implementations because these tools generally give explicit support for statechart diagrams (generating Web services or software components at the implementation level). In our case, we have successfully used Netsilon and Kennedy Carter's iUML Product Suite to develop, integrate, and simulate executable UML models. However, we admit that other tools specific to the Web domain can be more appropriate for implementing other WEI viewpoints, such as the *User Interface* layer. So, why not use them jointly to generate the implementation of a single Web application?

Reusing proprietary tools of other Web engineering methodologies. At first glance, there is nothing to prevent us from reusing the compilers of other Web engineering methodologies. For instance, we could start from the models defined using our framework as PIMs and then use the interoperability bridges previously defined (in the opposite direction) to derive, e.g., the PIM for the *Presentation* layer in the OO-H notation and the PIM for the *Data* layer in the WebML notation (see Figure 7).

Once we have the corresponding WebML and OO-H models (we could have also started with them), we can use

DOI: 10.1002/asi

both the Visual Wade and the WebRatio compilers to produce the application Web pages and the physical database, respectively.

Finally, we need to compile several PSMs into a single system. This is where the correspondences (i.e., bridges) described in the previous section need to be defined between elements at each level of abstraction. This is illustrated in the next section, which provides the description of this process in the particular case of a prototypal Web application, the *Conference Review System*.

The reader should note that the selection of these target methodologies (WebML and OO-H in this case) probably implies tying the implementation to particular architectural styles or technologies, e.g., client-server configurations or relational databases because these are the fixed technologies supported by the selected methodology. In addition, the interoperability bridges that we had to specify at all levels are usually developed by most Web engineering proposals at the implementation level only and are dependent on the model compiler and the code generation tool. However, they need to be explicitly represented at all levels in order to be both technology and platform independent.

Gluing implementations generated reusing Web engineering proprietary tools. In order to overcome the problem of the current lack of proper tool support for implementing QVT transformations (i.e., WEI bridges), we have developed the GlueWeb tool. It has been implemented as an Eclipse plugin on the Eclipse v3.3 platform that provides a project wizard allowing the creation of "glue projects" with a predefined structure. Basically, the GlueWeb plugin performs two main tasks. The first one is focused on collecting the technical information about the three WEI viewpoints. More precisely, for each WEI viewpoint the designer provides information about the physical location in which implementation files are stored, the platform technology that has been used for it (e.g., HTML, PHP, JavaScript, Java.), and the XMI representation corresponding to its model specification (see Figure 8). Then, a local copy of these files is stored so that they can be used later.

Once this task has been completed, GlueWeb carries out an internal step that is only necessary in those cases in which several heterogeneous technologies have to interact. This internal step consists of validating whether the provided implementation platform technologies are compatible or not (see Figure 8). If this is the case, GlueWeb performs the analysis of the interoperability bridges. Currently GlueWeb solves, to a certain extent, many of the problems pointed out in the *Designing Web Applications by Reusing Models from other Methodologies* section. To be precise, GlueWeb

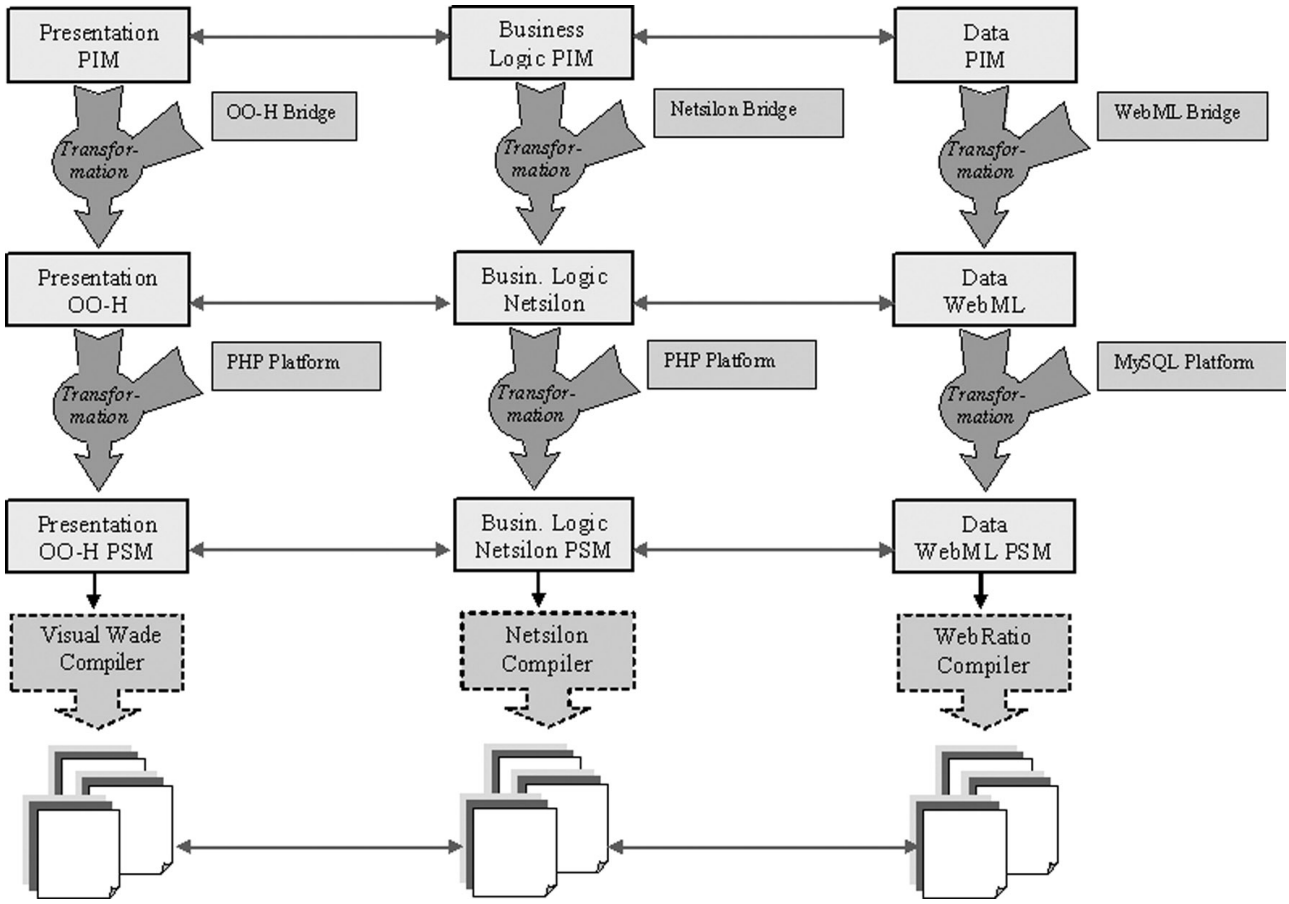


FIG. 7. Code generation with compilers from other proposals.

DOI: 10.1002/asi

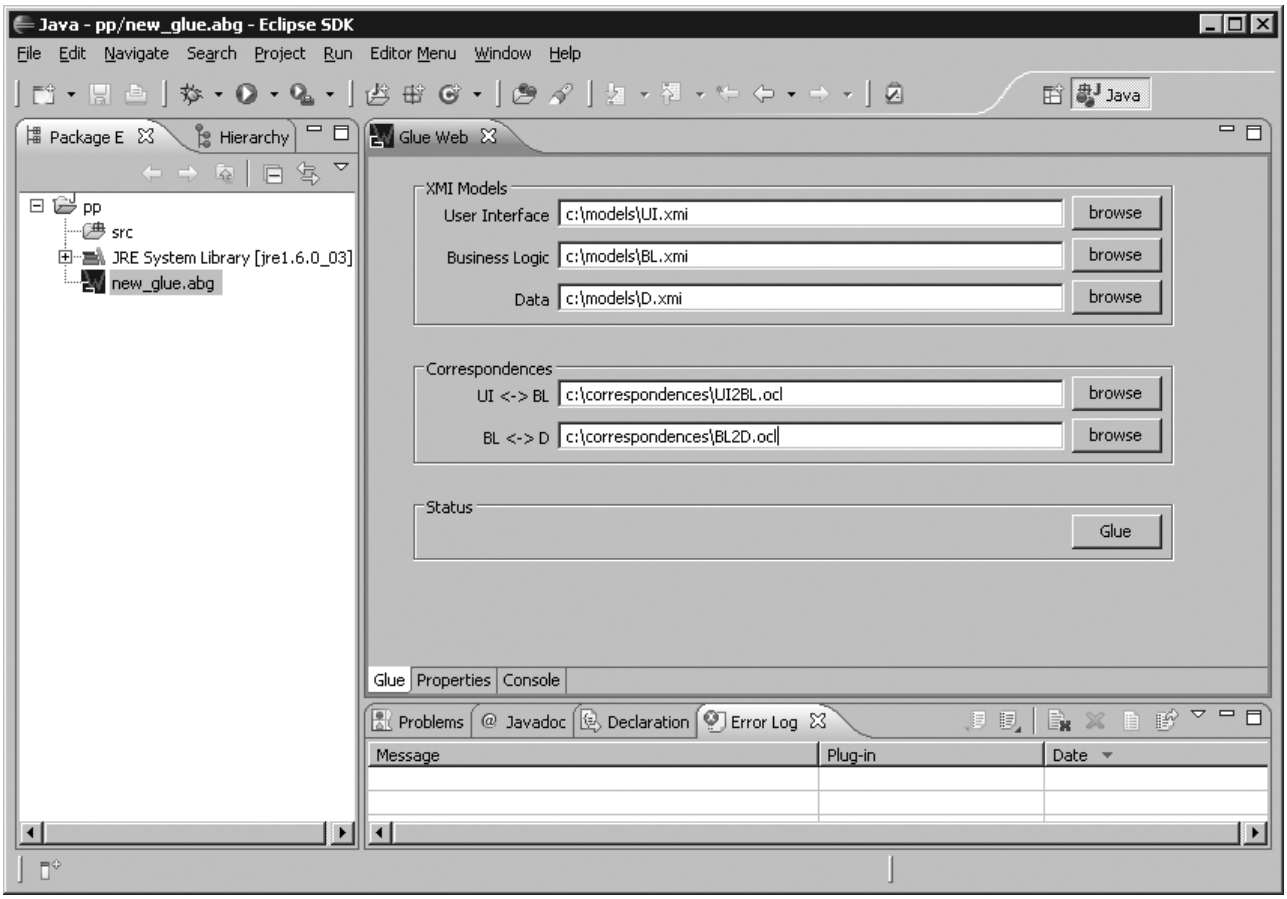


FIG. 8. A snapshot of the GlueWeb eclipse plugin.

includes a noncomplete OCL and QVT relation parser which is able to read and process OCL constraints and QVT correspondences assuming they are syntactically and semantically correct.

A relevant feature of this plugin is that its OCL parser appropriately modifies the initial implementation corresponding to each WEI viewpoint to allow the correct communication between the different layers. To illustrate how GlueWeb and its OCL parser work, let us suppose that we need to connect the *User Interface* with the *Business Logic* viewpoints, where both have been implemented with compatible platform technologies. Then, our tool analyses the XMI file of the *User Interface* PIM looking for transitions that invoke processes of the *Business Logic*. For every detected transition, the OCL parser finds the source page of the transition and includes in its implementation the call to the precise method that has been declared in the OCL constraint. This process is required to implement the bridges between the different viewpoints that we mentioned in previous sections.

The other kinds of conflicts cited in the *Designing Web Applications by Reusing Models from other Methodologies* section (i.e., name mismatches, different levels of abstraction and separation of mixed concerns) are solved by the QVT parser, which modifies the appropriate models to resolve the conflicts.

Proof of Concept: The Conference Review System

This case study was proposed for the *First International Workshop on Web Oriented Software Technology* (IWWOST 2001) and developed following several Web engineering methodologies (including UWE, WSDM, WebML, OO-H, and OOHDM, among others). To facilitate the exchange of ideas and the comparison of experiences among the different proposals, a common system specification was given to the authors (see http://www.dsic.upv.es/west/iwwost01/files/ConferenceReviewSystem.pdf).

Let us just remember that the purpose of the system is to support the process of submission, evaluation, and selection of papers for a conference. For the sake of simplicity, we will keep the example as small as possible, only considering the "manage conference" functionality carried out by the PC-Chair of the conference.

In order to support this functionality, we are going to delegate to OO-H/Visual Wade the design and PHP code generation of the *User Interface* level, while WebML/WebRatio will support the design and management of a MySQL database at the *Data* level. Finally, we shall use our framework to design the *Business Logic* level of this application, leaning on the Netsilon tool to assist the code generation process. Netsilon provides implementations for PHP and Java as two possible target platforms and also supplies Web developers

DOI: 10.1002/asi

with facilities to specify the behaviour of an application by means of the Xion action language (Muller et al., 2005).

The previous combination of tools and methodologies will allow us, on the one hand, to preserve the separation of concerns proposed by WEI and, on the other hand, to generate code reusing different models and tools. In particular, we will reuse the OO-H and WebML models for this example, faithfully taken from the proceedings of IWWOST'01, to represent the *User Interface* and Database designs of the application. The only task that is required is to specify the *Business Logic* layer with the behavior of the application. More precisely, the Netsilon design architecture groups in a single class diagram which WEI separates in two concerns: *Business Logic Structure* and *Internal Processes* models. However, this fact does not represent any problem from the WEI viewpoint because it is possible to move the WEI behaviour specification of each method (i.e., the Internal Processes description) to a Xion specification for the Netsilon class diagram.

The following subsections describe how to deal with each layer and how to connect, by means of bridges; both the models and the implementations generated using Visual Wade, Netsilon, and WebRatio. We assume at this point that readers are familiar with these tools and have a certain knowledge and understanding of their corresponding modeling languages. For further information about them the reader is referred to (Ceri et al., 2002; Gómez & Cachero, 2003; Koch, 2001; Muller et al., 2005).

Generating the Database With WebML and WebRatio

From the WebML viewpoint, the basic information objects and relationships that emerge from the case study specification are shown in the E/R diagram of Figure 9. The reader should note that this model is a faithful and accurate copy of the one presented by the WebML group at IWWOST'01 (Ceri, Fraternali, Matera, & Maurino, 2001).

Using this WebML design, it is possible to generate the database application code, and run it on a MySQL database management environment using the automatic database generation facility of WebRatio. For this purpose, a JDBC connection to MySQ and a properly configured data source must be available so that WebRatio finally produces an error-free database.

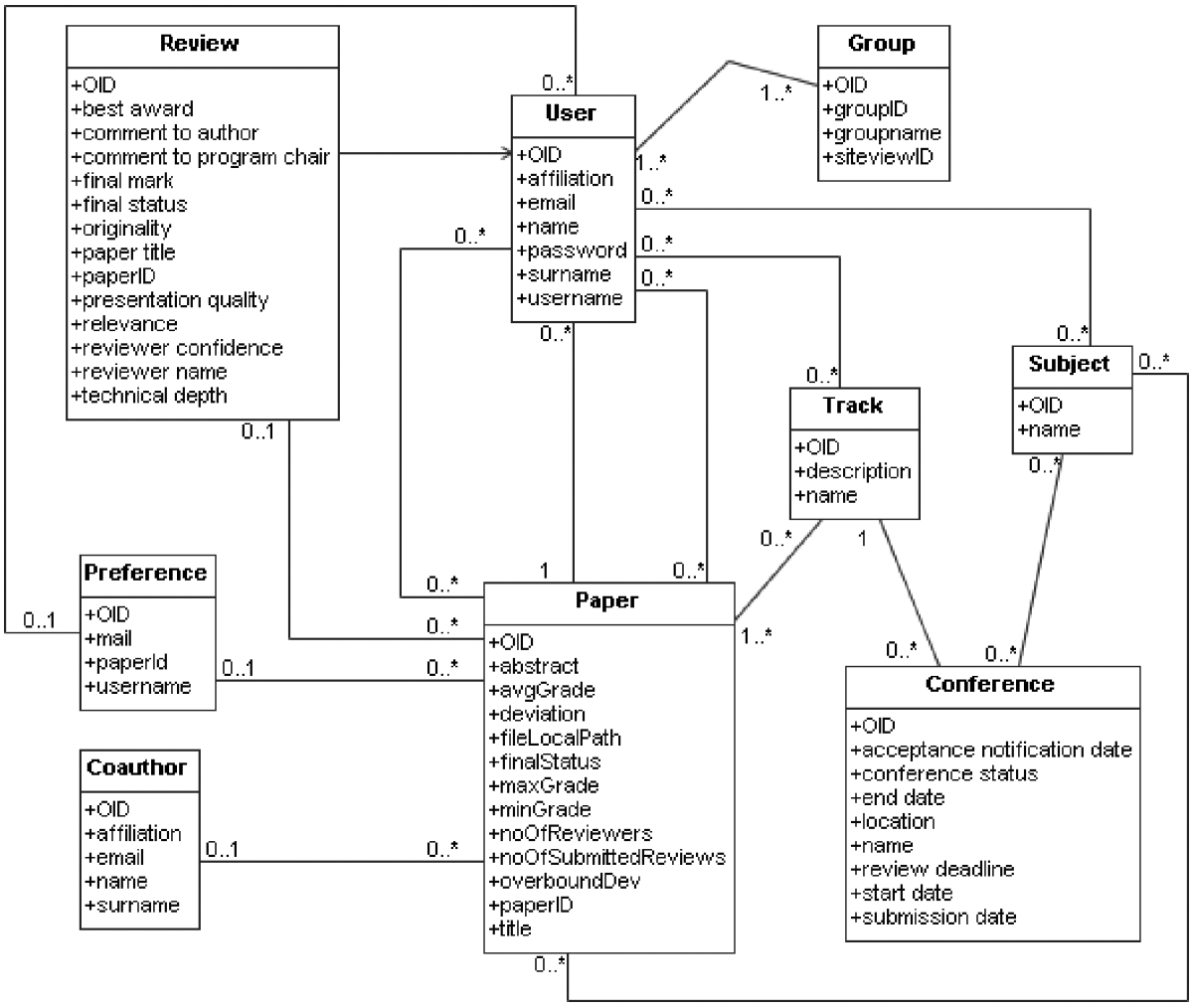


FIG. 9. The WebML PIM for the data layer.

DOI: 10.1002/asi

Generating the User Interface With OO-H and Visual Wade

The User Interface design is reflected in OO-H by a set of navigation Access Diagrams (NADs), one for each actor identified in the system. Because we are only focusing on the PC Chair user, Figure 10 shows the NAD diagrams corresponding to this actor of the system (taken from Cachero, Gómez, Párraga, & Pastor, 2001). Visual Wade not only provides a set of parameterized translators that enable the automatic generation of the PHP User Interface pages supporting the navigation but also prepares for the generation of connection modules by external modules, an aspect that will allow us to connect the code generated for the Business Logic layer with the User Interface implementation. Moreover, the OO-H method specifically provides mechanisms for describing filters that check and evaluate part of the business logic. In this way, an architecturally significant amount of business logic is executed on the client machine. However, we have opted for a thin Web client whereby all of the business logic will be executed on the server.

Using the model shown in Figure 10, Visual Wade produces a functional prototype of the application interface (see Figure 11) without needing to perform (at this point) a mapping process between navigation and the database schema. This does not mean that there is no need to maintain the consistency between the *User Interface* and the *Data* levels; on the contrary, integrating them requires that correspondences and relationships between the models be explicitly defined in the WEI *Conceptual* model.

At the last stage of the process, the abstract presentation diagrams (APDs) are collected by the Visual Wade model compiler which has knowledge of the target platform resulting in an operational PHP Web interface managed by a *Mediator Object*. The mediator manages user interface requests, and a request can consist of a transition of a navigation link or the invocation of a logic process. In the latter case, it recovers information about the objects involved and then sends the request to the corresponding business module, together with any other type of information of context (and the values of the parameters if needed). The specification of the behavior of these invocations is precisely the concern of the *Business Logic* layer described below.

Generating the Business Logic With WEI and Netsilon

Bearing in mind the correspondences between the OO-H and WEI models, we can take advantage of the OO-H conceptual domain diagram and use it as our WEI *Business Information* model because there is a direct relationship between both models. Furthermore, the WEI *Business Information* is a UML model, and therefore it can be directly fed into the Netsilon tool. In addition, Netsilon uses this model to automate Web specific concerns, such as session management, personalization, profiling, search, or statistics. Despite their potential interest we will not make use of such facilities here.

The implementation of methods is specified with the action language Xion, a platform independent action language that augments OCL with Java-like structures to describe the

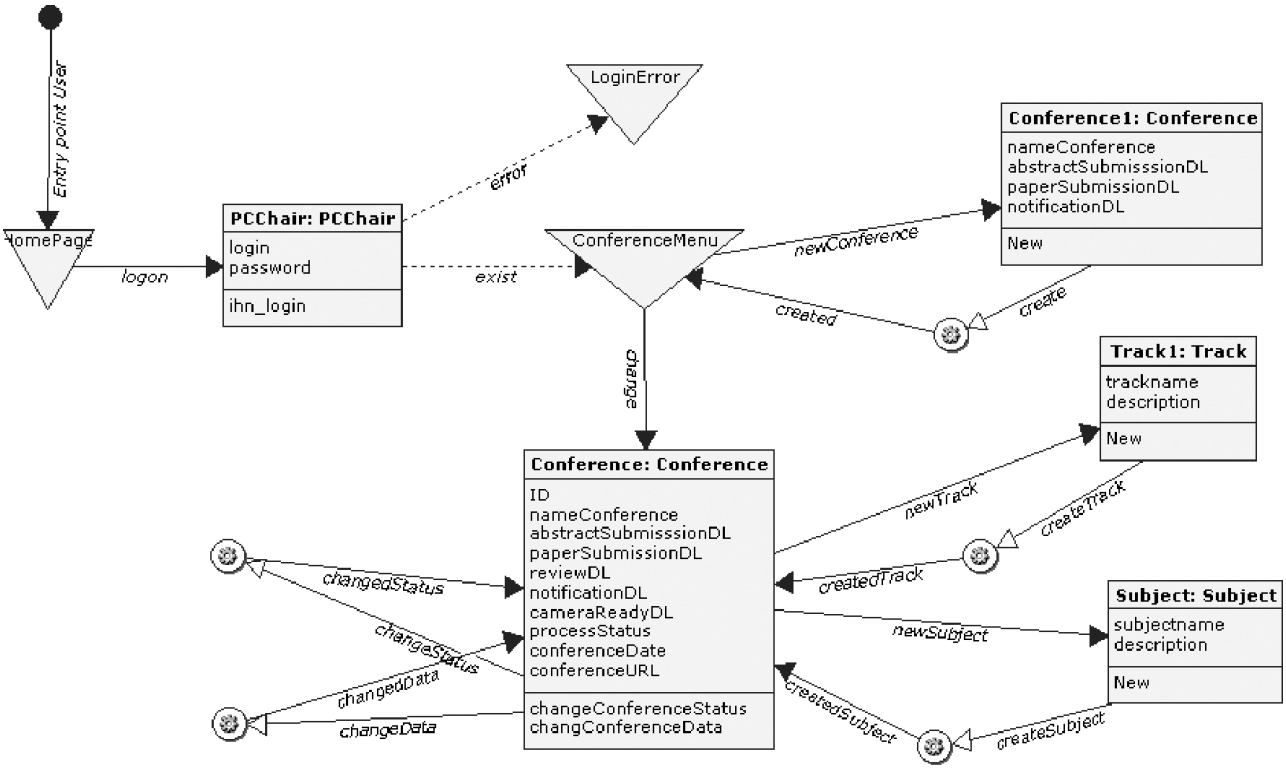


FIG. 10. The OO-H navigation access diagrams for the user interface.

DOI: 10.1002/asi

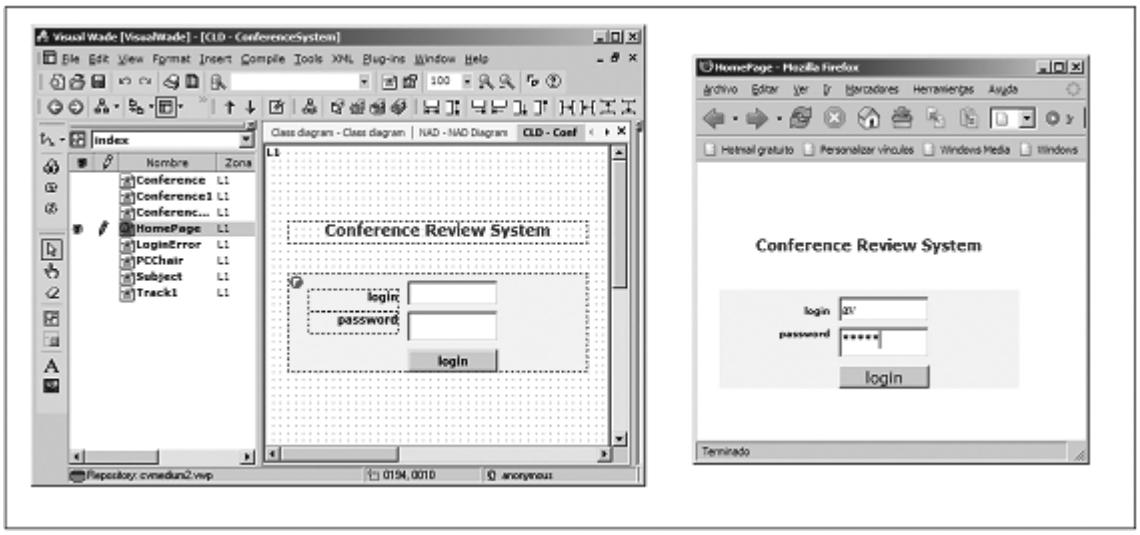


FIG. 11. The OO-H APD and Web page for the user interface.

behavior of the methods. Considering the Business Model shown in Figure 12, an example of Xion code for implementing the `login()` operation of the `PC_Chair` class is provided below. The purpose, here, is to verify that there is "only" one user using the `p_login` and `p_passwd` parameters as its login and password, respectively.

```
return PCChair.allInstances()->
  select(i:i.login==p_login&&i.passwd==
  p_passwd)->size()=1;
```

The following example shows the implementation of the `newConference()` method of the `Conference` class, illustrating the use of Xion's Java-like constructs such as conditional control blocks, return statements, and special variables such as this or self.

```
Set (CRS::Conference) c = CRS::Conference.
  allInstances();
if (c->size() = = 0) {
    this.name=p_name;
    this.abstractSubmissionDL=p_aDL;
    this.paperSubmissionDL=p_pDL;
    this.reviewDL=p_rDL;
    this.notificationDL=p_nDL;
    this.cameraReadyDL=p_crDL;
    this.processStatus=p_status;
    this.conferenceDate=p_date;
    this.conferenceURL=p_URL;
}
else {
    error("This system only support the man-
agement of ONE conference");
}
```

Please notice how in this way we are able to specify, in a platform independent manner, all the actions that need to be performed when "transitioning" between pages modeled at the *User Interface* level, as well as the verification of predicates that lead to the selection of a particular path between pages according to the current context and the requirements expressed in the Business model. From this point, the Netsilon tool can then generate and deploy the fully executable code for the PHP platform. In addition to a set of PHP files, one for each method of the *Business Structure Information* model, an administrator process is generated for the purpose of managing the objects stored in the database.

Now all that is left is to "glue" the three layers, that is, to define the bridges among the different levels and to transform those specifications to code.

Generating the Bridges

As mentioned in *A Generic Framework for Web Applications* section, the aim of the Conceptual model is twofold: first, it permits a common vocabulary shared among the different models, and second, it allows the description about how the different layers are related to each other.

By looking at the OO-H and WebML models (Ceri et al., 2001; Cachero et al., 2001), we find some name mismatches: what is called person or name at the user interface level, is named user and username, respectively, at the Data level. Analogously, some of the signatures of the methods specified at different layers do not match. These inconsistencies are the ones that need to be resolved by the bridges, which in this case were defined in terms of QVT relations that modified the code generated by Visual Wade and Netsilon, ensuring the use of common names and therefore resolving the name inconsistencies. An example of such bridges follows:

```
context UserInterface::Person
    inv: self.name.equivalent(Data::
      User.userName)
    inv: self.affiliation.equivalent(Data::
      User.affiliation)
...
```


DOI: 10.1002/asi

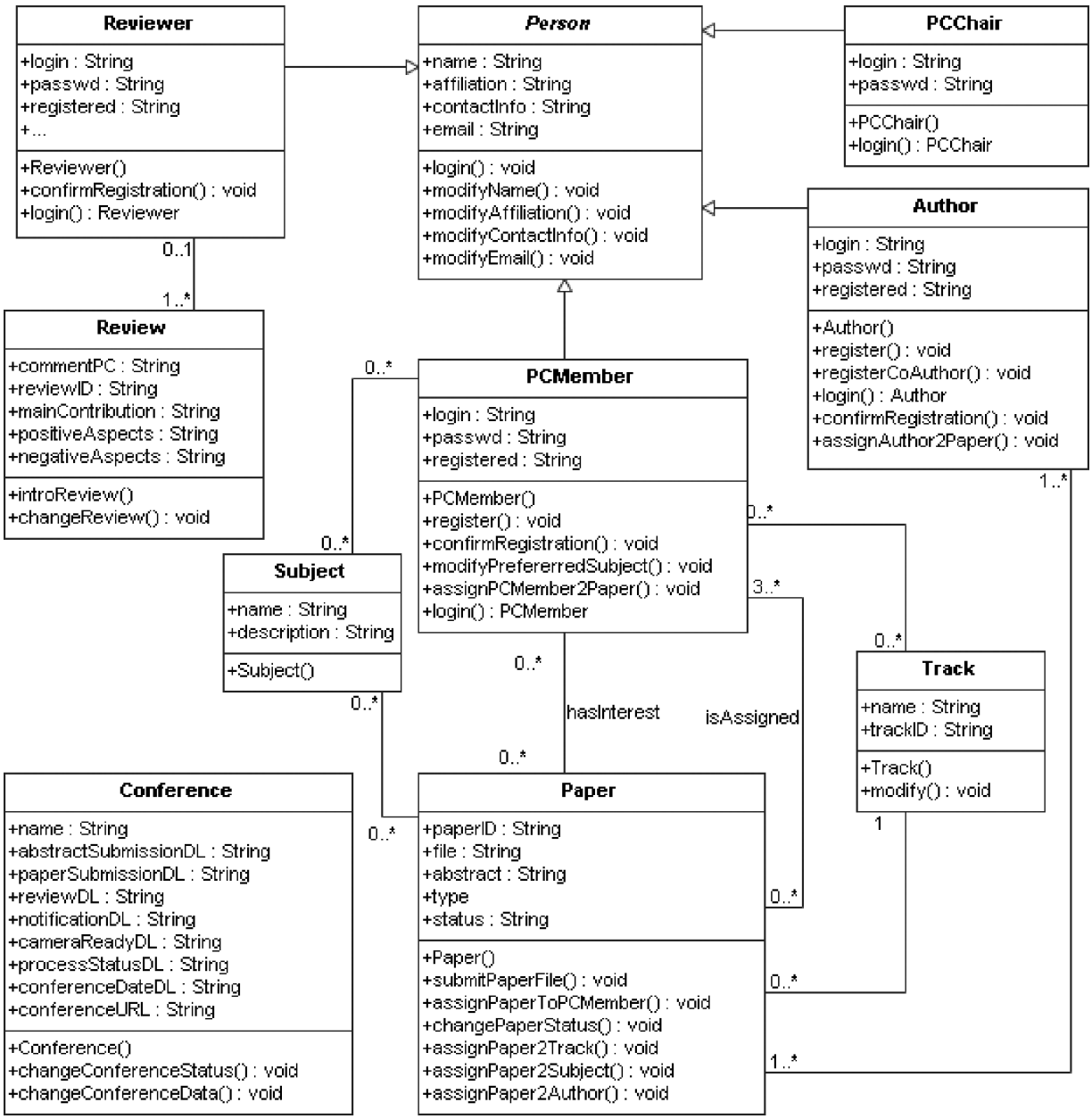


FIG. 12. The Netsilon representation for the WEI business Information model.

```
modeltype SC uses "ooh.OOH_Metamodel"
modeltype TG uses "webML.WebML_Metamodel"
...
relation Person2User{
   /* match Persons with Users */
   domain mdsc pe: Instance {
        classifier = "Person"
        name = X
        affiliation = Y
        email = Z
        ...
        }
   domain mdtg us: Instance {
        classifier = "User"
        name = X
        affiliation = Y
        email = Z
        ...
      }
}
```

At this point, the *GlueWeb* tool is used. The bridge definitions and the XMI representations of the three viewpoints of the *Conference Review* system are loaded into the tool, together with the information about their implementation (e.g., physical location in which implementation files are stored, the platform technology that has been used for it, the location of the actual models, etc.) Based on this information,

DOI: 10.1002/asi

GlueWeb performs the compatibility test of implementation platform technologies and then, if the results are positive, the *GlueWeb* QVT and OCL parsers read and process the bridge specifications. At the end of this step, we have a complete and functional implementation for the application under development.

Related Work

Our work is related to the emerging research area of Web engineering called model-driven web engineering (MDWE) and, in particular, to the works that try to use standards (e.g., UML, MDA, etc.) for Web application development.

First, we find some approaches that are defining UML-compliant DSLs (by means of UML profiles) for representing proprietary Web engineering modeling languages. This is the case of WebML, which has recently defined a metamodel and a UML profile (Moreno, Fraternalli, & Vallecillo, 2007; Schauerhuber, Wimmer, & Kapsammer, 2007) 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, having a metamodel for WebML will allow its integration with other MDA tools and also with other MDSD approaches and tools.

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 they then formulate their development processes in terms of model transformations and merges. Probably the most representative example is UWE, which has successfully restructured 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). This has significantly enhanced the original proposal with better modularity, expressiveness, and reuse. 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 successfully specifying the development process of Web information systems in terms of (meta)models and transformations between them (Cáceres, de Castro, Vara, & Marcos, 2006).

The reformulation of model-based Web engineering proposals is also providing 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 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, Kraus, Cachero, & Meliá, 2004). Their joint findings are very encouraging because they managed to define a common way of modeling them for both proposals. This shows that reuse of metamodels across Web engineering proposals is feasible. Finally, UWE has also shown recently how other concerns, such as the user requirements (Koch, Zhang, & Escalona, 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 reorganized their proposal as a set of separate models, related through model transformations. All these findings support the thesis that a common metamodel is possible for Web engineering, as originally proposed in (Koch, 2001), and as demonstrated by our proposal.

OOWS and OOHDM have also complemented their initial approaches to incorporate the MDA principles, models, and transformations. In the particular case of OOHDM, the changes have resulted in a new proposal called OOHDMDA (Schmid & Donnerhak, 2005). It proposes the behavioural OOHDM semantics specification to serve as a PIM, which is partitioned in conceptual and navigational PIMs. Additionally, a set of transformation rules describe the transformation from the PIM to a servlet-based PSM and also generates executable Java servlets and classes from the PSM model. Similarly, OOWS is also improving its original proposal to support a complete MDA-based process that allows the transformation of platform-independent models and the generation of code for the frontend of a Web application (Pastor, Fons, Pelechano, & Abrahao, 2006).

Last but not least is a study by the group of Alfonso Pierantonio at the University of L'Aquila (Cicchetti, Ruscio, & Pierantonio, 2006) that we find very interesting. The study shows how model weaving can be effectively used to specify

DOI: 10.1002/asi

and implement bridges, to connect the different artefacts 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 whose consistency is 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, in a similar way to the proposal we have presented here.

In summary, our work complements these approaches, providing a complete framework that can serve as an umbrella for all of them under which they can interoperate and exchange models and tools.

Conclusions

This article has presented a model-based interoperability framework for Web application design that may be considered as a step to harmonize as much as possible the models and semantics that current approaches use to address Web concerns. In this sense, the framework can be considered a common reference metamodel shared by most of the current Web methodologies. It provides mechanisms that enable the smooth integration of Web methodologies to be complementary in such a way that those concerns not covered initially by a methodology can be modeled in a next phase by transforming their models into their equivalent models in WEI and completing the original design within it.

Unlike other methodologies, WEI does not assume that the design of a Web application will always start from scratch. The separation of concerns and levels it distinguishes allow developers to use the framework also to design and implement applications by reusing models and tools (e.g., model compilers and code generation environments) coming from other Web methodologies. Regarding the reuse of models coming from other proposals, we think that the development of Web applications is a specific domain in which both model and code reuse can be successfully applied.

To be able to achieve this goal, we have identified and solved some of the issues that need to be addressed. The problems we have pointed out are similar to the traditional problems that appear when integrating models that represent different views of the same system. WEI distinguishes three complementary viewpoints and establishes correspondences between the appropriate viewpoint elements using the conceptual model. We have explored the use of MOF QVT relations for representing WEI correspondences showing that it can be expressive enough to represent them and to maintain the consistency between the different concerns.

Finally, tool support is another essential element of any software development methodology. Being MOF and UML-compliant allows WEI process to be fully supported by MOF and UML (generic) tools such as design and code generation environments. However, the majority of these CASE tools are not Web-specific and do not allow the smooth integration of all OMG tools (OCL, QVT, etc). Thus, we plan to continue actively working on the *GlueWeb* plugin in order to integrate all the notations and engines that we need.

Acknowledgement

This research is funded by the Spanish MCYT Project TIN2005-09405-02-01.

References

Baresi, L., Colazzo, S., Mainetti, L., & Morasca, S. (2006a). Web engineering: Model–based Web application development (pp. 303–334). Germany: Springer-Verlag.

Baresi, L., Colazzo, S., Mainetti, L., & Morasca, S. (2006b). W2000: A modelling notation for complex Web applications (pp. 335–408). Germany: Springer-Verlag.

Cáceres, P., de Castro, V., Vara, J. M., & Marcos, E. (2006). Model transformations for hypertext modeling on Web information systems. Proceedings of the ACM Symposium on Applied computing (SAC '06), pp. 1232–1239. NY.

Cachero, C., Gómez, J., Párraga, A., & Pastor, O. (2001). Conference review system: A case of study. Proceedings of the 1st International Workshop on Web-Oriented Software Technology (IWWOST'01).

Ceri, S., Fraternali, P., Matera, M., & Maurino, A. (2001). Designing multirole, collaborative Web sites with WebML: A conference management system case study. Proceedings of the 1st International Workshop on Web-Oriented Software Technology (IWWOST'01).

Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., & Matera, M. (2002). Designing data–intensive Web applications. San Francisco: Morgan Kaufmann.

Cicchetti, A., Ruscio, D. D., & Pierantonio, A. (2006). Weaving concerns in model based development of data-intensive Web applications. Proceedings of the ACM Symposium on Applied Computing (SAC '06), pp. 1256–1261. NY.

Fuentes, L., & Vallecillo, A. (2004). An introduction to UML profiles. UPGRADE, The European Journal for the Informatics Professional, 5(2), 5–13.

Gómez, J., & Cachero, C. (2003). OO-H method: Extending UML to model Web interfaces (pp. 144–173). Bloomington, IN: Idea Group Publishing.

Institute of Electrical and Electronics Engineers. (1990). Standard Computer Dictionary: A compilation of IEEE Standard Computer Glossaries. New York: IEE.

Koch, N. (2001). Software engineering for adaptive hypermedia systems: Reference model, modeling techniques and development process. Unpublished doctoral dissertation, Reihe Softwaretechnik- Trends, Munich.

Koch, N. (2006). Transformations techniques in the model-driven development process of UWE. Proceedings of the 2nd Workshop on Model-driven Web Engineering (MDWE 2006), Palo Alto, California.

Koch, N., Kraus, A., Cachero, C., & Meliá, S. (2004). Integration of business processes in Web application models. Journal of Web Engineering, 3(1), 22–49.

Koch, N., Zhang, G., & Escalona, M. J. (2006). Model transformations from requirements to Web system design. Proceedings of fhe 6th International Conference on Web Engineering (ICWE '06), (pp. 281–288), NY.

Meliá, S. & Gómez, J. (2006). The WebSA approach: Applying model-driven engineering to Web applications. Journal of Web Engineering, 5(2),121–149.

Mellor, S. J. & Balcer, M. (2002). Executable UML: A foundation for model-driven architectures. Boston, MA: Addison-Wesley.

Mens, T. & Tourwe, T. (2004). A survey of software refactoring. IEEE Transactions on Software Engineering, 30(2),126–139.

Miller, J. & Mukerji, J. (2003). The MDA guide (ab/2003-06-01). Object Management Group.

Moreno, N. & Vallecillo, A. (2005a). A model-based approach for integrating third party systems with Web applications. Proceedings of the 5th International Conference on Web Engineering (ICWE'05), pp. 441–452, Sydney, Australia.

DOI: 10.1002/asi

Moreno, N. & Vallecillo, A. (2005b). Incorporating cooperative portlets in Web application development. Proceedings of the 1st Workshop on Model-driven Web Engineering (MDWE 2005), (pp. 70–79), Sydney, Australia.

Moreno, N. & Vallecillo, A. (2005c). Modeling interactions between Web applications and third party systems. Proceedings of the 5th International Workshop on Web Oriented Software Technologies (IWWOST2005), Porto, Portugal.

Moreno, N., Fraternalli, P., & Vallecillo, A. (2007). WebML modelling in UML. IET Software, 1(3), 67–80,

Muller, P.-A., Studer, P., Fondement, F., & B´ezivin, J. (2005). Platform independent Web application modeling and development with Netsilon. Software and System Modeling, 4(4), 424–442.

Object Management Group. (2001). Model driven architecture. A technical perspective (ab/2001-01-01). Needham, MA, USA.

Object Management Group (2005a). MOF QVT final adopted specification (ptc/05-11-01). Needham, MA, USA.

Object Management Group (2005b). UML 2.0 Superstructure specification (formal/05-07-04). Needham, MA, USA.

Object Management Group (2006). OCL 2.0. (ptc/06-05-01). Needham, MA, USA.

Pastor, O., Fons, J., Pelechano, V., and Abrahao, S. (2006). In E. Mendes and N. Mosley (Eds.), Web engineering: Theory and practice of metrics and measurement for Web development (pp. 277–302). New York: Springer.

Raistrick, C., Francis, P., Wright, J., Carter, C., & Wilkie, I. (2006). Model driven architecture with eXecutable UML. Cambridge, UK: Cambridge University Press.

Schauerhuber, A., Wimmer, M., & Kapsammer, E. (2007). Bridging WebML to Model-Driven Engineering: From DTDs to MOF. IET Software, 1(3): 81–97.

Schmid, H. A. & Donnerhak, O. (2005). OOHDMDA–An MDA approach for OOHDM. Proceedings of the 5th International Conference on Web Engineering (ICWE'05), pp. 569–574, Sydney, Australia.

Shlaer, S. & Mellor, S. J. (1992). Object lifecycles: Modelling the world in states. Yourdon Press.

Vallecillo, A., Koch, N., Cachero, C., Comai, S., Fraternali, P., Garrigós, I., et al. (2007). MDWEnet: A practical approach to achieving interoperability of model-driven Web engineering methods. Proceedings of the 3rd International Workshop on Model Driven Web Engineering (MDWE 2007), Como, Italy.

DOI: 10.1002/asi