Improving Web Design Methods with Architecture Modeling [1]

Santiago Meliá[1], Jaime Gómez[1], and Nora Koch[2]

[1]Universidad de Alicante, Spain
{santi,jgomez}@dlsi.ua.es

[2]Ludwig-Maximilians-Universität München
and F.A.S.T GmbH, Germany
kochn@pst.ifi.lmu.es

Abstract. Many approaches have been developed for modeling the functional aspects of Web applications, but there is a lack of a modeling language for their architectural concerns. This paper proposes such a modeling language defined as a UML 2.0 profile, which allows the specification of domain-specific models for the architectural view of Web applications. The profile is part of the Web Software Architecture (WebSA) approach, which follows the Model Driven Architecture (MDA) principles. The modeling elements proposed for each WebSA model (subsystem, configuration and integration models) are both represented graphically and formalized by means of the profile and the metamodel, respectively. In this article we will focus on the Configuration model and how it is used to model the well-known Petstore example.

1 Introduction

In the Web domain, customers and users impose increasingly complex needs on the Web software being developed. In order to face such growing demands, during the last years the Web engineering community has proposed several languages, architectures, methods and processes for the Web. Among others, several methodologies, such as OO-H [2], UWE [11], WebML [3], have been proposed for the analysis and design of Web applications, and have shown their suitability for the specification of the functional requirements, in particular the navigational requirements posed by Web information systems. However, the design of architectural aspects of Web applications are almost always ignored, or postponed until the implementation phase with disadvantages related to scalability, platform-independence or security. Architecture models are fundamental in an MDA process, which consists in building and transforming platform-independent models and platform-specific models of the Web application. The objective is to generate only in the last steps platform-specific models and code. Such vision will have enormous consequences for the development and maintenance of the increasing amount of Web software that is being produced.

In order to overcome this lack of modeling elements for the early design of Web architectures the WebSA – Web Software Architecture – approach has been defined [12]. WebSA enriches Web engineering proposals with techniques for the development of software architectures for the Web and it is based on the Model Driven Architecture

[1] This research has been partially sponsored by the Spanish METASING (TIN2004-00779) and the EC 5th FP AGILE (IST-2001-32747).

(MDA) paradigm [14]. The approach proposes a set of architectural models and a set of transformations that permit the integration of these architectural models with a pre-existing functional model, defined by any of the above mentioned methodologies. The WebSA architecture models, namely the Subsystem Model, the Configuration Model and the Integration Model, provide a Web architecture perspective that includes the subsystems, Web components and connectors that make up the Web application.

The focus in this paper is set up on the modeling elements of the WebSA profile defined for the Configuration Model, which extends one of the new models in the UML 2.0 [17]: the composite structure. This model allows for a specification of software architecture following a properly component-based notation. The benefits of the composite structure are, by means of the Configuration Model, extended to the Web application domain. In addition, we show how to apply the profile to the architecture definition of the Petstore [19] Web application.

The rest of the paper is organized as follows: Sect. 2 provides a brief overview of the WebSA approach. Sect. 3 focuses on the metamodel and profile of the Configuration Model. Sect. 4 describes how the Configuration Model has been applied to model the architecture of the Web application Petstore and how this architectural model fits with traditional navigation models provided by Web design methods. Sect. 5 gives an overview of related work and finally, Sect. 6 outlines some conclusions and proposes further lines of work.

2 An overview of the WebSA Approach

WebSA is a proposal whose main target is to cover all the phases of the Web application development focusing on software architecture. It contributes to cover the gap currently existing between traditional Web design models and the final implementation. In order to achieve this, it defines a set of architectural models to specify the architectural viewpoint which complements current Web engineering methodologies such as [2, 11]. Furthermore, WebSA allows for the integration of the different viewpoints of a Web application by means of transformations between models.

The WebSA approach proposes three architectural models:

- **Subsystem Model (SM):** determines the subsystems that make up our application. It is mainly based on the classical architectural style defined in [1] – the so called "layers architecture" – where a layer is a subsystem encapsulating a certain level of abstraction. Furthermore, it makes use of the set of architectural patterns defined in [18] that determine which is the best layer distribution for our system.
- **Configuration Model** (**CM**): defines an architectural style based on a structural view of the Web application by means of a set of Web components and their connectors, where each component represents the role or the task performed by one or more common components identified in the family of Web Applications. This is explained with more detail in Sect. 3.
- **Integration Model** (**IM**): merges the functional and the architectural views into a common set of concrete components and modules that will make up the Web application. This model is inferred from the mapping of the components which are defined in the configuration model, the subsystem model and the models of the functional view.

The formalization of these models is obtained by means of a MOF-compliant [15] repository metamodel (part of the OMG proposed standards) that specifies (1) which is the semantics associated with each model element, (2) which are the valid configurations and (3) which constraints apply.

Furthermore, WebSA proposes a development process based on the *MDA development process* [10], which includes the same phases as the traditional life cycle (Analysis, Design, and Implementation). However, unlike in the traditional life cycle, the artifacts that result from each phase in the MDA development process must be a computable model. These models represent the different abstraction levels in the system specification and are, namely: (1) Platform Independent Models (PIMs) defined during the analysis phase and the conceptual design, (2) Platform Specific Models (PSMs) defined in the low-level design, and (3) code.

In order to meet these requirements, the WebSA development process establishes a correspondence between the Web-related artifacts and the MDA artifacts. As a main contribution, WebSA defines a transformation policy driven by the architectural viewpoint, that is, is an "architectural-centric" process [9] (see Fig. 1).

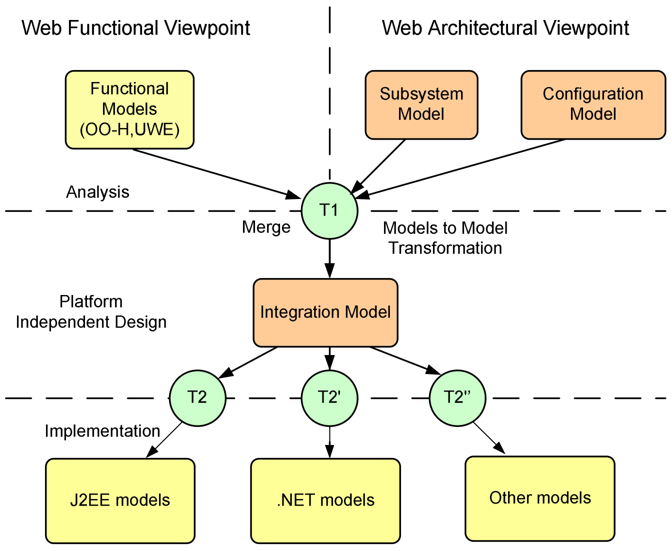


Fig. 1. The WebSA Development Process

Fig. 1 shows how in the analysis phase the Web application specification is divided vertically into two viewpoints. The functional-perspective is given by the Web functional models provided by approaches such as OO-H [2] or UWE [11], while the Subsystem Model (SM) and the Configuration Model (CM) define the software architecture of the Web Application. In the analysis phase, the architectural models are based on two different architectural styles to define the Web application. These models fix the application architecture orthogonally to its functionality, therefore allowing for their reuse in different Web applications.

The PIM-to-PIM transformation (T1 in Fig. 1) from analysis models to platform independent design models provides a set of artifacts in which the conceptual elements of the analysis phase are mapped to design elements where the information about functionality and architecture is integrated. The model obtained is called Integration Model (IM), which merges in a single architectural model the information gathered in the functional viewpoint (e.g., from Conceptual and Navigational models in OO-H and Conceptual, Navigational and Process models in UWE) with the information provided by the Configuration and Subsystem models.

It is important to note that the Integration model, being still platform independent, is the basis on which several transformations, one for each target platform (see e.g., T2, T2' and T2'' in Fig. 1). The output of these PIM-to-PSM transformations is the specification of the Web application for a given platform.

In the rest of the article we will focus on the Configuration Model since it represent the core of the WebSA architectural viewpoint.

3 Configuration Model (CM)

The Configuration model defines an architectural style based on the structural view of the Web application by means of a set of Web components and their connectors, where each component represents the role or the task performed by one or more common components identified in the family of Web applications. In this way, CM uses a topology of components defined in the Web application domain, and this allows us to specify the architectural configuration without knowing anything about the problem domain. At this level, we can define architectural patterns for the Web application as a reuse mechanism.

A diagram for the Configuration model is built by means of a UML 2.0 Profile of the new composite structure model, which is well-suited to specify the software architecture of Web applications. The main modeling elements of the CM are *WebComponent, Web Connector, WebPart* and *WebPattern*.

In order to formalize the Configuration model elements and their relationships, we define the Configuration metamodel (see Fig. 2).

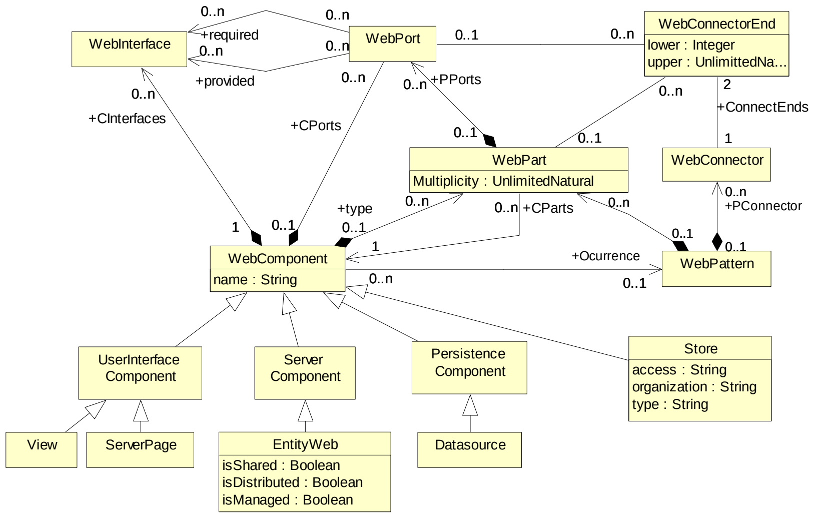


Fig. 2. Simplified CM Metamodel

3.1 WebComponent

A WebComponent represents an abstraction of one or more software components with a shared functionality or role in the context of a Web application. For example, a ClientPage is a WebComponent that contains presentation data and/or user interaction code. Note how a Web component does not necessarily map to a single physical page but reflects a general task that must be performed by the application, such as showing certain information to the user. The most important properties of a WebComponent are defined by the classes WebPort, WebInterface and WebPart.

The WebComponent is the root class of a type hierarchy that represents the different roles or tasks that may be performed by the components identified in the family of Web Applications. For example, the subclass EntityWeb is an object representing a concept of the application domain (see Fig. 2). In addition to the subtypes of WebComponent, which are shown in Fig. 2, the Petstore example (Sect. 4) will in addition use the following subtypes: ProcessComponent, UserAgent, DAC, LegacyView, Controller, View and EntityData. The complete topology of the WebSA components can be seen in [13].

3.2 WebPort

WebPort is an interaction point between a WebComponent and its environment. It decouples the internals of the component from the interaction with other components, making that component reusable in any environment that conforms to the interaction constraints imposed by its WebPorts. In this way, a WebComponent can only communicate with the outside through its WebPorts.

3.3 WebInterface

WebInterface represents the functionality the component to which it is associated offers to or requires from the rest of the system in order to be able to perform its task. Each WebInterface is associated with a WebPort specifying the nature of the interactions that may occur over this WebPort (see Fig. 2). On the one hand, the required interfaces of a WebPort characterize the requests which may be made from the WebComponent to its environment. On the other hand, the provided interfaces of a WebPort characterize requests the environment makes to the WebComponent.

3.4 WebConnector

WebConnector specifies a link that allows the communication in the system between two or more WebComponents or/and WebParts of the WebComponents (see 3.6). This communication is established through the WebPorts. However, in the case of a WebPart this relationship may affect either a WebPort or the whole WebPart. Each WebConnector has associated two WebConnectorEnds (see Fig. 2).

3.5 WebConnectorEnd

WebConnectorEnd represents an endpoint of the connector that attaches the connector to a WebPort or a WebPart. The WebConnectorEnd has two properties: (1) *lower* which specifies the lower bound of elements which could be connected with the WebConnectorEnd. (2) *upper* which specifies the upper bound of elements which could be connected with the WebConnectorEnd.

3.6 WebPart

WebPart represents a set of instances that are owned by composition belonging to a WebComponent instance. A WebPart has a property multiplicity, which using the notation [x{…y}] specifies the initial instance or the amount of instances (x) when the WebComponent is created, and the maximum amount of instances at any time (y).

3.7 WebPattern

WebPattern represents a Web architectural pattern, which is specified by a composite element made up of a set of WebConnectors, and WebParts that corresponds to Web components playing roles to accomplish a specific task or function. WebPattern instances are elements of reuse in a configuration model. For example, the Petstore application has two WebPatterns called *MVC2* (see Sect. 4.1) and *Façade* (see Sect.

4.2) which contain some possible configuration of elements that represent the patterns Model-View-Controller [1] and Façade [6].

In order to represent the architectural style defined by the Configuration Model, the CM Profile has been defined as an extension of the UML Composite Structure model including Web components and properties of the Web application domain.

In this way, the CM profile has incorporated all the classes of its metamodel as stereotypes, extending the UML metaclasses. The CM stereotyped classes will add the domain specific semantic defined in the Configuration metamodel to the semantic that they inherit from the UML metaclasses.

For the visual representation of the CM profile elements we stick to the notation of the corresponding UML metaclass elements. These modeling elements are described in Table 3.

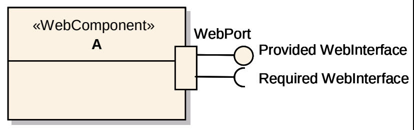
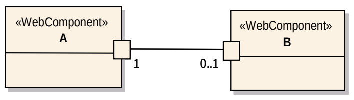
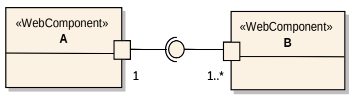
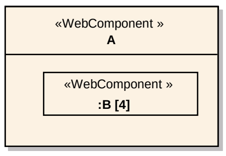
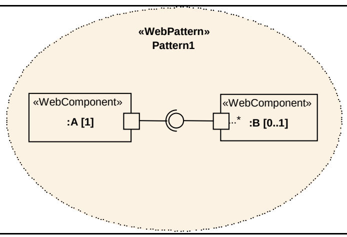
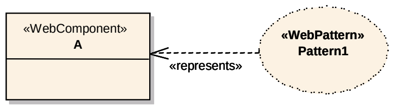
 «WebComponent» A WebPort Provided WebInterface Required WebInterface	**WebComponent** keeps the notation of UML structure class. It has incorporated the WebPort, by means of a small square on the boundary. **WebPorts** are associated to required or provided **WebInterfaces** with the lollipop notation. All subtypes of WebComponent, such as ClientPage, ServerPage and EntityWeb (Fig. 4) are represented using this notation.
	**WebConnector** establishes the communication directly between the WebPorts of WebComponents or/and WebParts. This connector is represented with the notation of a UML association.
	**WebConnector** is attached to two WebPorts which has required attached by two compatible WebInterfaces – one required interface and one provided interface – that are compatible. This connector is called assembly.
	**WebPart** is shown as a box inside a WebComponent or a WebPattern. As stated in Sect. 3.6, a multiplicity for a WebPart can be specified within the container WebComponent.
 «WebPattern» Pattern1 «WebComponent» :A [1] «WebComponent» :B [0..1]	**WebPattern** is represented as a UML collaboration with a dashed ellipse icon containing the name of a WebPattern. The internal structure of a WebPattern comprises WebParts and WebConnectors. It is shown in the compartment within the dashed ellipse icon.
 «WebComponent» A «represents» «WebPattern» Pattern1	A dashed arrow with a stick arrowhead and labelled with the keyword «represents» means that a **WebPattern** is used in a **WebComponent.**

Table 3. Notation used in a Configuration Model

4 A Case Study: Petstore

For the proof-of-concept of the CM profile, we have chosen the J2EE Petstore example [19]. This application constitutes a blueprint that uses best practices and design guidelines for a distributed component Web application.

As stated above, the CM represents an architectural style and it is made up of a set of Web components and their connectors. This model is independent of the application functionality and the development platform. Therefore in this article we will only focus on its architectural aspects. We first give an overview of the Petstore configuration model and then dive into two applied patterns *MVC2* and *Façade*.

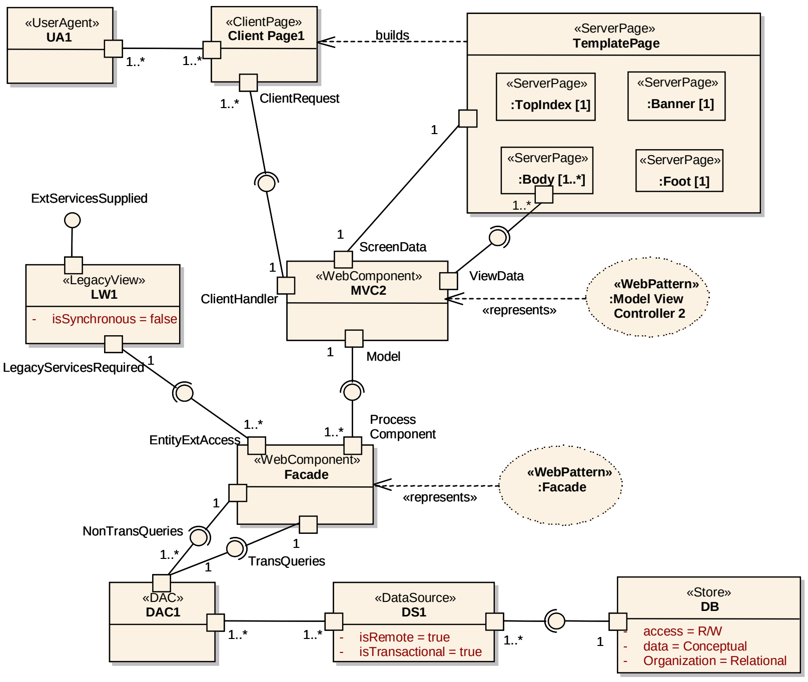


Fig. 4. Configuration Model of Petstore

Fig. 4 shows a general view of the CM representing the Petstore architecture, which is made up of a set of components and connectors that are described next.

In the front-end part of the model we find the UserAgent (e.g., a browser) which receives the user's requests and renders the ClientPage set. Each ClientPage contains the interface and functionality information and is responsible for sending messages to the *MVC2* WebPattern (described in detail in Sect. 4.1). The *MVC2* WebPattern receives the requests through the WebPort *ClientHandler* and establishes the interface reaction through the WebPort *ScreenData*, which is defined by the ServerPage components.

The ServerPages of Petstore are specified following the pattern *Master Template* defined by Conallen [4]. Following this pattern, in Fig. 4 we have defined a *TemplatePage* that builds the client pages by instantiation of the WebParts *TopIndex*, *Banner*, *Foot* and *Body*. Each instance of a Body ServerPage needs an interface to access the required data objects. Such interface is provided by the WebPort *ViewData* of the *MVC2* Web Pattern. Looking at the MVC2, we can observe that the MVC2 component needs information from the components that implement the business logic, which is obtained through the *BLogic* interface offered by the *Façade* WebPattern (described in detail in Sect. 4.2). The *Façade* invokes the DAC (Data Access Component, based on the Data Access Object pattern [6]), which contains the data access methods and decouples the business logic from the data. In our example DAC offers two interfaces, one for the non transactional queries, i.e. the data retrieval queries which are accessed through the WebPort process component of *Façade*, and one for the transactional queries (insert, update and delete) which are accessed through the *Entity* port of *Façade*. The WebComponent *Façade* is in turn related to the component LegacyView, which offers a series of services coming from the *EntityExtAccess* port to other applications and converts the received asynchronous calls into requests to the business logic. Finally, the specified remote and transactional data sources allow for the connection to Store that contains the information modelled in the conceptual model of the functional view of the Web application Petstore, and specifies a read/write access, as well as a relational organization.

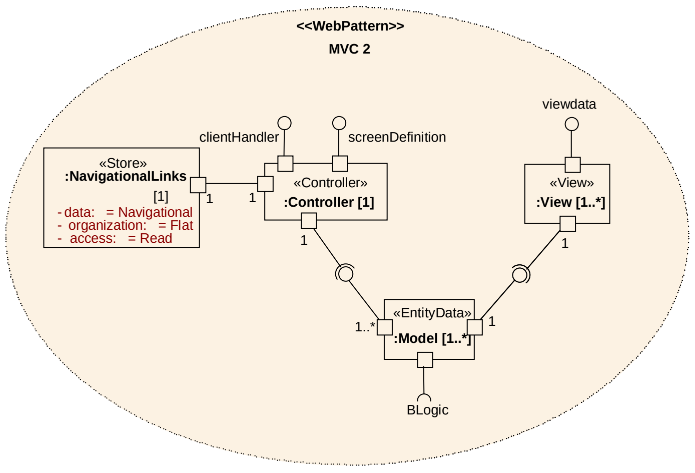


Fig. 5. Model-View-Controller 2 Pattern

4.1 Model View Controller 2 Pattern

Fig. 5 depicts the components of the *MVC2* WebPattern, a variant of the classic MVC. This pattern is made up of a controller component that has two Web Interfaces, one for receiving the requests from the client page and one for building the pages. It is connected to a Store component which contains information about the links among pages. The fact that links are contained in a Store supports the separation of navigation from presentation. Also, the controller needs the information contained in the component EntityData, which contains both data coming from the classes of the conceptual model and the connectivity through the BLogic WebInterface.

4.2 Façade Pattern

The Façade WebPattern (see Fig. 6) includes a set of stateless ProcessComponents (e.g., a Session Stateless EJB), which receives the requests through the *BLogic* WebInterface from the *MVC2*, and resends them to the Entity through the interfaces *createEntity* and *invokeEntity*.

This pattern requires an interface to DAC through the *nonTransactionalQueries* interface. Also, it has a set of EntityWeb components that represent the elements of the domain in the business logic. These have the tagged value isShare=true indicating that they can be shared by multiple transactions and users (e.g., it could be implemented by an EJB Entity). Each EntityWeb contains an EntityData representing the state of the domain elements, as defined in the ObjectValue pattern [6]. Note how this entity provides the *ExtEntityServices* interface for the View Legacy and sends the requests to the data layer through the *TransactionalQueries* WebInterface.

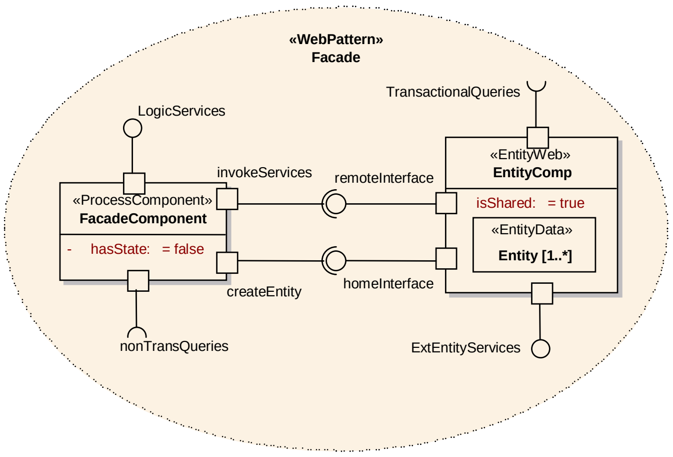


Fig. 6. Façade Pattern

5 Related Work

The modern Web architecture emphasizes scalability, independent deployment, and interaction latency reduction, security enforcement, and legacy systems encapsulation. For instance, approaches such as Representational State Transfer (REST) [5] architectural style, to represent Web architectures, with focus upon the generic connector interface of resources and representations. However, REST has only served both as a model for design guidance, and as test for architectural extensions to the Web protocols. WebSA is based on some concepts of the REST architecture style to define a process development for the production of Web applications.

Hassan and Holt [7] present an approach aimed at recovering the architecture of Web applications. The approach uses a set of specialized parsers/extractors that analyze the source code and binaries of Web applications. They describe the schemas used to produce useful architecture diagrams from highly detailed facts. Conversely, WebSA follows the opposite process that goes from the representation of the architecture to the implementation. However, this approach does not describe the transformation rules to realize this reverse engineering process.

Conallen's work is another well-known approach on extending UML [4] to model the design of Web applications. Conallen presents the Web Application Extension (WAE) for UML, which generates de skeleton code for a Web application. Unlike this approach, WebSA represents the software architecture of Web applications at different levels of abstraction, and this allows for a better scalability and reusability improving the productivity on the development of Web applications.

In the Web Engineering community moreover different approaches have been developed which tackle the Web software architecture. OOHDM-Java2 [8] proposes a product line architecture in J2EE for simplifying the systematic construction of different families of applications. However, this complex architecture is only useful for J2EE platforms. Another approach is proposed by WebRatio – the WebML tool [3]. WebML is a modeling language based on the entity relationship of data that provides its own notation and proposes a static architecture based on the J2EE struts framework.

6 Conclusions and Further Work

WebSA is an approach that complements the currently existing methodologies for the design of Web applications with techniques for the development of Web architectures. WebSA comprises a set of models and transformations, a modeling language and a development process. The development process also includes the description of the integration of these architectural models with the functional models of the different Web design approaches setting the base for an MDA

In this paper we have presented a UML 2.0 Profile for WebSA. We want to stress the importance of the UML compliance of this modeling language, which allows the use of any UML 2.0 CASE tool. Furthermore, it is the basis for the specification of the transformations that rely on the QVT standard.

Currently, we are working on the set of QVT [16] transformation models to support the WebSA refinement process. This work will formalize the transformations while guaranteeing the traceability between those models and the final implementation.

References

1. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture – A System of Patterns, John Wiley & Sons Ltd. Chichester, England, 1996
2. C. Cachero. OO-H. Una extensión de los métodos OO para el modelado y generación automática de interfaces hipermediales http://www.dlsi.ua.es/ ~ccachero/pTesis.htm, 2003
3. S. Ceri, P. Fraternali, M. Matera. Conceptual Modeling of Data-Intensive Web Applications, IEEE Internet Computing 6, No. 4, July/August 2002, 20–30
4. J. Conallen. Building Web Applications with UML, 2nd Edition, Addison-Wesley Longman, September 2002
5. R. Fielding, R. Taylor. Principled Design of the Modern Web Architecture, ACM Transactions on Internet Technology 2(2), May 2002, 115-150
6. E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995
7. A. Hassan, R. Holt. Architecture Recovery of Web Applications, International Conference on Software Engineering (ICSE'02), Orlando, Florida, May 2002
8. M. D. Jacyntho, D. Schwabe, G. Rossi. A Software Architecture for Structuring Complex Web Applications, Journal of Web Engineering, 1(1),37-60, 2002
9. I. Jacobson, G. Booch, J. Rumbaugh. The Unified Software Development Process, Addison-Wesley, 1999
10. A. Kleppe, J. Warmer, W. Bast. MDA Explained. The Model Driven Architecture, Practice and Promise, Addison-Wesley, 2003
11. N. Koch, A. Kraus. The Expressive Power of UML-based Web Engineering, In Proc. of the 2nd. IWWOST, CYTED, Málaga, Spain, June 2002, 105-119
12. S. Meliá, C. Cachero. An MDA Approach for the Development of Web Applications, In Proc. of 4th ICWE, LNCS 3140, July 2004, 300-305
13. S. Meliá. The WebSA Configuration Model Profile. Technical Report TR-WebSA2, http://www.dlsi.ua.es/~santi/pPublicaciones.htm, November 2004
14. OMG. Model Driven Architecture, OMG doc. ormsc/2001-07-01
15. OMG. Meta Object Facility (MOF) v1.4, OMG doc. formal/02-04-03
16. OMG. RFP: MOF 2.0 Query / Views /Transformations, OMG doc. ad/2002-04-10
17. OMG. UML 2.0, Final Adopted Specification, OMG doc. ptc/2003-08-02
18. Klaus Renzel, Wolfgang Keller. Client/Server Architectures for Business Information Systems: A Pattern Language, PLoP Conference, 1997
19. TM J2EE Blueprint. Java Petstore 1.1.2, http://developer.java.sun.com/developer/releases/petstore/petstore1_1_2.html, November 2004