Journal of Web Engineering, Vol. 5, No.2 (2006) 121-149
© Rinton Press

THE WEBSA APPROACH: APPLYING MODEL DRIVEN ENGINEERING TO WEB APPLICATIONS

SANTIAGO MELIÁ and JAIME GOMEZ

Universidad de Alicante, Spain

{santi, jgomez}@dlsi.ua.es

Received December 22, 2005
Revised April 26, 2006

The Web engineering research community has proposed several Web design methods that have proven successful in the specification of the functional aspects (such as domain, navigation and presentation) posed by Web applications. However, the architectural aspects are often ignored in the design and the Web application is not specified enough. This development process produces a gap between the Web design models and the final implementation. To overcome this limitation, we extend the different Web methodologies with a generic approach called WebSA. WebSA is based on the Model Driven Engineering (MDE) paradigm that promotes models as the primary artifacts needed to carry out a software project from beginning to end. To do this, WebSA proposes a Model Driven Development made up of a set of UML architectural models and QVT model transformations as a mechanism for (1) integrating the functional aspects of the current Web methodologies with the architectural models as well as for (2) defining a set of transformations from the architectural models to platform-specific models such as J2EE, .NET, etc. To illustrate our approach, in this paper we combine WebSA with the OO-H method, to tackle the design of a running example such as the Travel Agency system.

Keywords: Web Engineering, Model Driven Engineering, Software Architecture, Transformations, UML, QVT, Architectural Patterns
Communicated by: N Koch

1 Introduction

The Web Engineering community is well aware that, in order to keep track of changes and assure the feasibility of applying their methods to commercial Web applications, the different proposals that should now incorporate the explicit consideration of architectural features in the Web application design process need to be carried through to a more complete development. In order to do so, several authors propose the use of well- known techniques in the Software Architecture discipline [2] to identify and formalize which subsystems, components and connectors (software or hardware) should make up the Web application. As Booch [4] has stated, the presence (or absence) of a meaningful architecture is an essential predictor of the success of a Web application for two reasons: first of all, the creation of a stable architecture helps cut the highest risks out of the project. Secondly, the presence of a stable architecture provides the basis upon which the system may be continuously developed with minimal scrap and rework.

The integration of architectural features into the different Web proposals provides a closer match between the system modeled and the final implementation of the Web application. This feature is especially important in methodologies that provide a code generation environment such as WebML [7], OO-H [11], etc. The inclusion of one such model would therefore decrease the set of predefined architectural decisions that are usually taken in generating the code in such environments. These are decisions that are often not the most appropriate as regards the solution sought by the customer. Moreover, the addition of an architectural viewpoint would provide a mechanism for discussing, documenting and reusing (by means of pattern catalogs) the architectural decisions which answer the different non-functional user requirements.

For this purpose, we propose the WebSA (Web Software Architecture) [18] [19] approach based on the MDE (Model Driven Engineering) paradigm [3] and more specifically on the OMG's Model Driven Architecture (MDA) framework [21] [22]. Basically, WebSA provides the designer with a set of architectural models and transformation models for specifying a Web application. Starting from these models, the designer can integrate the Web functional models (domain, navigation and presentation) with the architectural models, applying a set of model transformations. The result is the Integration model, which is a platform independent model that can be transformed into the different platforms such as J2EE, .NET, etc. For defining these transformations, there are several initiatives related to the MDA approach- the new Draft Adopted Specification for a Query/Views/Transformations (QVT) [24] language, among others. QVT is, in our opinion, the most interesting one, as it is a well defined language and has a graphical as well as a textual notation.

This paper explains each of the steps in the WebSA process by means of a running example: the Travel Agency system. To do this, WebSA has been combined with a well-known Web design method called OO-H. It allows us to merge the architectural models into the Web functional models which have been applied in many real Web applications. It can be seen in [10].

In order to understand this process, we should first give an overview of the WebSA approach in section 2 and introduce the user requirements of the Travel Agency in section 3. Thus we start the process with the analysis phase. On the one hand, section 4 presents the functional viewpoint provided by the domain and navigation models of the OO-H approach. And on the other hand, section 5 presents the architectural viewpoint with the most important model in the analysis phase of WebSA, the Configuration model. Once the analysis models are defined, section 6 proposes the specification of the transformation which for its part specifies the merging of the functional and the architectural models in the QVT language. As a result of the first transformation, the Integration model is obtained and described in section 7. The last step is explained in section 8 and shows how a second transformation is defined in QVT, to convert the Integration model into a J2EE implementation. Finally, the relevant related work and the future lines of research are outlined in section 9 and 10, respectively.

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 and to contribute to covering the gap existing between traditional Web design models and the application implementation. In order to achieve this, it defines a set of architectural models (see section 2.1) for specifying the architectural viewpoint which complements current Web engineering methodologies such as [11], [16].

Furthermore, WebSA defines an instance of the *MDA Development Process* [14] for the Web application domain, which allows for the integration of the different viewpoints of a Web application by means of transformations between models (see section 2.2). As the reader may already know, MDA is an initiative that can be defined as the realization of MDE principles, that is, the models as the primary artifacts needed to carry out a software project, from the definition of the requirements to the representation of transformations. In addition, MDA proposes a framework around a set of OMG standards (such as MOF, UML, XMI, etc.). This standardization allows us to specify the WebSA models in different tools, interchange the metamodels and models with different approaches and reduce the learning curve of the stakeholders. Consequently, MDA paves the path for the development of WebSA which includes model compilation capabilities and processes based on extreme programming.

2.1 WebSA Architectural Models

The WebSA approach proposes three architectural models:

- **Subsystem Model (SM):** determines the subsystems that make up our application. It is based mainly on the classical architectural style defined in [5] – 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 [26] that determines what the best layer distribution for our system is.
- **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 in more detail in section 5.
- **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. This is explained in more detail in section 7.

The formalization of these models is obtained by means of a MOF-compliant [23] repository metamodel and a set of OCL constraints (both part of the OMG proposed standards) that together specify (1) what the semantics associated with each model element are, (2) what the valid configurations are and (3) which constraints apply.

2.2 WebSA Development Process

The WebSA Development Process is based on the *MDA development process*, 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 computable models. 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. Another major contribution is that WebSA defines a transformation policy driven by the architectural viewpoint, that is, an "*architectural-centric*" process [13] (see figure 1).

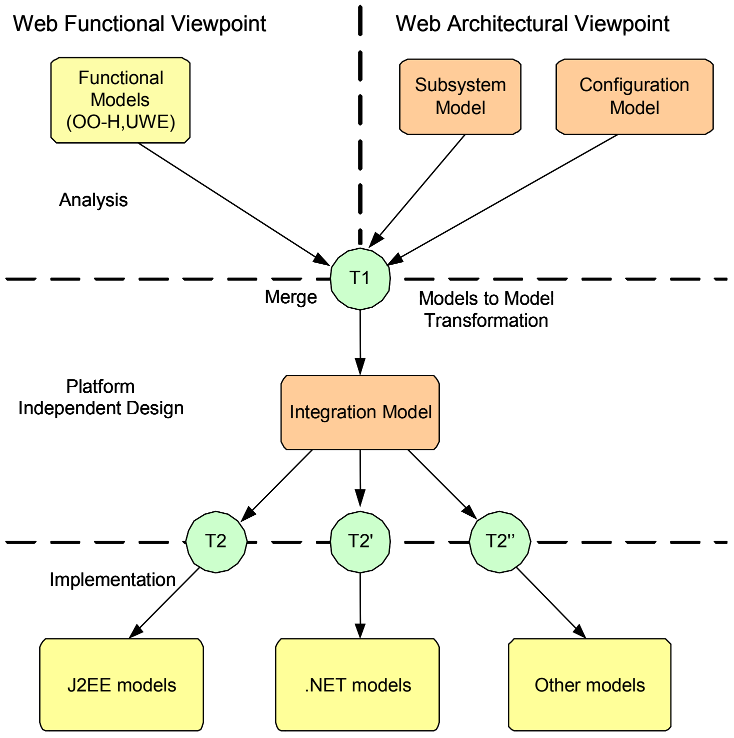


Figure 1. The WebSA Development Process

Figure 1 also 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 [11] or UWE [16], 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 propose two new architectural styles for the representation of the Web application. As defined in [5], "*an architectural style is independent from its realization, and does not directly refer to a concrete application problem it is intended to solve*". The specific architectural styles introduced by WebSA are more suitable for the Web applications than the pre-existing general architectural styles because they use a set of specific elements of the Web domain. In this way, these models fix the application architecture orthogonally to its functionality, thereby allowing for their reuse in different Web applications.

The PIM-to-PIM transformation (T1 in figure 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 brings together, in a single architectural model, the information gathered in the functional viewpoint and 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 figure 1), can be defined. The output of these PIM-to-PSM transformations is the specification of the Web application for a given platform.

The inclusion of an architectural perspective in this process plays a pre-eminent role in the completion of the specification of the final Web application, and drives the refinement process from analysis to implementation.

2.3 WebSA Profile

The WebSA approach is completed by the standardization which is performed by the definition of a UML Profile, just as other Web methodologies [16] have done previously. The WebSA Profile comprises a profile for each model: (1) SM Profile, (2) CM Profile and (3) IM Profile. Figure 2 depicts these profiles and the dependency between the CM and IM packages because both models share architectural elements as WebComponents, WebConnectors, etc. However, SM profile is defined with different artifacts as Subsystems which are later converted into IM elements by the T1 transformation. Finally, it also shows how the WebSA profile has been applied to the Travel Agency running example.

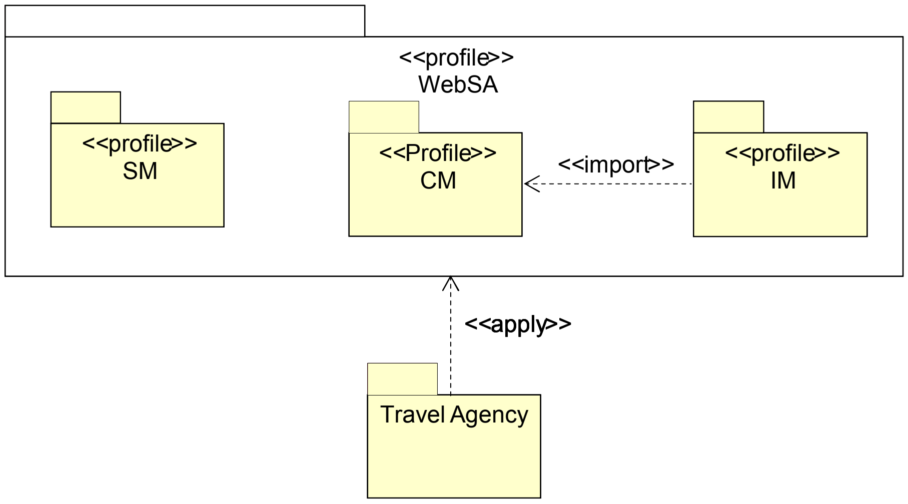


Figure 2. The WebSA Profile

The WebSA Profile focuses on:

- Providing modeling elements for a particular platform or domain. The WebSA domain is the set of components of the family of Web applications.
- Adding information that can be used when transforming one model to another model or code. On the one hand, the CM Profile and the SM profile provide the information for the transformation rules of T1 (see figure 1), which allow for the integration of the Web functionality with the architectural aspects. The result of this integration is the IM model. On the other hand, in the T2 transformation (see figure 1) the IM Profile provides the information needed to obtain the different PSMs models.

To show the usefulness of the WebSA approach, we have chosen the travel agency system proposed in the workshop Model Driven Web Engineering 2005 [30]. The next section illustrates the functional and the accessibility requirements of the travel agency system.

3 A Running Example: The Travel Agency System

To help make our approach understandable, we present an example that describes the process followed by a travel agency for selling trips to its customers. It is a Web Application that provides these services electronically. To make matters simple, we will concentrate on just Transport services, which can be carried out using planes, trains, cars, boats, or combinations of these. Accommodation, tourist packages, excursions, and other services also sold at travel agencies are not gone into here. The Web travel agency is a system of open distribution, because it uses some services from other legacy systems in fulfilling customer requirements. It is worth highlighting that it is only the legacy connection that is designed - it is assumed that these legacy systems exist previously.

At this point we bring in the functional requirements dictated by the client. The system starts when the Customer provides a description of the trip required to the Personal Travel Assistant, including personal constraints and preferences. The trip description contains the cities of origin and the destination, as well as the dates of departure and return. For one-way trips, only the places of departure and the dates are required. Constraints on the trip may include limits on the total price of the trip, duration of the trip itself, and any undesired Transport method (e.g. the customer may not like planes). Preferences may include the preferred means of transport for each way, as well as rules that can be applied to the set of acceptable service offers, thereby yielding an ordered sequence of service offers.

Once the Personal Travel Assistant receives the request from the Customer, it checks that it is well-formed, then selects the Broker Agents that work with the agency and who can service the trip. The Personal Travel Assistant also interacts with each Broker Agent, asking them for an offer that meets the demands of the Customer’s requested trip.

Each Broker Agent may work with several Transport Companies, asking them to provide an offer for the service requested. If the offer matches the customer requirements, the Broker Agent will ask the Transport Company to book the service provisionally, and pass the offer (with the corresponding surcharge in the case of external Brokers) to the Personal Travel Assistant of the Travel Agency. Where the service has to be split (e.g. a plane, a train and a boat all need to be used), the Broker Agent will have the task of dividing it into separate services and of asking different Transport Companies for separate offers. If a complete service can be successfully put together with all the Transport Companies’ offers, the Broker Agent will book them temporarily, and the separate offers will then be combined, to provide a single offer to the Personal Travel Assistant.

The Personal Travel Assistant will then sort the list of all suitable trips and quotations received from all the Broker Agents, according to the Customer preferences, and provide that sorted list to the Customer for him or her to choose one option. The Customer may select one of the offered trips, reject them all and quit, or refine his or her requirements and start the process again. Whenever the Customer selects one of the trips, he/she will provide credit card details to the Personal Travel Assistant, who will process payment through the corresponding Financial Company. When the payment is completed correctly, the Personal Travel Assistant will notify the corresponding Broker Agent to confirm the booking(s), and this Agent will then pass on this confirmation to the appropriate Transport Companies. If the Personal Travel Assistant cannot process the payment (not enough credit, invalid or expired card, etc.) the Customer will be asked to either re-enter his payment details, or quit.

In any case, the Personal Travel Assistant will notify those Broker Agents whose offers have not been selected so they can cancel their bookings- information about this cancellation will then be passed on to the appropriate Transport Companies.

Once a month, the travel agency will pay each external Broker Agent for the services it has provided during the previous month. The payment will be done by by means of a money transfer to the bank account number of the Broker Agent.

The way Broker Agents pay their Transport Companies depends on their particular arrangements, and is not part of this specification.

Finally, as regards the description of the system, its accessibility requirements are as follows: (1) Customers may access the system using three different kinds of devices: browsers, PDAs, and mobile phones. The particular characteristics of each constrain the possible interfaces and functionality that need to be offered to each one. Potentially, the application will have to interact with new kinds of devices, although what their specific characteristics may turn out to be are unknown as yet. (2) Frequent customers will be able to store and modify their preferences, and will be eligible for discounts, depending on agency policy.

The rest of the article details this process step by step, applying it to the travel agency. The following section presents the OO-H method Web engineering approach used by the WebSA process in gathering the functional aspects.

4 A Web Functional Design Method: OO-H

The OO-H (Object-Oriented Hypermedia) method [11] is a generic model, based on the object-oriented paradigm, which provides the designer with the semantics and notation necessary for the development of Web-based interfaces. OO-H defines a set of models, techniques and tools that shape a sound approach to the modeling of Web interfaces. The OO-H proposal includes: (1) a design process, (2) a pattern catalog, (3) a domain model, (3) a navigation model, and (4) a presentation model.

The extension to "traditional software" production environments is achieved by means of complementary views: (1) the domain model which represents the domain entities of the Web application, (2) the navigation model that gathers the concepts related to the abstract structure of the site, and (3) the presentation diagram which represents specific presentation details. In this paper, we have only focused on the domain model and the navigation model, because they can provide important information as a contribution to the architectural point of view.

Figure 3 depicts the domain model for the travel agency running example. As you can see, this model represents the most important domain entities, free from any technical or implementation details, so in other words it represents an ideal class model. The *customer* class contains a set of different attributes that represent personal data such as (credit card, name and address*)* and it also provides a description of the trip required (*TripReq*), including personal constraints (*TripConstrains*) and preferences (*CusDetails*). The *TripRep* class contains a set of attributes to represent the places of departure and destination, as well as the dates of departure and return. The *TripConstraints* class includes the limits on the total price of the trip along with the duration of the trip itself, and any undesired means of transport. Finally, the *CustDetails* class has two attributes such as any undesired means of transport (e.g. if the customer does not like planes) and an order criteria of the services offered.

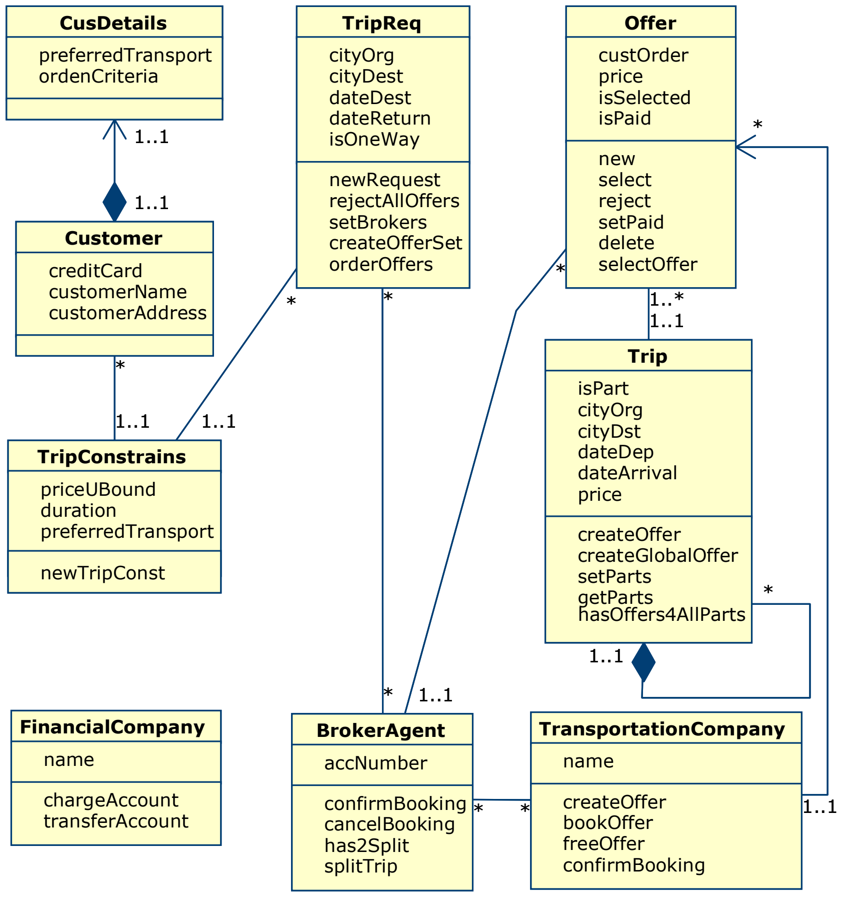


Figure 3. Domain Model of the Travel Agency

Once trip requirements have been selected, the system receives the request from the *Customer*, checks that it is well-formed, and selects the Broker Agents (*BrokerAgent*) that work with the agency and that can service the trip. The system interacts with each instance of *BrokerAgent*, asking them for an offer (*Offer*) that satisfies the demands of the Customer's requested trip. Each *BrokerAgent* may work with several Transport Companies (*TransportCompany*), asking them to provide an offer for the service requested. If the offer matches the customer requirements (select method of the *Offer* class), the *BrokerAgent* will ask the *TransportCompany* to book the service provisionally (*confirmBooking* operation of the *BrokerAgent* class). If the service has to be split (e.g. a plane, a train and a boat need to be used), the *BrokerAgent* will be in charge of dividing it into separate services and will ask different *TransportCompanies* for separate offers.

Once the designer has specified the domain model, a navigation model must be designed to define navigation and visualization constraints. The navigation model is based on a MOF navigation metamodel (see figure 4) which defines a set of different types of navigational constructs such as: (1) NavigationalClass, (2) NavigationalTarget, (3) NavigationalAssociation and (4) Collection.

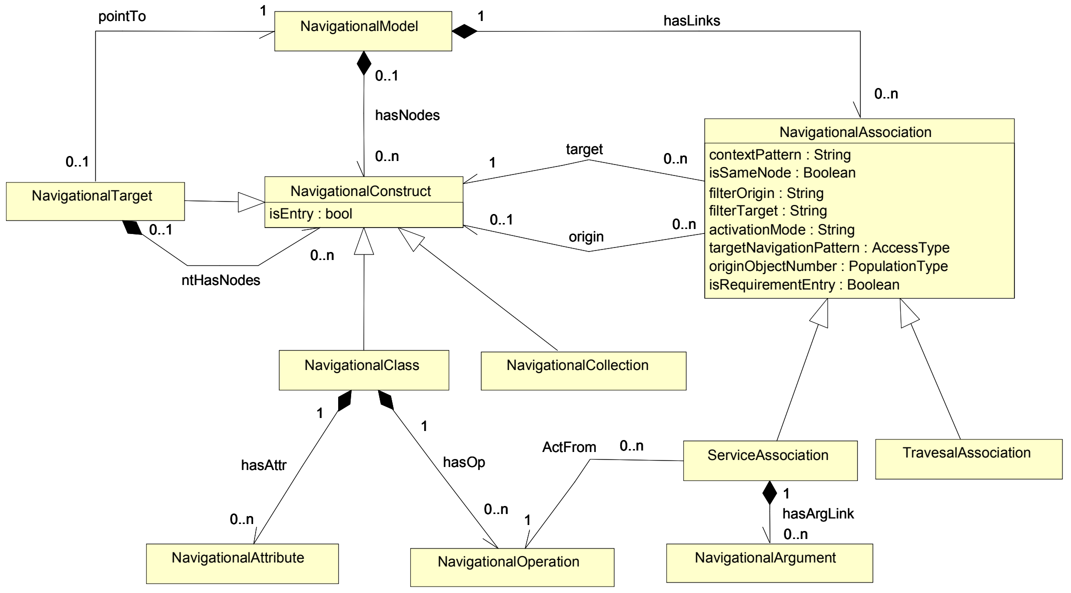


Figure 4. Simplified OOH Navigation metamodel

In addition to all this, when defining the navigation structure, the designer must take into account some orthogonal aspects such as the desired navigation behavior, the object population selection, and the order in which objects should be navigated, or the cardinality of the access. These features are captured by means of different kinds of navigation patterns and filters associated with the metaclass NavigationalAssociation.

Figure 5 depicts an OO-H navigation model for the travel agency case study. The navigation starts with a home page which has a link to create a new instance of customer trip constraints. Once an instance of TripConstraints (navigational class *TC*) has been set, the trip description, including places of departure and destination as well as dates of departure and return, must be keyed in by the customer. The execution of the *SLNR* ServiceAssociation produces a set of offers (navigational class *Offer1 as a result*). Each offer has a reference of the broker that provides the offer (navigational class *BA*), the name of the Transport company that manages the trip (navigational class *TCo*), and finally a combination of one or several trips that fulfill the customer's travel requirements from origin to destination (navigational class *Trip1*). The customer can accept an offer by means of the *SLSO* ServiceAssociation (selectOffer). In that case, the customer must provide his credit card data to formalize the booking. This is modeled with the *SLCA* ServiceAssociation (chargeAccount) from the navigational class *FinancialCompany*. If the transfer is accepted, the application will come back to the home page.

A default presentation reflecting the page structure of the interface can be derived from the navigation model. The OO-H CASE tool (VisualWADE) gives tool support to this process. This default presentation gives a functional but rather simple interface (with default location and styles for each information item), which will probably need further refinements in order to become useful for its inclusion in the final application. It can, however, serve as a prototype on which to validate that the user requirements have been correctly captured. We have modeled the travel agency running example with VisualWADE [29].

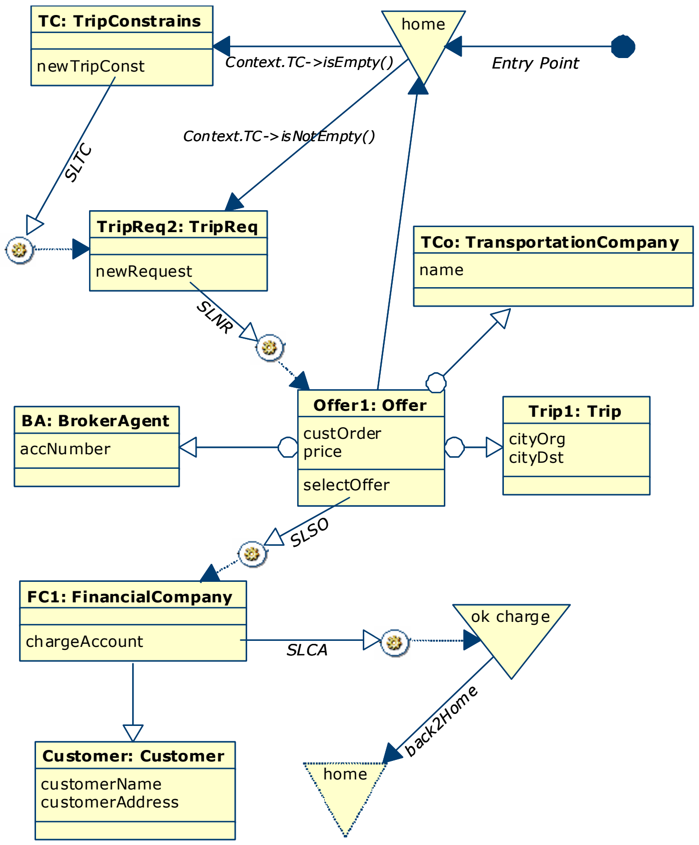


Figure 5. Navigation Model of the Travel Agency

At this point functional models (domain and navigation) have been specified. The next step in the analysis phase of WebSA is to specify the Web architectural models. For the purposes of this paper, only the configuration model needs to be specified.

5 Web Architectural Viewpoint: Configuration Model

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. Each element 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. This in turn allows us to specify the architectural configuration without knowing anything about the problem domain. At this level, we can also define architectural patterns for the Web application as a reuse mechanism.

A Configuration model is built by means of a UML 2.0 Profile of the new composite structure model. This model is well-suited to the task of specifying the software architecture of applications. The main modeling elements of the CM are *WebComponent, WebConnector, WebPart* and *WebPattern*.

To formalize the Configuration model elements and their relationships, we define the Configuration metamodel (see figure 6).

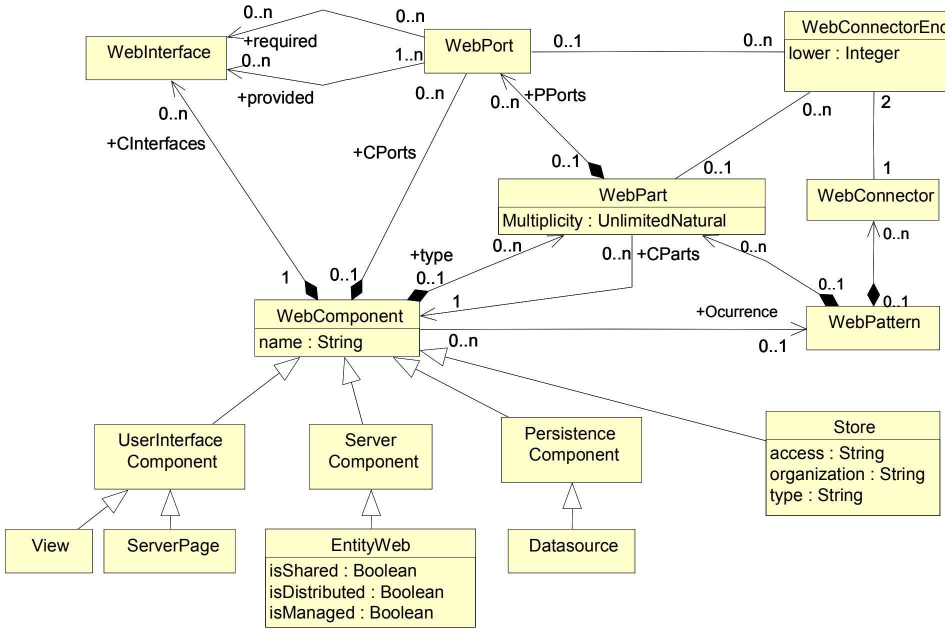


Figure 6. Simplified CM metamodel

5.1 *WebComponent*

A *WebComponent* in the Configuration Model 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 this kind of 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 figure 6). In addition to the subtypes of *WebComponent*, which are shown in figure 6, the Travel Agency example will also use the following subtypes: *ServerPage*, *ProcessComponent*, *UserAgent*, *DAC*, *LegacyView*, *Controller*, *View, Store* and *EntityData*. The complete topology of the WebSA components can be seen in [20].

5.2 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*. So a *WebComponent* can only communicate with the outside through its *WebPorts*.

5.3 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 figure 6). 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 interfaces provided in a *WebPort* characterize requests the environment makes to the *WebComponent*.

5.4 WebConnector

WebConnector specifies a link that allows the communication in the system between two or more *WebComponents* or/and *WebParts* of the *WebComponents* (see 5.6). 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 *WebConnectorEnd* (see figure 6).

5.5 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*.

5.6 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).

5.7 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 Travel Agency application has one *WebPattern* called *Façade* (see Figure 10) which contains some possible configuration of elements that represent the pattern Façade of [9].

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. Some authors [15], [27], consider the Composite Structure model as one of the major improvements incorporated into UML 2.0, because it permits us to specify software architectures following a proper component-based notation which incorporates ports, connectors and parts.

As [1] have observed, there are several ways of using UML profiles. One of these is to support the classification of classes as a means of emulating metamodel extensions. Thus, WebSA has defined a CM metamodel (see Figure 6) whose purpose is to represent the specific components of its Web specific architectural style. A CM profile was proposed later, in an effort to adapt the CM metamodel classes to the UML metaclasses. This provides with us two benefits: (1) the CM model can be specified in any UML tool and (2) the learning curve of the CM model is reduced.

Apart from all outlined above, the CM profile will also provide the necessary information for the T1 transformation defined in the WebSA development process (see figure 1) for integrating the functionality with the architecture in the IM model.

A UML 2.0 Profile mechanism is defined as a UML package stereotyped «profile» that can extend either a metamodel or another profile. UML profiles are defined in terms of three basic mechanisms: stereotypes, constraints, and tag definitions.

Figure 7 shows how 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 (see figure 6) to the semantics inherited from the UML metaclasses (see figure 7).

UML 2.0 has incorporated a relationship named *extension*, to specify how a stereotype extends a metaclass in a profile. It is depicted by an arrow with filled arrowhead (see figure 7). An extension is a binary relation, i.e., a stereotype is dependent on only one element of the underlying metamodel. An extension is marked as *{required}*, when the stereotype is always created if an instance of the extended class is created. In the CM profile, for example, an instance of *WebComponent* stereotype must be created when an instance of metaclass *Class* of UML is created (see figure 7).

We will only describe in detail the central modeling element of the profile, i.e. the stereotyped class *WebComponent* which extends the UML 2.0 metaclass "CompositeStructure:: StructuredClasses::Class". This metaclass Class defined by the Composite Structure Model extends the Class of Kernel so as to incorporate the capability to have an internal structure and ports. The WebComponent extends this metaclass Class, because by doing this it obtains two benefits: (1) The hierarchical decomposition of the components into subcomponents (called parts) the which increases the detail and the expressiveness of the component model and (2) the definition of architectural patterns using the Collaboration element which allows us to define and apply the architectural pattern in any CM.

Interested readers can download the complete profile at [20]. The description includes the constraints on the Composite Structure Model.

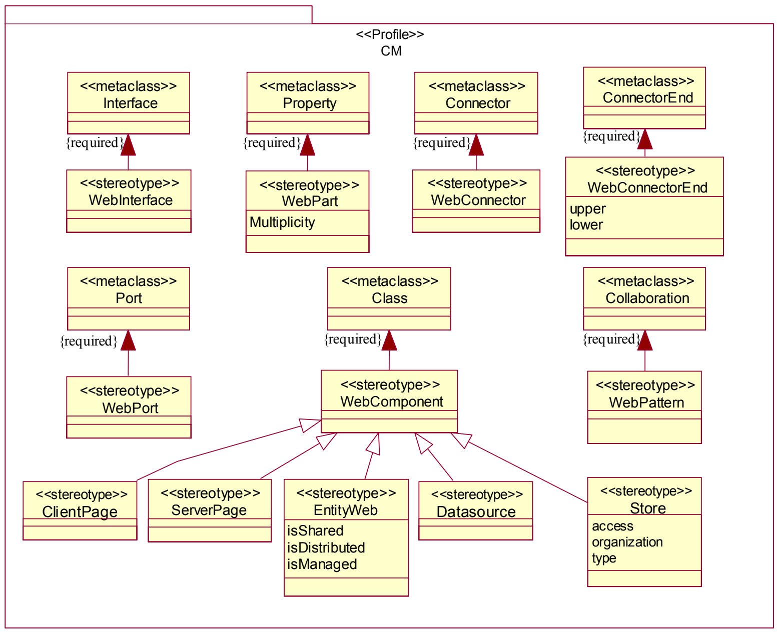


Figure 7. Configuration Model Profile

Name: WebComponent
Description This stereotype extends a UML metaclass "*CompositeStructures::StructuredClasses::Class*" (see figure 7)
Constraints **context** *CompositeStructures::StructuredClasses::Class* **inv**: self.isStereotyped ("WebComponent") implies -- *A WebComponent has not got features (attributes and operations).* self.attributes->isEmpty() and self.ownedBehavior->isEmpty() --*The Interfaces of a WebComponent must be instances of WebInterface* and self.implementation->forAll(i| i.contract.oclIsTypeOf (WebInterface)) --*All connectors are WebConnectors* and self.ownedConnector->forAll (c| c.oclIsTypeOf(WebConnector)) -- *All its ports must be instances of WebPort* and self.ownedPort->forAll (p | p.oclIsTypeOf(AWebPort))

where *isStereotyped* is an OCL operation defined as follows:
isStereotyped (stereotypeName:String) : Boolean;
self.extension-> exists (x | x.ownedEnd.type = stereotypeName)

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 8.

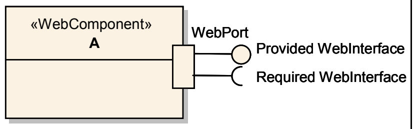
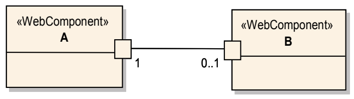
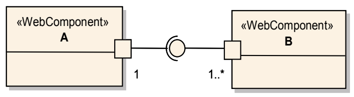
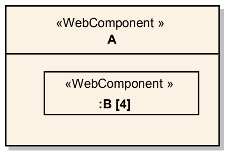
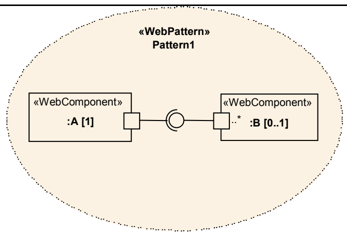
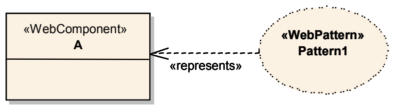
 «WebComponent» A WebPort Provided WebInterface Required WebInterface	**WebComponent** keeps the notation of UML structure class. It has incorporated the WebPort, by means of a small square on the boundary. **WebPorts** are associated to required or provided **WebInterfaces** with the lollipop notation. All subtypes of WebComponent such as ClientPage, ServerPage and Entity-Web (see figure 6) are represented using this same 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 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 section 5.6, 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 labeled with the keyword «represents» means that a **WebPattern** is used in a **WebComponent.**

Table 8**.** Notation used in a Configuration Model

In this article, therefore, we give an overview of the Travel Agency configuration model. figure 9 shows a general view of the CM representing the Travel Agency architecture, which is made up of the set of components and connectors that are described next.

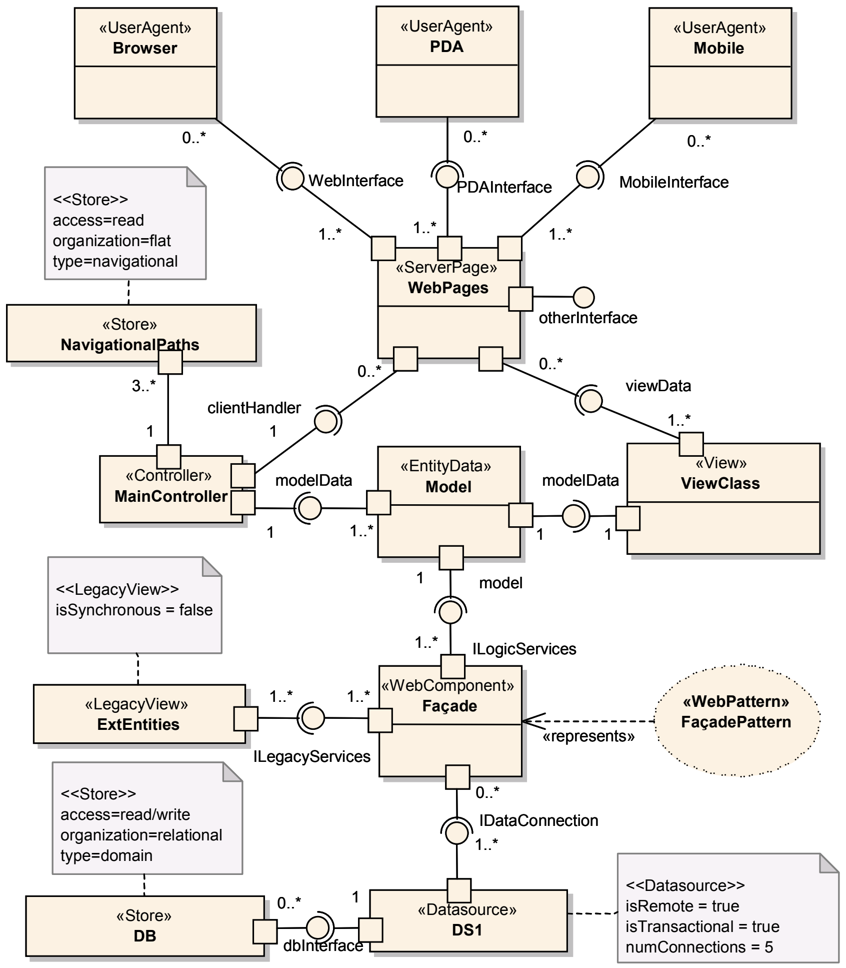


Figure 9. Configuration Model of Travel Agency

To deduce architectural aspects needed for the travel agency, we have based our work on the accessibility requirements and non functional requirements. We have thereby established five architectural assumptions:

- There must be a separation between the user interface that has to adapt to the different devices (e.g. cell phone, PDA, Web, etc.) and the presentation logic which is common to all users.
- Due to the fact that navigation requirements are different for each device, the MVC 2 pattern is applied. It allows us to locate the navigation from the different devices in an independent way (e.g. in an external file or store).
- The application presents different offers from the agencies continually, and the user interface is modified every day.

- The travel agency is an Internet application and has a large amount of clients. This application has to provide very good performance by means of middleware with distributed components, applying the Façade pattern.
- In order to obtain data from different companies about the trips offered, the Web application will need to connect to legacy systems.

Having obtained the architectural assumptions of the travel agency, we established its Configuration model (see figure 9). In the front-end part of the model we can find three different UserAgent components; we refer to the component or device that allows the user to interact with the system. In the travel agency there are three UserAgent: browsers, PDAs and mobiles. In order to decouple the different graphical interfaces with the same presentation logic, we have applied the Model-View-Controller *2* pattern. First, the *view* is provided by the ServerPage which receives the user's requests and renders the response in their device. Each ServerPage component provides a separate interface for attending to each UserAgent. It also contains the functionality information and is responsible for sending messages to the *Controller* component. The instances of a ServerPage are obtained from the navigational classes of the navigation models of OO-H [11] or UWE [16].

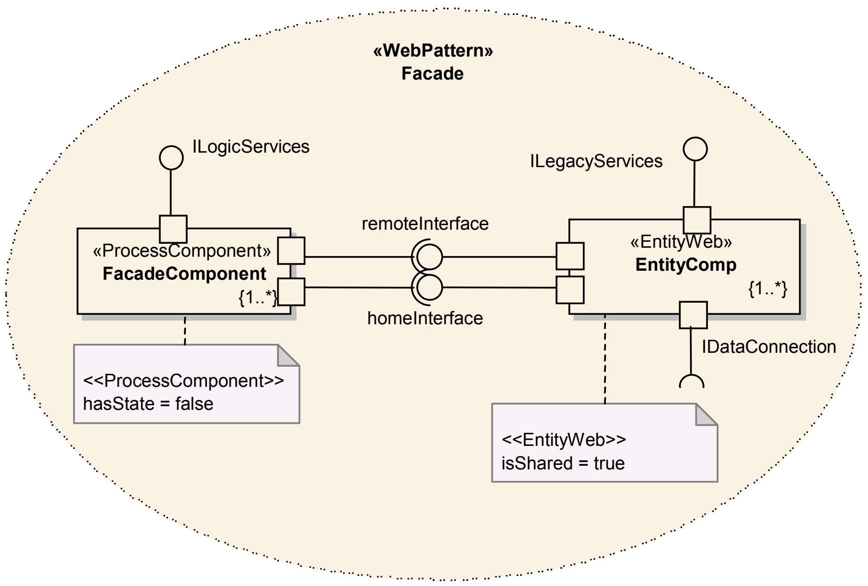


Figure 10. WebPattern Façade of Travel Agency

The *Controller* receives the requests through the WebPort *ClientHandler*. To establish navigation, it is connected to a Store component (*Navigational Path)* containing information about the links between pages. It separates the navigational aspects from the presentation aspects.

Each instance *ServerPage* needs an interface to access the required data objects. Such interface is provided by the WebPort *ViewData* of the *View* component. We can observe that the *model* component needs information from the components that implement the business logic. This is obtained through the *IProcessComponent* interface offered by the *Façade* WebPattern. Finally, the specified remote and transactional Datasource allows the connection to a Store component that contains the information modeled in the conceptual model of the functional view, which also has a read/write access, as well as a relational organization.

In addition, the Façade WebPattern has a group of one or more EntityWeb components called *EntityComp* that represents the elements of the domain in the business logic layer and each component contains the state of one domain entity (e.g., it could be implemented by an EJB Entity). In the Façade WebPattern, the set of EntityWeb components have the tagged value isShare=true indicating that they can be shared by multiple transactions and users. The *EntityComp* is also related to the component LegacyView, which offers a series of services that come from the *ILegacyServices* port to other applications and converts the received asynchronous calls into requests, sending them to the business logic. Finally, the EntityComp is connected with the persistence through the DataConnection Web Interface, to store the information of the application.

6 The WebSA Transformation Process

The WebSA transformation policy is driven by the architectural viewpoint, i.e. it is defined by a set of transformations in which first class citizens are the classes of the architectural view. The WebSA development process consists of two types of transformations: T1 and T2. T1 merges the elements of the architectural models of WebSA with those of the functional models, and translates them into a platform independent design model called Integration Model. T2 maps the platform specific implementation models (e.g. J2EE or .NET) from the Integration Model. Both transformations are complex, i.e. they are made up of a set of smaller transformation rules, which are executed in a deterministic way to bring about the completion of the transformation.

In MDA [22] there are different alternatives available for getting the information necessary to transform one model into another (e.g. using a profile, using metamodels, patterns and markings, etc). For WebSA we have selected a metamodel mapping approach to specify the transformations, because it allows us to obtain the information of the different Web approaches with just their MOF metamodel. Furthermore, the T1 transformation uses the model-merge approach defined by [22] which allows us to obtain information from several functional and architectural models and convert them into one design model. In this article we limit ourselves to explain the merging process of WebSA with the OO-H models (T1 in figure 1). For the purpose of obtaining this integration, we extend the MDA model transformation pattern of Bézivin [3]. The extension of this pattern integrates the OO-H and WebSA models by means of the metamodel based transformations. These metamodels based on the MOF language are the source of the transformation models that carry out the transformation to the target metamodel elements. The transformation models are defined in the QVT language, which is an MDA standard also based on the MOF language.

Recently, OMG has launched a new Draft Adopted Specification for QVT on MOF 2.0 [24]. This new version of QVT has been developed by the different groups of people who presented the previous proposals of QVT. The QVT specification has a hybrid declarative/imperative nature. The declarative part is split into a user-friendly part, based on transformations and composed of a rich graphical and textual notation. It also contains a core part which provides a more verbose and formal definition of the transformations. The declarative notation is used to define the transformations that indicate the relationships between the source and target models, but without specifying how a transformation is actually executed. In this way, QVT also defines operational mappings that extend the metamodel of the declarative approach with additional concepts. This allows us to define the transformations which use a complete imperative approach.

The QVT metamodel is defined using EMOF from MOF 2.0 and extends the MOF 2.0 and OCL 2.0 specifications. It allows for the expression of higher order transformations and fits in the central concept of MDA, namely, that transformations are themselves models. QVT transformations can be composed and extended by inheritance or overriding, which is necessary for scalability and reusability.

In the following lines, we present an example of a T1 transformation using the graphical notation of QVT. Due to the complexity of the T1 transformation, it is helpful to build a map of transformation rules (also called relations in QVT) that indicates the flow of execution and avoids redundancies in the specification. In the transformation map each transformation rule is related to the rest by means of three different types of relationships: (1) Composition – A transformation rule can be composed of one or more transformation rules (2) Dependency – One transformation rule must be executed before another transformation rule (3) Inheritance – A transformation rule extends or overrides another transformation rule.

We have therefore chosen to define a simple UML profile to represent the transformation map as a UML class model (see figure 11). The first transformation shown in the T1 map is from Subsystem Model to Integration Model.

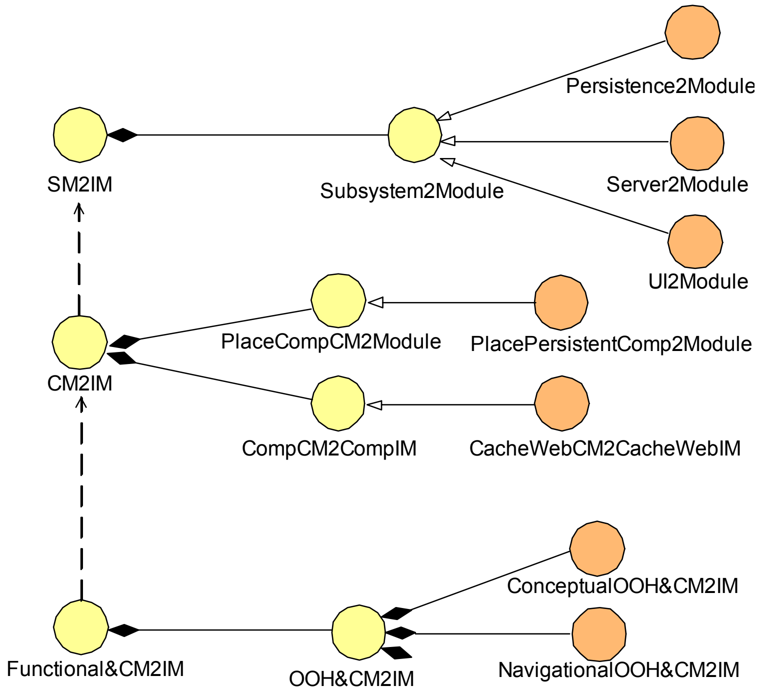
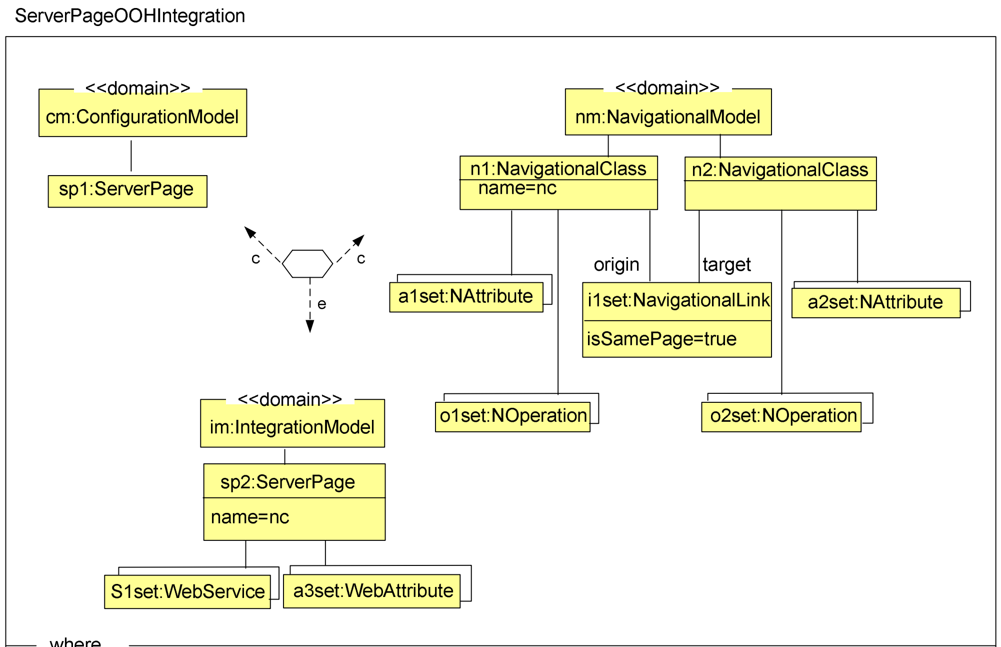


Figure 11. T1 Transformation Map

The second transformation rule (*CM2IM*) maps from Configuration Model to Integration Model. It is composed of a set of two types of rules.

The first one places components into the modules (*PlaceComp2Modules*), and the second one transforms each configuration component into one or more integration components (*CompCM2CompIM*). The last transformation rule *Functional&CM2IM* merges the functional OO-H models (conceptual and navigation) with the Configuration Model and introduces the functional aspects into the components of the Integration Model.

Figure 12 shows an example using the QVT graphical notation for the *ServerPage-OO-H2Integration* relation which involves three domains: Navigation, Configuration and Integration models. First, the relation checks if there is a set of instances in the Navigation model and another set of instances in the Configuration model (the arrow with the 'c' indicates that only this domain is being checked). It is at this particular moment that a set of instances in the Integration Model will be created, modified or deleted (the arrow with the 'e' indicates enforced, that is, the values of this domain will be modified so as to satisfy the rule).



SPOperation2WebService (o1set, s1set) ; SPOperation2WebService (o2set, s1set) ; SPNAttribute2WebAttribute (a1set,a3set) ; SPNAttribute2WebAttribute (a2set,a3set)

Figure 12. Example of T1: ServerPage and Navigational Class to Integration Model

To be specific, this relation checks whether there is at least one instance of ServerPage in the Configuration model (see figure 7), as well as two NavigationalClasses with a set of NAttributes and NOperations which are related through a NavigationalLink with its isSamePage attribute with true value in the Navigational model (see figure 5). Only if all these conditions are satisfied will the transformation rule create one ServerPage in the Integration model that merges the NOperations and NAttributes from the two NavigationalClasses into WebServices that represent a behavioural property of the Integration WebComponent and WebAttributes which in turn represent a structural property of the Integration WebComponent. Additionally, the "where" clause contains a set of relations that extends the previous relation. SPOperation2WebService generates for all NOperation of each NavigationClass element a WebService in a ServerPage. SPNAttribute2WebAttribute generates for all NAttribute of each NavigationClass element a WebAttribute in a ServerPage.

Figure 13 uses the QVT graphical notation to define the FacadeDomain2Integration relation. This transformation checks ('c' arrow) whether there is a class in the Domain model that contains a set of operations (see NOperation o1set in figure 13). It also checks in the Configuration model whether there is a WebPattern called Façade that contains both a ProcessComponent and an EntityWeb instances. If both patterns are found, the transformation enforces ('e' arrow) that both one stateless Process Component (that is, with its WebAttribute hasState=false) and an EntityWeb are created in the Integration model. In addition, the o1set from the class c1 is transformed into a set of WebServices associated to the Process component (s1set) and a set of WebServices associated to the EntityWeb component (s2set). As well as this, the NAttributes from the Domain model (a1set) are converted into a set of WebAttributes in the Integration model (a2set).

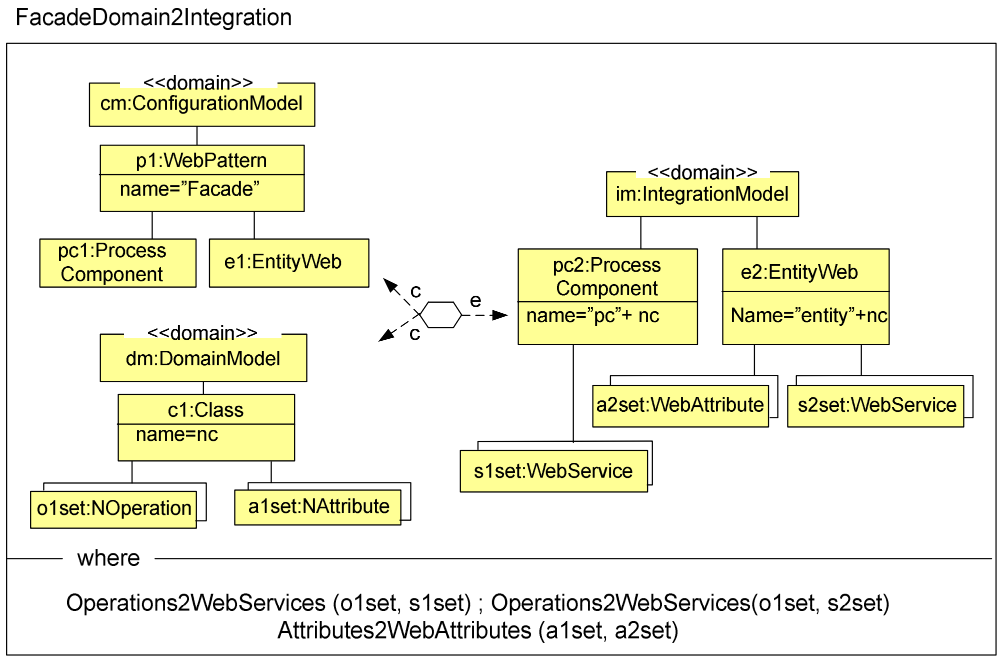


Figure 13. Example of T1: Façade and Domain Class to Integration Model

The links among the n-ary elements (depicted by two superimposed rectangles) in the T1 transformation rule are defined in the Operations2WebServices and Attributes2WebAttributes transformation rules included in its Where clause. On one hand, Operations2WebServices generates, for all NOperations of each Class element, a WebService in a Component. Operations2WebServices is a composed transformation where the name, type and parameters of each Operation are queried, the aim being to create a new WebService in the Integration component. On the other hand, Attributes2WebAttributes generates, for all NAttributes of each Class element, a WebAttribute with the same name and type in an EntityWeb.

7 Integration Model

IM defines a complete structural design of our application in a platform independent way. It integrates SM and CM with the functional viewpoint made for a specific problem. IM is also built by means of a UML 2.0 Profile of the composite structure model. From the T1 transformation, IM inherits some elements of CM such as the hierarchy of WebComponent, WebConnector and WebPort. But in contrast to CM, IM has introduced additional elements with the aim of representing a complete design of the Web Application. The most important element is WebModule, which allows us to represent a Subsystem from the SM. Moreover, gathered from the functional models, IM represents the functional properties of the WebComponent as WebServices and WebAttributes.

So it is clear that this model plays a preponderant role in WebSA, due to the fact that certain application characteristics are only identifiable when we consider functional and non-functional aspects together. For instance, in order to determine the granularity of the business logic components, it is necessary to know both architectural structure (e.g. whether this logic is likely to be distributed) and the business logic functionality itself (the tasks to be performed).

The IM does not need to be built up from scratch. The model is obtained by means of a PIM-to-PIM transformation applied on the SM and the CM together with the functional view (see T1 in figure 1). This mapping is based on a set of transformation rules defined in QVT that may vary, depending on the abstract component and/or the abstract dependency types. This automated mapping reduces the modeling effort. Another aspect of note is that this automated mapping causes the IM to inherit the architecture and design patterns defined in the CM, which will now be reflected in the concrete application.

Important, too, is the need to stress that this model still centers on design aspects (WebComponents, their WebPorts and WebParts, WebInterfaces, WebModules and WebConnectors), and does not show any detail about a specific platform. It should be said, furthermore, that the model is still independent from the target platform. Thus we will establish from this model a set of transformations to the different specific platforms such as J2EE, .NET, PHP, etc (see T2 figure 1). This makes it possible to classify it as a PIM (Platform Independent Model) in the context of MDA.

At this point we will focus on the Integration Model of the Travel Agency system. This model is made up of four WebModules (Presentation, UserControl, BusinessLogic and Persistence) which correspond to the four-tier distribution specified by the Subsystem Model. Each WebModule contains a set of WebComponents and WebConnectors located into a specific subsystem (e.g. the WebModule UserControl contains the WebComponents ServerPage and Controller which must be located into the UserControl subsystem).

Firstly, we present the WebModule *Presentation* which represents a thin user interface subsystem separated from the User Control functionality. This thin client has a poor graphical aspect, but it has a lower distribution cost, more security, and a better level of reuse. A typical implementation of Presentation layer is the HTML interface. The WebComponents like UserAgent, StaticPage, FunctionalPage can be contained within the WebModule Presentation. The Configuration model of the travel agency (see figure 7) only includes three UserAgents as possible candidates for this WebModule.

Figure 14 shows how the WebModule presentation contains the three types of UserAgent (Browser, PDA and Mobile) which are defined in the Configuration Model (see figure 9). Each UserAgent is connected with the only interaction point of the WebModule which is a WebPort with three required WebInterfaces: the *WebInterface, the PDAInterface and the MobileInterface.* These interfaces are offered by the next WebModule *UserControl* which will receive the different requests from the three UserAgents. At this point, it is worthy highlighting that the three UserAgents are pre-existing components like Web Browsers, PDA browsers, etc. For this reason, we do not need to generate these components. However, in order to communicate the UserAgents with the UserControl WebModule, the three interfaces must be obtained in the final implementation.

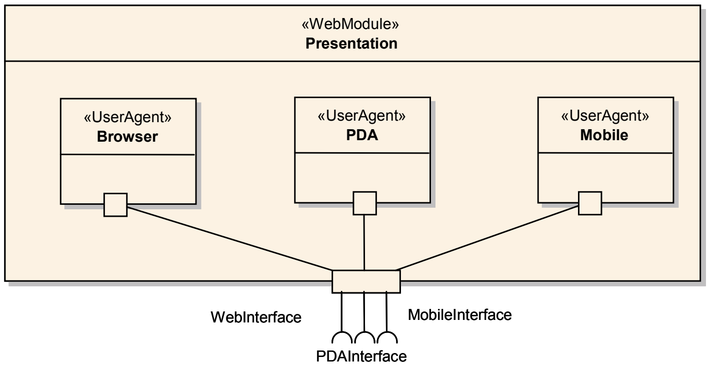


Figure 14. Presentation WebModule of the Travel Agency Integration Model

Following the top-down description of the travel agency Integration model, the next model is UserControl WebModule (see figure 15). This WebModule manages the request from the Presentation, establishes the navigation and redirects the logic services requests to the WebModule *BusinessLogic*.

Figure 15 shows a simplified WebModule UserControl of the Travel Agency IM. This module contains a set of WebComponents and their WebConnectors obtained by the T1 transformation. On the top, the module has three interfaces which are provided by the ServerPages that correspond to the three UserAgents. Each ServerPage WebComponent is obtained from one or more navigational classes (e.g. Offer, FunctionalCompany, Menu, etc.). A ServerPage has one required interface to access the View component (e.g. the serverPage *TripReq* has a required *IViewTripReq* to access the view *TripReqV*) and another required interface called *IClientHandler* to access the Controller called *MainController*. The *MainController* receives the requests through the *IClientHandler* and invokes the interfaces defined by the *model* components. Each of these *model* components is derived from one class of the domain model (e.g. *TripReqModel, TripModel, OfferModel*, etc.). We can also observe that each different *model* component sends information to the Business Logic through the *ILogicServices* interface.

Finally, we will show the BusinessLogic and Persistence WebModule of the travel agency IM. On the one hand, the *BusinessLogic* WebModule's task is to resolve the business rules established by the problem domain. In this case, this Business logic is composed of a set of different distributed components. On the other hand, the *Persistence* WebModule provides a storage system for the application Web data.

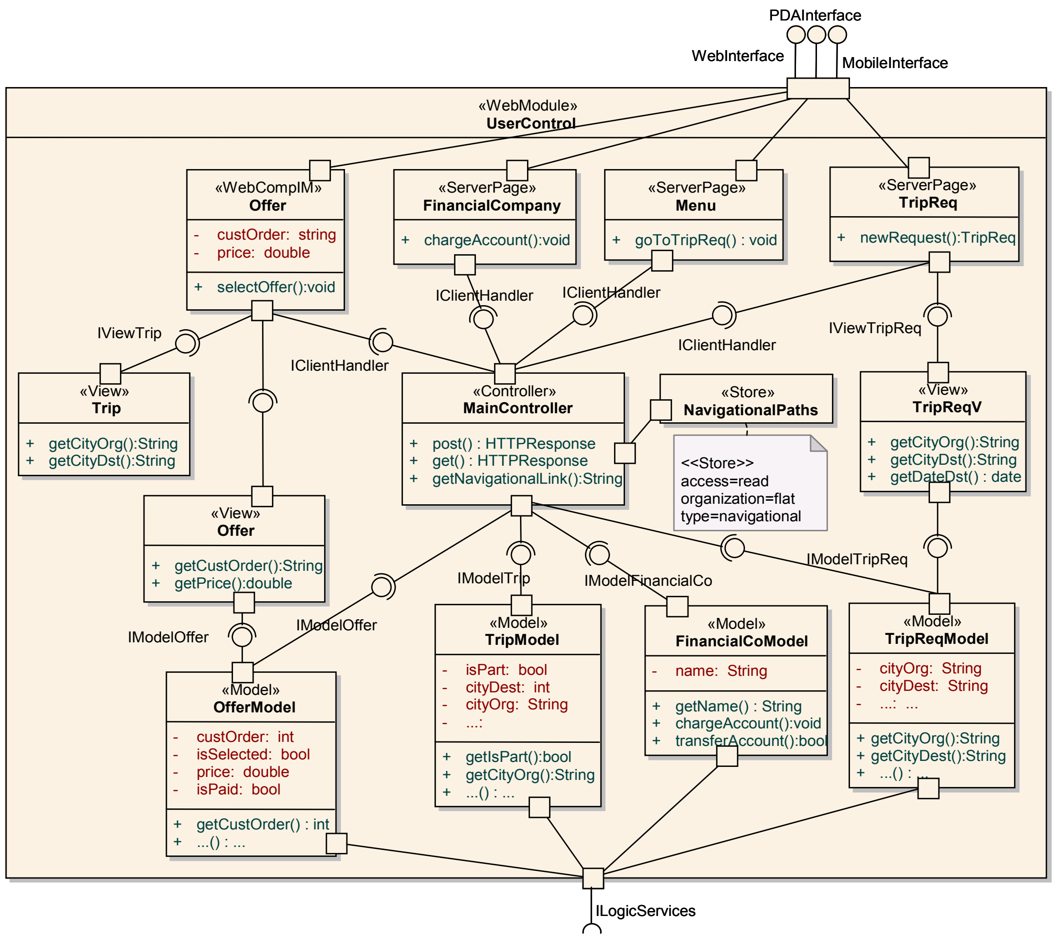


Figure 15. UserControl Module of the Travel Agency Integration Model

Figure 16 shows a portion of the travel agency IM that represents the simplified WebModule *BusinessLogic* and the WebModule *Persistence*. These WebModules contain a set of WebComponents and WebConnectors obtained by the T1 transformation. On the top, the WebModule called *BusinessLogic* has the *ILogicServices* WebInterface that gathers the requests from the UserControl components. This interface grants access to the different ProcessComponents in charge of obtaining all the requests from the client and launching the transactions into the business logic. As stated in the *FaçadeDomain2Integration* transformation (see figure 13), each ProcessComponent is obtained from one or more domain classes (Offer, TripReq, BrokerAgent, etc., see figure 3). When a ProcessComponent begins the transaction, it creates an EntityWeb by means of a Home Interface (e.g IHomeOffer), and subsequently invokes such an EntityWeb to access their WebServices through the Remote interface (e.g. IRemoteOffer). In our example, the EntityWeb stores the state of the class instances of the Domain model by means of a ServiceWeb *store* that invokes the Persistence WebModule. In the same way, the EntityWeb recovers the state of the class instances of the Domain model by means of the ServiceWeb *load*.

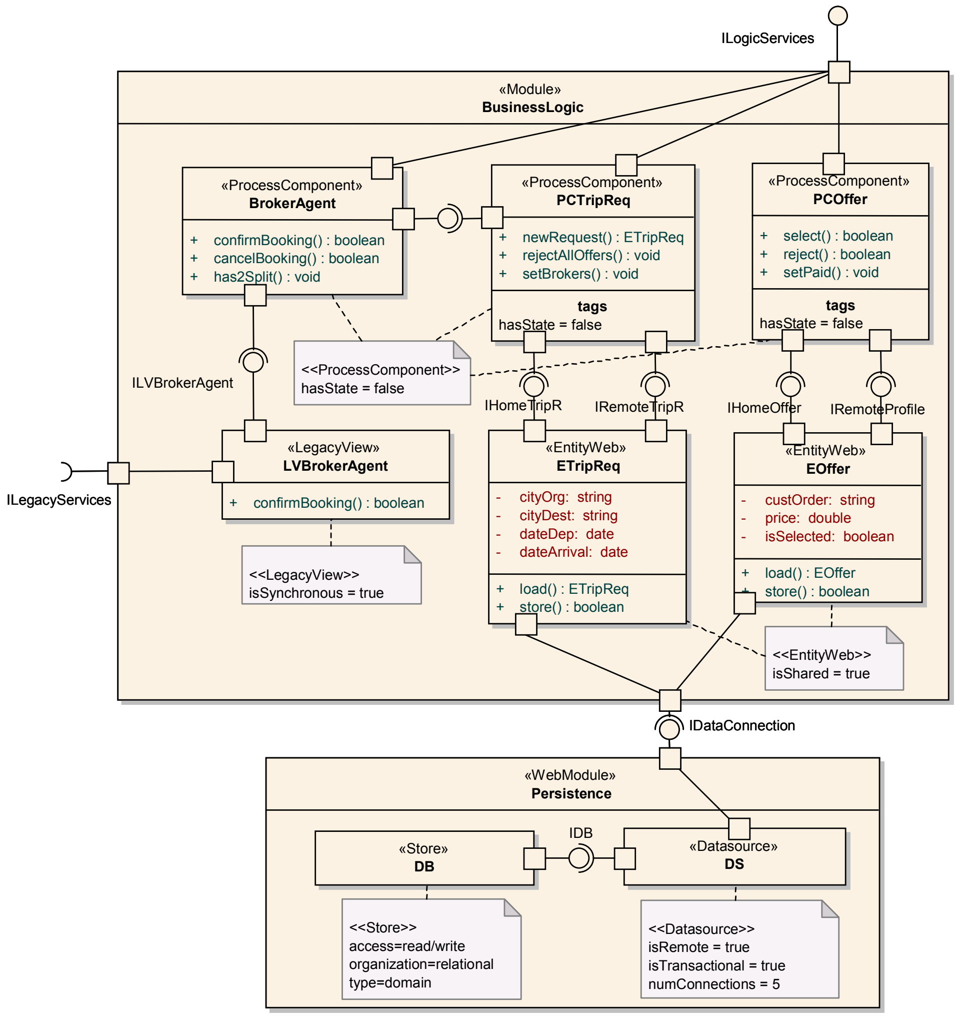


Figure 16. Business Logic and Persistence Modules of the Travel Agency Integration Model

The persistence WebModule is composed of two WebComponents. The DataSource provides one or more physical connections to store data. In this example, the DataSource provides a pool of five remote and transactional connections. It is connected to a relational database represented by a store *DB* WebComponent which contains the data information from the domain model.

In the next section, we will present the T2 transformation which allows us to obtain the specific platform models from the IM.

8 T2: Transformation from PIM to a PSM

Once the transformation T1 is completely executed, the functionality becomes interwoven into the architectural aspects in the Integration Model. Now we can tackle the final step of the WebSA development process, defining a set of PIM-to-PSM transformations for each target platform, such as J2EE, .NET from the Integration Model. As is specified in [22], in order to make a transformation from PIM-to-PSM, design decisions must be made. These decisions are specified in the transformation T2 and taken in the context of a specific implementation design. Hence T2 is made up of a set of simple transformations in which one Integration Model component is transformed into a platform specific component with the specific properties of this platform. To specify the T2 transformation, it is necessary to have the metamodels of the target platforms (e.g. the J2EE metamodel can be obtained from [25]).

```
relation ServerPage2J2EE {

    checkonly domain  IntegrationModel  sp:ServerPage {
                     name=nc,
                     services = Set((WService) {name=on,
    type=ot}),
                     views = Set ((View) {name = vn})
     }
    enforce domain J2EEModel jsp:JavaServerPage {
                      name=nc,
                      forms = Set((Form) {name=on, type=ot}),
                      beans = Set((JavaClass) {name = vn}
    }
     where {
         services->forAll  (s1| WebService2Form (s1, forms))
         views-> forall (v | View2Bean (v, beans))
    }
}
```

Figure 17. Example of T2: A ServerPage to a JSP of J2EE

Figure 17 shows a QVT example of the transformation rule of T2 for J2EE using the textual notation. It transforms each *ServerPage* component of the Integration Model specified in the first *domain* into a *JavaServerPage* specified in the second *domain*. The elements of the Integration Model domain are for the purpose of accomplishing the relation, but the J2EEModel domain has to create, modify or delete its elements to satisfy it. In this example, the *ServerPage* has a set of *WebServices*; each one of them is translatable into a java method, a javascript method or an HTML form. In this example, we have chosen a translation into an HTML form by the *WebService2Form* transformation rule defined in the *forall* OCL sentence of the *{where}* part. In the same way, each of the View elements related to the *ServerPage* is translated into a *JavaBean* through the *View2Bean* transformation rule. The PSMs obtained from the WebSA process are considered as an implementation, because they provide all the information needed to construct an executable system. It should be underlined that the implementation obtained is not complete because the behavioral aspects have to be introduced manually. The behavioral models will be introduced soon, and this will provide WebSA with a complete code generation.

9 Related Work

This section compares our work with related research in the areas of Model Driven Engineering and Software Architecture for Web applications.

An example of approach based on MDA for Web applications is Tai et al [28]. They provide a set of models based on a metamodel that is used as a central contract between the developers. They also describe a tool implemented on the basis of the metamodel. The tool provides a variety of code generators and a mechanism for checking whether view artifacts, such as JavaServer Pages, are compliant with the model. Our approach improves this idea with (1) the integration of proven successful models from the Web engineering field and (2) the formalization of the code generation phase by means of transformation rules.

Another Model Driven methodology for Web Information System development is MIDAS [6]. This methodology applies a MDA metamodel to the Web platform using XML and object-relational technology. MIDAS proposes different PIMs and PSMs and defines some mapping rules between models. Unlike the WebSA approach, it establishes the transformation mapping following a low level solution in XML, instead of defining the transformations using standard models as QVT which are easier and more understandable for the stakeholders. Furthermore, MIDAS does not provide any Web application aspect that is architectural.

For their part, the Web architecture approaches focus on emphasizing scalability, independent deployment, interaction latency reduction, security enforcement and legacy systems encapsulation. The first approach we are going to review is the Representational State Transfer (REST) [8] which is an architectural style aimed at representing Web architectures and focused on the generic connector interface of resources and representations. However, REST has only served both as a model for design guidance and as a test for architectural extensions to the Web protocols, whereas WebSA has used some concepts of these architectural styles to define a process development for the production of Web applications.

Hassan and Holt [12] also present an approach aimed at recovering the architecture of Web applications. Their 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 extracted facts. Conversely, WebSA follows the opposite process that goes from the representation of the architecture to implementation.

Conallen's work is another well-known approach to extending UML [6] for the modeling of the architecture and design of Web applications. Conallen presents the Web Application Extension (WAE) for UML, which generates the 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 in the development of Web applications.

Finally, a more recent approach is WAM (WebComposition Architecture Model) [17] which is an extension of WebComposition that introduces an architectural description and serves as a map to keep track of the interrelations between the different federated Web applications. Among the modeled artifacts are Web services, Web Applications and organizational zones of control that are all subject to evolution in the sense of the WebComposition approach. In contrast with WebSA, WAM is not based on model driven development and does not formalize the mapping from the architectural model to the implementation.

10 Conclusions and Further Work

WebSA is an approach that, with techniques for the development of Web architectures, complements the currently existing methodologies for the design of Web applications. WebSA contains a set of UML architectural models and QVT 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 OO-H approach.

In this paper we focus on the development process of WebSA and describe how models are integrated and generated, based on model transformations. For the specification of the transformations we choose a promising QVT approach that allows for visual and textual description of the mapping rules.

Despite the fact that WebSA can be modeled using any standard UML 2.0 tool, we are working on a WebSA tool which would establish the set of Web models and represent QVT transformation models that support the WebSA refinement process. First, the Web models are represented in a UML 2.0 tool of the market which generates XMI documents. The WebSA tool reads the XMI documents and the QVT models are specified on this tool. Finally, the tool generates an XMI file when the target is a model and a source code file when the target is the implementation. This allows us to define the transformations while guaranteeing the traceability between those models and the final implementation.

Acknowledgments

This work has been partially funded by the METASIGN project (TIN2004-00779) from the Spanish Ministry of Education and Science, by the DADASMECA project (GV05/220) from the Valencia Government, and by the MESSENGER (PCC-03-003-1) and DADS (PBC-05-012-2) projects from the Regional Science and Technology Ministry of Castilla-La Mancha (Spain). This article is expended from the work presented at the 1st Workshop on Model-Driven Web Engineering (MDWE 2005), at the 5th International Conference on Web Engineering (ICWE 2005).

References

1. C. Atkinson, T. Kühne, B. Henderson-Sellers. Systematic stereotype usage, Software and System Modeling 2 (3), 153-163, 2003
2. L. Bass, M. Klein, F. Bachmann. Quality Attribute Design Primitives, CMU/SEI-2000-TN-017, Carnegie Mellon, Pittsburgh, December 2000
3. J. Bézivin. In Search of a Basic Principle for Model Driven Engineering, Novática nº1, June 2004
4. G. Booch. The Architecture of Web Applications, DeveloperWorks: IBM developer solutions, June 2001
5. 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
6. P. Cáceres, E. Marcos, B. Vela. A MDA-Based Approach for Web Information System, Workshop in Software Model Engineering (WisME), 2004
7. S. Ceri, P. Fraternali, M. Matera. Conceptual Modeling of Data-Intensive Web Applications, IEEE Internet Computing 6, No. 4, 20–30, July 2002

8. R. Fielding, R. Taylor. Principled Design of the Modern Web Architecture, ACM Transactions on Internet Technology, Vol. 2, No. 2 , 115-150, May 2002
9. E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995
10. J. Gómez, A. Bía, A. Párraga. Tool Support for Model-Driven Development of Web Applications. The 6th International Conference on Web Information Systems Engineering (WISE, 2005), 721-730, November 2005
11. J. Gómez, C. Cachero, O. Pastor. Conceptual Modeling of Device-Independent Web Applications. IEEE Multimedia, 8(2), 26–39, 2001
12. A. Hassan, R. Holt. Architecture Recovery of Web Applications, International Conference on Software Engineering (ICSE'02), May 2002
13. I. Jacobson, G. Booch, J. Rumbaugh. The Unified Software Development Process, Addison-Wesley, 1999
14. A. Kleppe, J. Warmer, W. Bast. MDA Explained. The Model Driven Architecture, Practice and Promise, Addison-Wesley, 2003
15. C. Krobyn. UML 3.0 and the Future of Modeling, Software and System Modeling, Vol. 3, No. 1, 4-8, 2004
16. N. Koch, A. Kraus. The Expressive Power of UML-based Web Engineering, In Proc. of the 2nd. Int. Workshop on Web-Oriented Software Technology, CYTED, Spain, 105-119, June 2002
17. J. Meinecke, M. Gaedke, M. Nussbaumer. A Web Engineering Approach to Model the Architecture of Inter- Organizational Applications. COEA'05, 125-137, 2005
18. S. Meliá, J. Gomez. Applying Transformations to Model Driven Development of Web applications. 1st International Workshop on Best Practices of UML (ER, 2005).LNCS 3770,63-73, Austria, October 2005
19. S. Meliá, A. Kraus, N. Koch. MDA Transformations applied to Web Application Development, In Proc. of 5th International Conference on Web Engineering (ICWE'05), LNCS 3579, 465-472, July 2005
20. S. Meliá. The WebSA Composition Model Profile. Technical Report TR-WebSA2, http://www.dlsi.ua.es/~santi/papers/WebSA%20CM%20profile.pdf, November 2004
21. OMG. Model Driven Architecture, OMG doc. ormsc/2001-07-01
22. OMG. MDA Guide, OMG doc. ab/2003-05-01
23. OMG. Meta Object Facility (MOF) v1.4, OMG doc. formal/02-04-03
24. OMG. MOF Query/Views/Transformations Draft Adopted specification: OMG doc. ptc/05-11-01
25. OMG. UML Profile for Enterprise Distributed Object Computing Specification. OMG doc. ad/2001-06-09
26. K. Renzel, Wolfgang Keller. Client/Server Architectures for Business Information Systems. A Pattern Language, PLoP Conference, 1997
27. B. Selic. An Overview of UML 2.0 (Tutorial), UML 2004
28. H. Tai, K. Mitsui, T. Nerome, M. Abe, K. Ono. Model-Driven Development of Large-scale Web Applications, IBM J. Res. & Dev. Vol. 48 No. 5/6, Sep/November 2004
29. VisualWADE Case Tool. http://www.visualwade.com, May 2005
30. Workshop on Model-driven Web Engineering (MDWE 2005). http://www.lcc.uma.es/~av /mdwe2005