

Applying WebSA to a case study: A travel agency system

Santiago Meliá and Jaime Gómez

Web Engineering Research Group.

Dept. of Languages and Information Systems. Universidad de Alicante

{santi,jgomez}@dlsi.ua.es

Abstract

Web engineering research community has proposed several web design methods that have proven successful for the specification of the functional and navigational requirements posed by Web information systems. However, the architectural features are often ignored in the design process. This situation causes Web applications with rigid and predefined architectures depending on the Web design method the designer is applying. To overcome this limitation, we propose a generic approach called WebSA. WebSA is based on the MDA (Model-driven Architecture) paradigm. It proposes a Model Driven Development made up of a set of UML architectural models and QVT transformations as a mechanism to integrate the functional aspects of the current methodologies with the architectural aspects. In this paper, we apply WebSA with the OO-H method using as a running example the Travel Agency specification.

1. Introduction

The Web Engineering community is well aware that, in order to keep track of the changes and assure the feasibility of applying their methods to commercial Web applications, it is necessary to evolve the different proposals that should now integrate the explicit consideration of architectural features in the Web application design process. In order to do so, several authors propose the use of well known techniques in the Software Architecture discipline [1] in order to identify and formalize which subsystems, components and connectors (software or hardware) should make up the Web application.

These architectural features are especially important in methodologies that provide a code generation environment WebML [5], OO-H [7], UWE [13], etc. The addition of an architectural view would cover the gap that nowadays exists between the Web design models and the

code architecture. Therefore, the inclusion of one such model would decrease the set of arbitrary decisions that are usually taken in order to generate the code in such environments, decisions that sometimes compromise the universal usefulness of the solution. Also, the addition of an architectural model would provide a mechanism to discuss, document and reuse (by means of pattern catalogs) the architectural decisions that answer the different non-functional user requirements.

For this purpose, we propose the WebSA (Web Software Architecture) [14] [15] approach based on the standard MDA (Model Driven Architecture) [17]. The MDA framework provides WebSA not only with the possibility to specify a set of Web-specific models, but also to specify each process step from the models to implementation by means of a set of transformation rules. In order to define these transformations, there are several initiatives related to the MDA approach, among others the Request for Proposals for a Query/Views/Transformations (QVT) [20] language. QVT is, in our opinion, the most interesting one as it is well defined language and it comprises a graphical as well as a textual notation.

In order to understand this approach, we explain each of steps of the WebSA development process through the running example a Travel Agency.

The paper is organized as follows: Sections 2 and 3 give an overview of the WebSA development process and the OO-H approach, respectively. Section 4 presents the most important model in analysis phase of WebSA, the Configuration model. Section 5 proposes the specification of the T1 transformations that specify the merging of the functional and the architectural models in the QVT language. Section 6 explains the Integration model. Section 7 shows how the T2 transformation is defined in order to obtain a J2EE implementation. In section 8, the relevant related work is compared to our approach and finally, in section 9, some future steps of the application of WebSA to the development of Web applications are outlined.

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 (see Sect. 2.1) to specify the architectural viewpoint which complements current Web engineering methodologies such as [7], [13]. Furthermore, WebSA also establishes an instance of the *MDA Development Process* [11], which allows for the integration of the different viewpoints of a Web application by means of transformations between models (see Sect. 2.2).

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 mainly based on the classical architectural style defined in [3] – 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 [22] 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. 4.
- **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 [19] repository metamodel and a set of OCL constraints (both part of the OMG proposed standards) that together specify (1) which is the semantics associated with each model element, (2) which are the valid configurations 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 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.

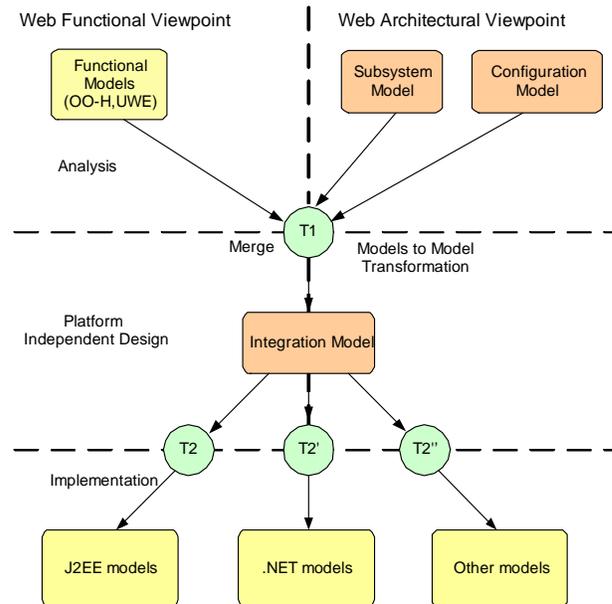


Fig. 1. The WebSA Development Process

In order to meet these requirements, the WebSA development process establishes a correspondence between the Web-related artifacts and the MDA artifacts. Also, and as a main contribution, WebSA defines a transformation policy driven by the architectural viewpoint, that is, an “architectural-centric” process [10] (see Fig. 1).

Fig. 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 [7] or UWE [13], 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. As it is defined in [3], “an architectural style is independent from its realization, and does not directly

refer to a concrete application problem it is intended to solve”. In this way, 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 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), 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 view in this process plays a pre-eminent role for the completion of the specification of the final Web application, and drives the refinement process from analysis to implementation.

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

3. A Web Functional Design Method: OO-H

The OO-H (Object-Oriented Hypermedia) method [7] is a generic model, based on the object-oriented paradigm that provides the designer with the semantics and notation necessary for the development of Web-based interfaces. OO-H defines a set of diagrams, 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 navigation diagram, and (4) a presentation diagram.

The extension to “traditional software” production environments is achieved by means of two complementary views: (1) the navigation diagram (ND) that defines a navigation view, and (2) the presentation diagram (PD) that gathers the concepts related to the abstract structure of the site and the specific presentation details, respectively.

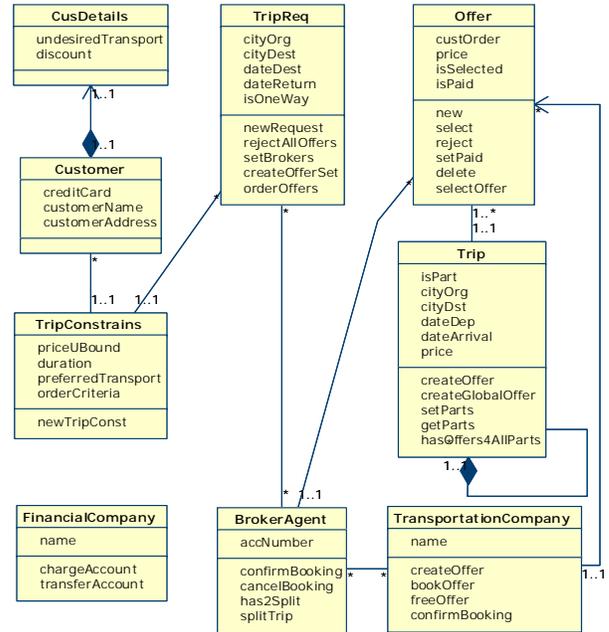


Fig. 2. Conceptual Model of the Travel Agency.

For the purposes of this paper, only the ND are relevant. The ND diagram enriches the domain view provided by a standard UML class diagram with navigation and interaction features. Fig. 2 depicts a potential class diagram for the travel agency running example. The customer provides a description of the required trip to the system, including personal constraints (Tripconstrains) and preferences (CusDetails). The trip description (TripReq) contains the cities of origin and destination, as well as the departure and return dates. For one-way trips, only the departure cities and dates are required. Constraints on the trip may include bounds on the total price of the trip, duration of the trip itself, and any undesired transportation method (e.g. the customer does not like planes). Preferences may include the preferred transportation mechanisms. Once trip requirements has 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 them and that can service the trip. The system interacts with each Broker Agent, asking them for an offer (Offer) that fulfils the Customer’s requested trip. Each Broker Agent may work with several Transportation Companies (TransportationCompany), asking them to provide an offer for the requested service. If the offer matches the customer requirements (select method of the Offer class), the Broker Agent will ask the Transportation Company to temporarily book the service (confirmBooking method of the BrokerAgent class). In case the service has to be split (e.g. a plane, a train and a boat need to be used), the Broker Agent will be the one in

charge of dividing it into separate services and ask different Transportation Companies for separate offers. If a complete service can be successfully put together with all the Transportation Companies' offers, the Broker Agent will temporarily book them, and the separate offers will be then combined to provide a single offer to the Personal Travel Assistant.

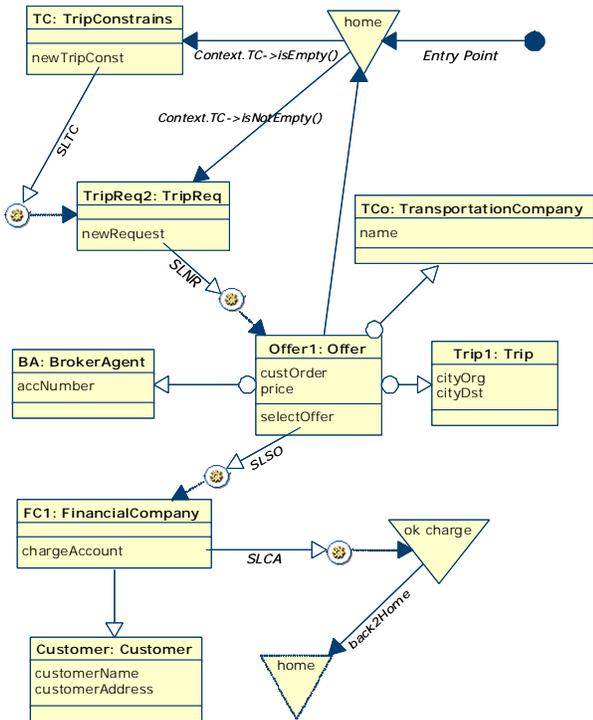


Fig. 3. Navigation Model of the Travel Agency.

Once the designer has specified a class diagram, to define navigation and visualization constraints, a ND must be designed. This diagram is based on four types of constructs: (1) navigation classes, (2) navigation targets, (3) navigation links and (4) abstract pages. Also, when defining the navigation structure, the designer must take into account some orthogonal aspects such as the desired navigation behaviour, 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 links and abstract pages [8].

Fig. 3 depicts a potential ND for the travel agency running example. The navigation starts with a home page that has a link to create a new instance of customer trip constraints. Once trip constraints has been set, the trip description including cities of origin and destination as well as departure and return dates must be introduced by

the customer. The execution of the *SLNR* service link produces as a result a set of offers. Each offer has a reference of the broker that provides the offer, the name of the transportation company that manages the trip, and finally a combination of one or several trips that fulfill the trip customer requirements from origin to destination. The customer can accept an offer by means of the *SLSO* service link (*selectOffer*). In that case, the customer must provide their credit card data to formalize the booking. This is modeled with the *SLCA* service link (*chargeAccount*).

A default PD reflecting the page structure of the interface can be derived from the ND. The OO-H CASE tool (VisualWADE) gives tool support to this process. This default PD 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 and the result can be viewed in [25].

At this point functional models (class, navigation and presentation models) has 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.

4. 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, 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 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, which is well-suited to specify the software architecture of applications. The main modeling elements of the CM are *WebComponent*, *WebConnector*, *WebPart* and *WebPattern*. Their notation and semantics will be specified in [16].

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 [12], [23] consider the Composite Structure model as one of the major improvements incorporated to UML 2.0, because it allows us to specify software architectures following a proper component-based notation that incorporates ports, and provided interfaces and required interfaces, connectors, parts, etc.

The CM profile will also provide the necessary information for the T1 transformation defined in the WebSA development process (see Fig. 1) for integrating the functionality with the architecture in the IM model.

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

Therefore in this article we give an overview of the Travel Agency configuration model. Fig. 4 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.

In order to deduce architectural aspects needed for the travel agency, we have based our work on the accessibility requirements and functional requirements proposed at the Workshop. In this way, we have established five architectural assumptions:

- There must be a separation between the user interface that has to adapt to the different devices (p.e cell phone, PDA, web, etc) and the presentation logic which is common to all users.
- Due to the navigation requirements are different for each device, the MVC 2 pattern is applied. It allows to locate the navigation from the different devices in a independent way (p.e in a external file or store).
- Continuously, the application has to present different offers from the agencies and the user interface has to modify. It drives to maintain the user interface every day.
- As the travel agency is an Internet application and has a large amount of clients. This application has to provide a very good performance by means of a middleware with distributed components applying the Façade pattern.
- In order to obtain data from different companies about the offered trips. The application will need to connect to legacy systems.

Once we have the architectural assumptions of the travel agency, we established its Configuration model (see Fig. 4).

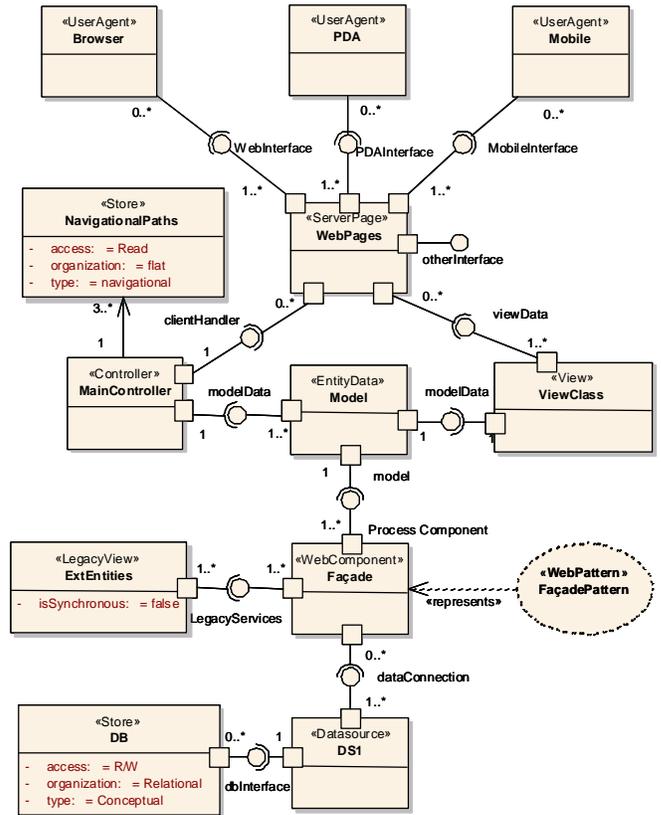


Fig. 4. Configuration Model of Travel Agency

In the front-end part of the model we can find three different components UserAgent, that is, the component or device that allows 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 in order to attend 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 [7] or UWE [13].

The Controller receives the requests through the WebPort ClientHandler. In order to establish the navigation, it is connected to 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.

The *Façade* WebPattern represents a group of one or more stateless *ProcessComponents* (e.g. a *Session Stateless*), which receives the requests through the *BLogic WebInterface* from the *MVC2*, and resends them to the *Entity*. This pattern requires the interface *dataConnection* to access the *Datasource* to store the information. The *WebComponent Façade* is in turn related to the component *LegacyView*, which offers a series of services that come from the *LegacyServices* port to other applications and converts the received asynchronous calls into requests and sends them to the business logic. Finally, the specified remote and transactional *Datasource* allows the connection to a *Store* component that contains the information modelled in the conceptual model of the functional view, which also has a read/write access, as well as a relational organization.

5. 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 transformations, which are executed in a deterministic way in order to complete the transformation.

In *MDA* [18] there are different alternatives to getting the information necessary for transforming 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 just with their *MOF* metamodel. In this article we limit ourselves to explain the merging process of *WebSA* with the *OO-H* models (*T1* in Fig. 1). In order to obtain this integration we extend the *MDA* model transformation pattern of *Bezibin* [2]. 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 *Request For Proposals (RFP)* for *QVT* on *MOF 2.0* [20]. 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 which comprises a rich graphical and textual notation, and 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 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.

Next, we present an example of a *T1* transformation using the graphical notation of *QVT* and also an example of a *T2* transformation in the textual notation of *QVT*.

5.1 Transformation T1: Merging Web Functionality with Architectural models

Due to the complexity of the *T1* transformation, it is helpful to build a map of transformations that indicates the flow of execution and avoids redundancies in the specification. In the transformation map each transformation is related to the rest by means of three different types of relationships: (1) *Composition* – A transformation can be composed by one or more transformations (2) *Dependency* – A transformation must be executed before another transformation (3) *Inheritance* – A transformation extends or overrides another transformation.

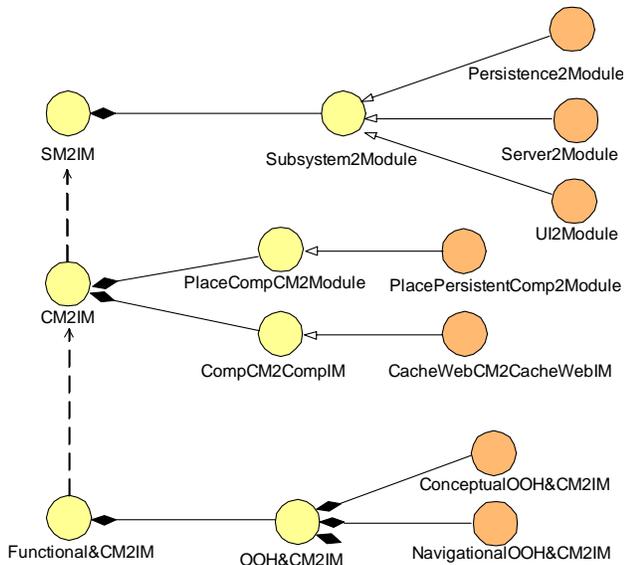


Fig. 5. T1 Transformation Map

Therefore, we have chosen to define a simple UML profile to represent the transformation map as a UML class model (see Fig. 5). The first transformation shown in the T1 map is from Subsystem Model to Integration Model.

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

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 *Functional&CM2IM* merges the functional OOH models (conceptual and navigation) with the Configuration Model and introduces the functional aspects into the components of the Integration Model.

Fig. 6 shows an example using the QVT graphical notation for the *ServerPage-OOH2Integration* transformation which involves three domains: Navigation, Configuration and Integration models. First, the transformation 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). At this moment, 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 in order to satisfy the rule).

Specifically, this transformation checks whether there is at least one instance of *ServerPage* in the Configuration model (see Fig. 4), 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 Fig. 3). Only if all these conditions are satisfied, will the transformation create one *ServerPage* in the Integration model that merges the *NOperations* and *NAttributes* from the two *NavigationalClasses* into its *WebServices* and *WebAttributes*, respectively. Additionally, the where clause contains a set of transformations that extends the previous transformation. *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*.

ServerPageOOHIntegration

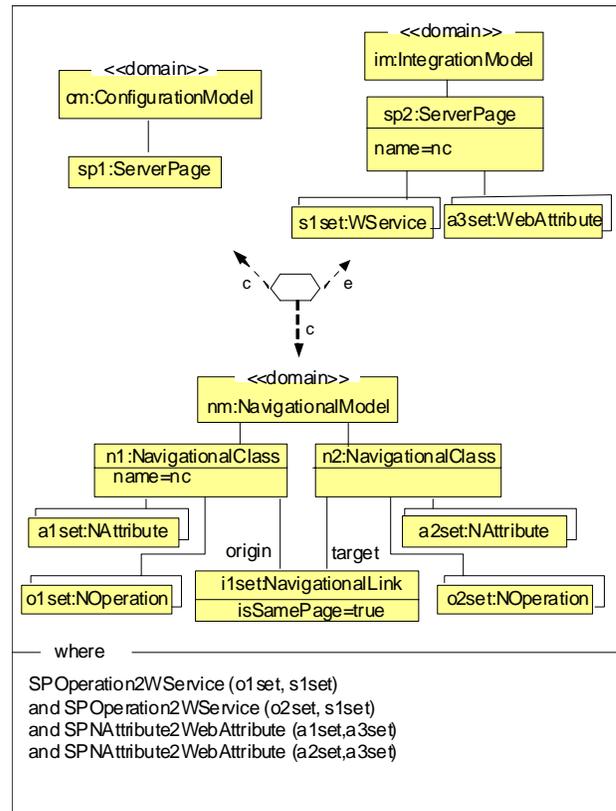


Fig. 6. Example of T1: NavigationalOOH&CMTtoIM

6. 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. Therefore, 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).

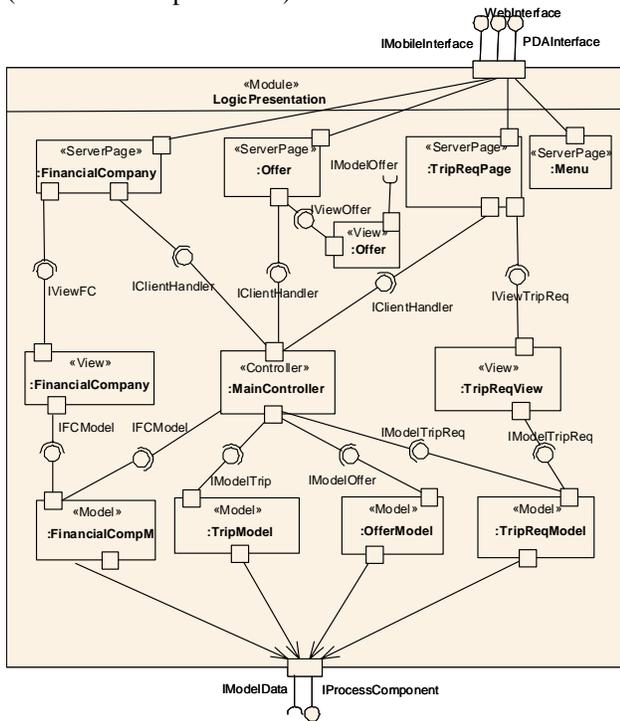


Fig. 7. Integration Model of Logic Presentation Module of Travel Agency

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 Fig. 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. Also, this automated mapping causes the IM to inherit the architecture and design patterns defined in the CM, which will be now reflected in the concrete application.

The resulting model is the basis on which the designer may perform further refinements in order to fine-tune the architecture to the system needs.

It is also important to stress that this model still centers on design aspects (WebComponents, their WebPorts and WebParts, WebInterfaces, WebModules and WebConnectors), and does not say anything about implementation. In this way, the model is still independent from the target platform. From this model, we define a transformation to the different specific platforms such as J2EE, .NET, PHP, etc (see T2 Fig. 9).

This makes possible to classify it as a PIM (Platform Independent Model) in the context of MDA.

Fig. 7 shows a portion of the Travel Agency IM that depicts the module *LogicPresentation*. This module contains a set of WebParts that represent the instances of the WebComponents and their relationships obtained by the T1 transformation. On the top, the module has three interfaces which are provided by the ServerPages that correspond to the three UserAgent. Each ServerPage instance is obtained from one or more navigational classes (e.g. Offer, Menu, etc.). A ServerPage needs an interface to access the required *View* component and another interface to access the *Controller*. The *Controller* 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.). Finally, we can observe that each different *model* component requires and provides information through the *IModelData* and *IProcessComponent* interfaces, respectively.

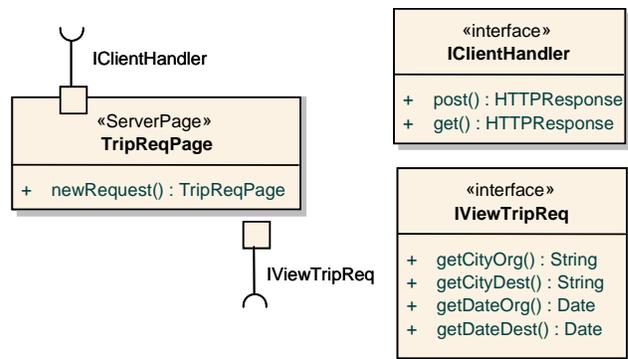


Fig. 8. Definition of the TripReq ServerPage and its interfaces.

The definition of the complete WebComponents and WebInterfaces can be made on the component definition, but it is usually more useful to define them elsewhere in the model using component and interface classes, as shown in Fig. 8. This makes it easier to maintain the model, because their definitions are usually used in more than one place. By having a single definition that is reused by referencing it where needed, it is easy to make changes without introducing errors. Fig. 8 depicts the TripReqPage component which is a ServerPage that contains one service (*newRequest*) and two interfaces (*IClientHandler* and *IviewTripReq*). Furthermore, we can see all the services offered by each interface.

7. Transformation 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 [18], 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. Therefore, 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 [21]).

```
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| WService2Form (s1, forms))
    views-> forall (v | View2Bean (v, beans))
  }
}
```

Fig. 9. Example of T2: ServerPage2J2EE

Fig. 9 shows a QVT example of transformation 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 only checked in order to accomplish the transformation, 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 defined in the *forall*

OCLE sentence of the *{where}* part. In the same way, each of View elements related to the *ServerPage* is translated into a *JavaBean* through the *View2Bean* transformation. The PSMs obtained from the WebSA process are considered an implementation, because they provide all the information needed to construct an executable system.

8. Related Work

This section compares our work with related research in the area of Web Engineering: On the one hand, MDA applied to the development of Web applications. On the other hand, we focus on the approaches that address the software architecture of Web applications.

An example of approach based on MDA for Web Applications is Tai et al [24]. They provide different kinds of artifacts in a consistent and cohesive way by means of a metamodel. In contrast, our approach formalizes the code generation by means of transformation rules, and uses the traditional views from the Web engineering methodologies.

Another model-driven methodology for Web Information System development is MIDAS [4]. This methodology uses XML and object-relational technologies for the specification of the PSMs. Unlike the WebSA approach, it does not establish the transformation mapping following the standard QVT, and it does not provide any architectural aspect of the Web application.

On the part of the Web architecture, the approaches are focused on emphasizing the scalability, independent deployment, and interaction latency reduction, security enforcement, and legacy systems encapsulation. For instance, approaches such as Representational State Transfer (REST) [6] 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 has used some concepts of this architectural style to define a process development for the production of Web applications.

Finally, Hassan and Holt [9] also 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 extracted facts. Conversely, WebSA follows the opposite process that goes from the representation of the architecture to the implementation. However, it does not describe the transformation rules used to realize this reverse engineering process.

9. 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 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 different Web design approaches.

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.

We are currently working on a tool to represent the set of QVT transformation models that support the WebSA refinement process. This work will allow to represent the transformations while guaranteeing the traceability between those models and the final implementation.

10. References

- [1] L. Bass, M. Klein, F. Bachmann. Quality Attribute Design Primitives, CMU/SEI-2000-TN-017, Carnegie Mellon, Pittsburgh, December 2000.
- [2] J. Bézivin. In Search of a Basic Principle for Model Driven Engineering, *Novática* n°1, June 2004, 21-24
- [3] 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
- [4] P. Cáceres, E. Marcos, B. Vela. A MDA-Based Approach for Web Information System, Workshop in Software Model Engineering, WisME 2004.
- [5] S. Ceri, P. Fraternali, M. Matera. Conceptual Modeling of Data-Intensive Web Applications, *IEEE Internet Computing* 6, No. 4, 20–30, July/August 2002
- [6] R. Fielding, R. Taylor. Principled Design of the Modern Web Architecture, *ACM Transactions on Internet Technology*, Vol. 2, No. 2, 115-150, May 2002
- [7] J. Gomez, C. Cachero, O. Pastor. Extending a Conceptual Modelling Approach to Web Application Design. In 12th CAiSE '00. International Conference on Advanced Information Systems, LNCS 1789, Springer, 79-93, 2000
- [8] J. Gómez, C. Cachero, O. Pastor. Conceptual Modeling of Device-Independent Web Applications. *IEEE Multimedia*, 8(2), 26–39, 2001
- [9] A. Hassan, R. Holt. Architecture Recovery of Web Applications, International Conference on Software Engineering (ICSE'02), May 2002
- [10] I. Jacobson, G. Booch, J. Rumbaugh. *The Unified Software Development Process*, Addison-Wesley, 1999
- [11] A. Kleppe, J. Warmer, W. Bast. *MDA Explained: The Model Driven Architecture, Practice and Promise*, Addison-Wesley, 2003
- [12] C. Krobyn. UML 3.0 and the Future of Modeling, *Software and System Modeling*, Vol. 3, No. 1, 4-8, 2004
- [13] 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, Málaga, Spain, 105-119, June 2002
- [14] S. Meliá, C. Cachero, J. Gomez. Using MDA in Web Software Architectures, 2nd OOPSLA Workshop of Generative Techniques in the Context of MDA, <http://www.softmetaware.com/oopsla2003/mda-workshop.html>, October 2003
- [15] S. Meliá, C. Cachero. An MDA Approach for the Development of Web Applications, In Proc. of 4th International Conference on Web Engineering (ICWE'04), LNCS 3140, 300-305, July 2004
- [16] S. Meliá. The WebSA Composition Model Profile. Technical Report TR-WebSA2, <http://www.dlsi.ua.es/~santi/pPublicaciones.htm>, November 2004.
- [17] OMG. Model Driven Architecture, OMG doc. ormsc/2001-07-01
- [18] OMG. MDA Guide, OMG doc. ab/2003-05-01
- [19] OMG. Meta Object Facility (MOF) v1.4, OMG doc. formal/02-04-03
- [20] OMG. 2nd Revised submission: MOF 2.0 Query / Views / Transformations RFP, OMG doc. ad/05-03-02
- [21] OMG. UML Profile for Enterprise Distributed Object Computing Specification. OMG doc. ad/2001-06-09
- [22] K. Renzel, Wolfgang Keller. Client/Server Architectures for Business Information Systems: A Pattern Language, PLoP Conference, 1997
- [23] B. Selic: An Overview of UML 2.0 (Tutorial), UML 2004.
- [24] 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
- [25] VisualWADE Case Tool. <http://www.visualwade.com>, May 2005