Workshop on Model-driven Web Engineering (MDWE 2005)

Nora Koch, Antonio Vallecillo, Gustavo Rossi

Sydney (Australia), July 26, 2005
http://www.lcc.uma.es/~av/mdwe2005/

Proceedings

In conjunction with:
ICWE 2005. 5th International Conference on Web Engineering
http://www.icwe2005.org

Editors

Nora Koch
Institute for Informatics,
Ludwig-Maximilians-Universität München and FAST GmbH,
Oettingenstr. 67, 80538 München, Germany
kochn@pst.informatik.uni-muenchen.de
http://www.pst.informatik.uni-muenchen.de/~kochn/

Antonio Vallecillo
University of Málaga
ETSI Informática
Campus Teatinos
29071 Málaga (Spain)
av@lcc.uma.es
http://www.lcc.uma.es/~av

Gustavo Rossi
Lifia. Facultad de Informática. UNLP
50 y 115, La Plata, Buenos Aires, Argentina
gustavo@sol.info.unlp.edu.ar
http://www-lifia.info.unlp.edu.ar

Table of Contents

List of Authors

Program Committee

Preface

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

The development of Web applications is a specific domain in which MDE can be successfully applied, due to its particular characteristics. In fact, existing model-based Web engineering approaches currently provide excellent methodologies and tools for the design and development of Web applications. They address different concerns using separate models (navigation, presentation, data, etc.), and count with model compilers that produce most of the application's web pages based on these models. However, these proposals also present some limitations, especially when it comes to modeling further concerns, such as architectural styles, or distribution. Furthermore, current Web applications need to interoperate with other external systems, which require their integration with third party web-services, portals, portlets, and also with legacy systems.

Recently, the MDA initiative has provided a new approach for organizing the design of an application into (yet another set of) separate models so portability, interoperability and reusability can be obtained through architectural separation of concerns. MDA covers a wide spectrum of topics and issues (MOF-based meta-models, UML profiles, model transformations, modeling languages, tools, etc.) that need to be yet solved.

The Model-Driven Web Engineering (MDWE) Workshop provided a discussion forum where researchers and practitioners on these topics met, disseminated and exchanged ideas and problems, identified some of the key issues related to the model-driven development of Web applications, and explored together possible solutions and future works.

All papers submitted to MDWE 2005 were formally peer reviewed by at least by two referees. This volume contains the 10 papers finally selected for presentation at the workshop. They deal with different issues of Model-Driven Web Engineering, both theoretical and practical, focused in general aspects and in specific sub-domains such as Web Services, Portlets and Business Processes. We feature papers devoted to specific methodologies or methodological issues and to formal approaches to Model-Driven development.

We would like to thank to the ICWE 2005 organization for giving us the opportunity to organize the workshop. Many thanks to all the authors for their submissions, and particularly to the contributing authors. Our gratitude also goes to the paper reviewers and the members of the Program Committee for their help in the selection of the papers.
.

Sydney, Australia, July 2005

Nora Koch, Antonio Vallecillo, Gustavo Rossi
MDWE2005 Organizers

A Model Driven Approach for the Integration of External Functionality in Web Applications. The Travel Agency System

Victoria Torres, Vicente Pelechano, Marta Ruiz, Pedro Valderas
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camí de Vera s/n, Valencia-46022, España
{vtorres, pele, mruiz, pvalderas }@dsic.upv.es

Abstract

Nowadays, it is getting more and more common to develop Web applications where part of the functionality is carried out by different systems. These systems provide functionality developed in different technologies that are integrated to build a complete web application. To deal with the integration issue that allows us to build this kind of Web applications, current Web Engineering methods should provide the mechanisms that facilitate this integration with third business parties during the modeling process. This paper presents a model driven method to achieve integration with external parties at a high level of abstraction. The method provided is an extension to the OOWS approach for the construction of this new kind of Web applications. The Travel Agency System has been taken as a case study to clearly understand how the whole method is applied.

1. Introduction

Web applications cannot be longer conceived as isolated applications. Moreover, the different possibilities in which business partners can provide their functionality (CORBA, J2EE or .NET) motivate us to propose a method that helps in the construction of more opened and collaborative Web Applications that integrate functionality from different sources.

There are several ways in which web applications can be built integrating functionality provided by external parties. For instance, a web application could require a concrete external functionality to accomplish a specific functional requirement or to provide some information that complements the data handled by our system. A more complex way could be when business process supported by the web application makes use of activities implemented by external business partners.

Web engineering methods are extending their solutions to provide support and/or integrate functionality and business processes into web conceptual models. In this context, we can distinguish approaches that deal with business process modelling and integration into navigational models like OOHDM [1], WSDM [2] or UWE [3] (that introduce process definitions into navigational models, causing a semantic overload of the navigational nodes because activities and processes are living together with nodes and links), other methods like WebML [5] and UML-Guide [6] model business processes as some kind of navigation; UML-Guide is based on the semantic web technology (OWL) to specify state machines that are used to express navigation and web service operation calls. Both approaches introduce some kind of syntactic mechanisms to include web service calls into the navigational model. Finally, OO-H [4] and WIED [7] (in the form of a companion notation to the WebML), model business processes and navigational models as separate concerns and notations. Only WebML and UML-Guide are worried about how to support integration of external functionality.

Our proposal introduces some contributions in this context because we think that the integration of business process in web application modelling should follow a concern oriented approach preserving the role and the notation of current business process modelling techniques (for instance, UML activity diagrams) and navigational modelling techniques as OO-H and WIED states, but also focusing on solving the integration to external parties problem in a model driven fashion (following the MDA principles). We also want to emphasize that we provide a methodological guide that helps web designers in the construction process of this new kind of web applications. Moreover, we think that Web Engineering methods also should face up the integration problem from two different points of view, which are the consumer and the provider perspective.

On one hand, as consumers we need mechanisms that help us to model Web Applications that use external artifacts. On the other hand, as producers we need mechanisms for generating artifacts that could be exported and used by other applications.

In this context it is necessary to provide a methodological guide that helps web developers in the construction process of this new kind of web applications. We think that the integration issue should be tackled following a model driven approach.

The rest of the paper is structured as follows. Section 2 provides an overview of the method, explaining the set of models use it and the existing dependencies between them. Moreover, we state how this proposal fits into the MDA approach. Section 3 presents briefly the Travel Agency System (TAS) case study. In the following sections, from 4 to 6 we present the method, by means of the TAS case study. Section 7 shows the strategy followed to build Web Services that make accessible the Broker Agent implemented in the TAS. In section 8 we outline how we generate the interaction with external functionality provided in different technologies. In section 9 we show the user interfaces generated from the specifications made in the Navigational Model. Finally, section 10 draws some conclusions and outlines further work.

2. An Overview of the Model-Driven Method

This method provides an extension to the OOWS [12] approach. This extension introduces the required expressivity to capture the integration requirements that are necessary to build distributed web applications.

Following the MDA guidelines, this method has been organized in three views: the Computation Independent Model (CIM), the Platform Independent Model (PIM) and the Platform Specific Model (PSM). Our proposal, as can be seen in figure 1, introduces new models for: requirements elicitation (Task Definitions) and supporting integration (the Services and the Business Process Models). To support integration when modeling navigation we extend the already defined Navigational Model and reuse those existing models like the Dynamic, Structural, Functional and Presentation.

Due to the fact that our proposal is based on an existing method that provides a code generation tool (OlivaNova CASE tool1), in this extended version, we want to provide a solution for (1) producing functionality for third party consumption and (2) integrating functionality supplied by external providers. In particular, we focus on the integration of external services at the Business Process and Navigational level.

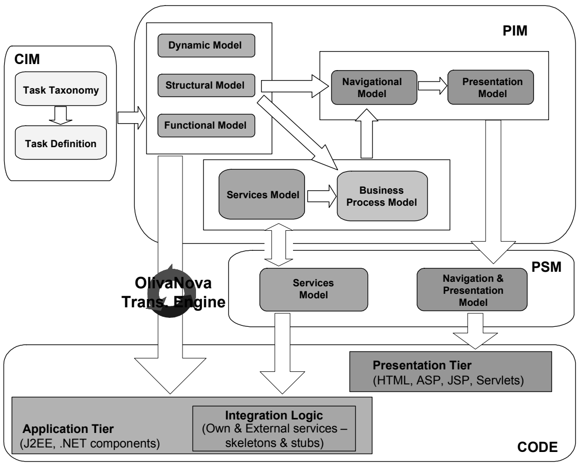


Fig. 1 Method Overview

In Fig. 1 we can see how the models proposed are organized in each different level (CIM, PIM, PSM and code).

3. Applying the Method to the TAS Case Study

The TAS is a Web Application that sells electronically travels to its customers. In particular, we only concentrate on providing transportation services (plains, trains, cars, boats or combinations of those) for a trip. This service can be either provided by external Broker Agents (implemented by other Travel Agencies) that work in conjunction with our TAS or implemented in our system. These Broker Agents use the services provided by Transportation Companies to supply an offer that matches with the customer requirements. In case a trip cannot be supplied by any Transportation Company, it is the Broker Agent which has to split the trip and try to compose the complete trip from split services. Once the customer has selected an offer that matches with his/her requirements, the TAS uses the services provided by the corresponding Financial Company to proceed with the payment of the selected offer. Finally, once a month, the TAS pays external Broker Agents for the services they have provided during the previous month.

In the following sections we are going to apply the TAS case study to the method roughly presented in the previous section. Moreover, we will also include how

[1] http://www.care-t.com

we build the Broker Agent business logic and the strategy followed to construct the Web Services that provide the functionality implemented by our Broker Agent.

4. Defining the CIM

The Computation Independent Model (CIM) proposed by MDA is built mainly to bridge the existing gap between those that are experts about the domain and those that are experts on how to build the artifacts that satisfy the requirements domain [12]. Then, according to MDA, a CIM must describe the requirements of the system.

We specify the early requirements of a Web application by means of a task model. This model is built from the tasks that users must be able to achieve when interacting with the application as well as from the tasks that the system must perform. The requirements specified when building the task model are used in following stages for systematically generating part of the PIM. This approach allows us to provide a higher degree of traceability than those requirement specification methods which transform models manually.

We propose two steps to define the task model:

(1) *Task identification*: we identify the set of tasks that the system together an actor must achieve to accomplish each requirement. An actor represents a user or any other system that interacts with the system under development [10]. The set of identified tasks are organized in a task taxonomy.

(2) *Task description*. To accomplish the goal defined by each leaf task included in the task taxonomy, we describe the set of actions that must be performed to succeed in achieving that goal. This description is made by using the UML activity diagrams [10].

4.1 Task Identification

To identify the set of tasks that represent the web application requirements we must detect, as a first step, which actors can interact with the system. Then, for each detected actor we must define a task taxonomy that represents the tasks that this actor can achieve when interacting with the system.

In the TAS example, we only detect an actor: the internet user. The task taxonomy associated to this actor is shown in Fig. 2. For the construction of the task taxonomy, we take as the starting point, a statement of purpose that describes the main goal of the web application. The statement of purpose is considered the most general task of the system. From this task, a progressive refinement is performed, obtaining as a result more specific tasks. Tasks are decomposed into subtasks by following structural or temporal refinements. The *Structural* refinement (represented by solid lines in Fig. 2) decomposes complex tasks into simpler subtasks. The *Temporal* refinement (represented by dashed lines in Fig. 2) provides order constraints for the children tasks according to the task logic. To define these temporal constraints we propose the use of the temporal relationships introduced by the ConcurTaskTree approach (CTT) [11]. In Fig. 2 we can see the temporal relationship *Enabling* (>>) which represents that after being finished the first task the second is activated. The rest of temporal relationships proposed by the CTT approach are not explained in this work due to space constraints.

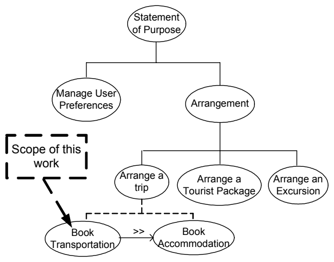


Fig. 2 A Task Taxonomy of the TAS

The statement of purpose of this system is decomposed into two tasks: *Manage User Preferences* and *Arrangement*. At the same time, the task Arrangement is divided into three tasks: *Arrange a Trip*, *Arrange a Tourist Package* and *Arrange an Excursion*. Finally, to arrange a trip the user must first Book Transportation and then (>> Enabling relationship) Book Accommodation. On the other hand, regarding to the Broker Agent (BA), Fig. 3 depicts the task taxonomy that represents the requirements of this system. In this case, there is only an actor that can interact with the BA system, which is the TAS system.

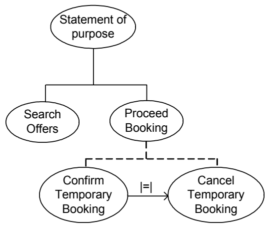


Fig. 3 A Task Taxonomy of the Broker Agent

In the following section we introduce a strategy to describe each identified tasks. To better understand this strategy we show the description of one task of each presented task taxonomy: Book Reservation (from the TAS) and Search Offers (from the BA).

4.2 Task Description

In order to describe the set of tasks detected in the previous stage we extend the traditional descriptions which specify the system actions that are needed to achieve each task. We introduce information about the *interaction between actors and the system*, indicating explicitly when (at which exact moment) it is performed. To do this, we introduce the concept of interaction point (IP). An IP can define two different types of interaction:

(1) *Output Interaction*: the system provides actors with information and/or access to operations which are related to an entity[2]. Actors can perform several actions with both data and operations: they can select information (as a result the system provides them with new information) or activate an operation (as a result the system carries out with an action).

(2) *Input Interaction*: the system is waiting for the user to introduce some data about an entity. The system uses this information to correctly perform a specific action (for instance, to carry out with an on-line purchase with the provided data client). In this case, the only action that the user must perform is to introduce the required data.

In order to perform descriptions based on IPs we propose the use of UML Activity Diagrams [10] as can be seen in Fig. 4 where:

– Each node (activity) represents an IP (solid line) or a system action (dashed line). In addition, IPs are stereotyped with the *Output* or the *Input* keyword to indicate the interaction type.
– In the *Output IPs,* the number of information instances[3] that the IP includes (cardinality) is depicted as a small circle in the top right side of the primitive.
– As far as the *Input IPs,* we said that the data introduced by the user is taken by the system to correctly perform a specific action. To capture that this kind of IPs exclusively depends on a system action (it does not take part in the general process of the task), nodes that represent both elements (input IP and system action) are encapsulated into dashed squares.

[2] Any object of the real world that belongs to the system domain (e.g. customer, product, invoice, etc)

[3] Given a system entity (e.g. customer), an information instance is considered to be the set of data related to each element of this entity (Name: Joseph, Surname: Elmer, Telephone Number: 9658789).

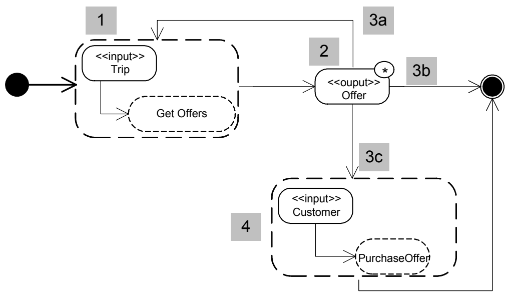


Fig. 4 Book Transportation Business Process

Fig. 4 shows the description of the task *Book Transportation***.** This task begins with the system action *Get Offers* (1). This action temporally books the offers that match with the trip description introduced by the actor (in this case, the internet user). This description is introduced by means of the Input IP defined previously to the system action. Once this action is finished, the task continues with an Output IP, where the system provides the internet user with the list of matched offers (2). From this IP the internet user can either start the process again to refine his/her trip description (3a), reject all the supplied offers and quit (3b) or select one offer (3c). When the internet user selects an offer the system performs the action *Purchase Offer* (4). To perform this action the user must introduce its client information by means of an Input IP. Then the task finishes. Fig. 5 shows the description of the *Search Offers* task. This task starts with the system action *Search Transportation*. This action searches transportation that matches with the trip information provided by the actor (the TAS system) through an Input IP. Next, the system (the Broker Agent) temporally books the transportation to finally conclude the task.

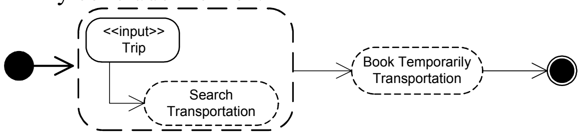


Fig. 5 Search Offers Business Process

In order to make task descriptions easy, details about the information exchanged (in each IP) between the user and the system are not described (we just indicate the entity which the information is related to). This information is specified by means of a technique based on information templates that is next introduced.

4.2.1 Describing the system data

The information that might be stored in the system is represented by means of a template technique that is based on data techniques such as the CRC Card [12]. We propose the definition of an information template (see Fig. 6) for each entity identified in the description of a task. In each template we indicate an identifier, the entity name and a *specific data* section. In this section, we describe the information in detail by means of a list of specific properties associated to the entity. For each property we provide a name, a description and a data type. In addition, we use these templates to indicate the information shown in each IP. For each property we indicate the IPs where it is shown (if there is any). To identify an IP we use the next notation: *Output (Entity, Cardinality)* for Output IPs and *Input (Entity, System Action)* for Inputs IPs.

Identifier:	T1			
Entity:	Customer			
Specific Data:	*Name*	*Description*	*Type*	*IPs*
	Name	Name of the Customer	String	
	Address	Postal Address	String	
	Email	Address of the E-mail	String	
	Birthday Date	Date of the Customer's Birthday	Date	
	Credit Card	Number of the Customer's Credit Card	String	Input(Trip,Pre-Book Offers)
	Phone Number	CD cover frontal	Number	

Fig. 6 Information Template for the Customer entity

According to the template showed in Fig. 6, the information that the system must store about a Customer is (see the specific data section): his/her name, address, email, birthday date, credit card number (which is requested in the IP *Input(Trip, Pre-Book Offers)* and the phone number.

5. Building PIM models from the CIM

Five models should be specified in order to describe the web application at the OOWS PIM level:

(1) the structural view of the system
(2) the external functionality that our system will consume
(3) the business process of the application
(4) the navigational view of the system
(5) the presentation view of the system.

Although there is not an explicit order in which these models might be built, the existing dependencies between them introduce some constraints about the sequence in which some of these views should be built.

Model transformation is applied at this point. From the requirements gathered previously, we proceed to transform them into more detailed models of the web app.

We want to note that CIM models did not specify what part of the system was going to be provided by an external business partner, if any. Nevertheless, we should state at this stage (PIM modeling) which functionality is going to be provided by external parties.

In the following subsections we proceed to model the structure, external functionality, business processes, navigation and presentation views of the TAS case study.

5.1 Structural Modeling

The Structural Model specifies by means of a *UML Class Diagram* the system structure (its classes, operations and attributes) and relationships between classes (specialization, association and aggregation).

This model can be partly obtained (just classes and attributes) from the *Templates* built at the requirements step (see section 4.2.1). As Fig. 7 shows, we have obtained for the TAS case study four classes (Customer, Trip, Broker Agent and Line Trip) that describe the part of the domain that needs to be fully managed by our system.

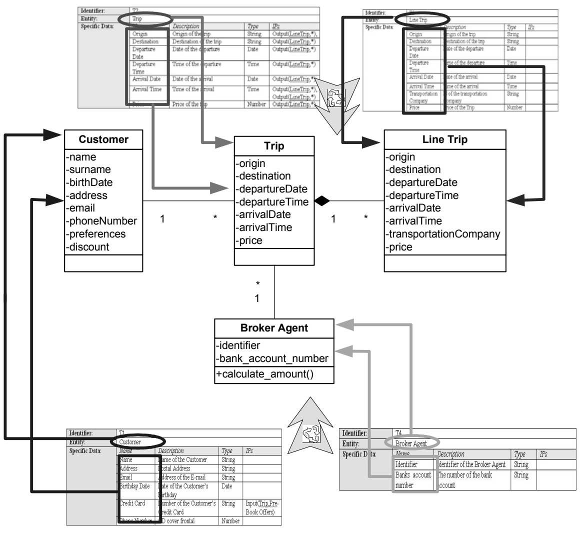


Fig. 7 TAS Structural Model

For instance, the TAS requires keeping customer data (such as his/her name, surname, preferences for travel searches and the kind of discount associated to him/her) among others.

5.2 Services Modeling

To build Web applications that make use of external artifacts (such as components, class libraries, Web services, etc.), we need to represent them at the modeling level. Following this approach we can work with external functionality as if they were native elements of our system. Therefore, we model those external artifacts that are going to interact with our system in a specific model called the *Services Model*. This model has been conceived to specify in a technology independent fashion the external providers based on the functionality that they supply. This specification helps us to easily handle external functionality as if they were part of the native system at high level of abstraction.

In this model we define the services supplied by external providers as well as the set of operations (their interfaces) that they offer. This definition allows us to have a generic description of functionality that is provided at the same time by different partners and in different technologies. There are two main benefits of having a generic representation of external functionality. The former is that the modeling process gets easier because we work with a generic specification (note that we define adaptors to match real external operations with those generic modeled in this model). The latter is that adding or/and removing providers do not have a collateral effect in the rest of models that depend on the services model.

As can be seen in Fig. 8 we have modeled each external system type (the Broker Agent, The Financial Company and the Transportation Company) with the set of operations that they provide.

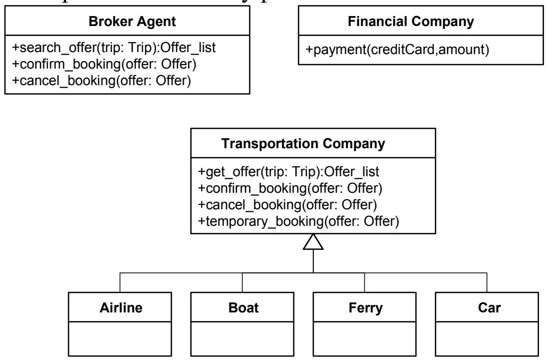

Broker Agent
+search_offer(trip: Trip):Offer_list
+confirm_booking(offer: Offer)
+cancel_booking(offer: Offer)
Financial Company
+payment(creditCard,amount)
Transportation Company
+get_offer(trip: Trip):Offer_list
+confirm_booking(offer: Offer)
+cancel_booking(offer: Offer)
+temporary_booking(offer: Offer)
Airline
Boat
Ferry
Car

Fig. 8 TAS Services Model

5.3 Business Process Modeling

The Business Process Model defines the set of business processes (BP) that characterize the business of the application. The BPs that are defined in this model correspond to processes that describe the flow and operations that made up the system actions (nodes in the task description) detected at the CIM level. In the case of distributed web systems, as it is the case with the TAS, these BPs can be formed not only of activities performed by our system but also from activities carried out by external partners. Therefore, these processes can be made up by internal (implemented by our system) or/and external activities (provided by business partners). To differentiate in this diagram the external activities we mark them with the external stereotype.

For the TAS we have specified two task taxonomies, one to specify the interaction between the user and the TAS system (see Fig. 2) and another to specify the tasks that should perform the Broker Agent that implements our system.
Associated to the first task taxonomy, and taking the task description specified for the *Book Transportation* leaf in the task tree, we proceed to specify/refine the description of the process that defines the system actions (*Get Offers* and *Purchase Offer*) included in the Book Transportation Business Process.

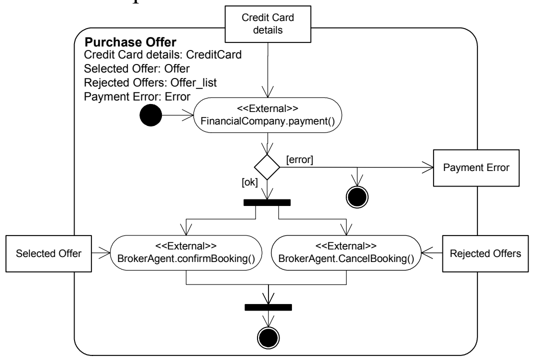

Credit Card details
Purchase Offer
Credit Card details: CreditCard
Selected Offer: Offer
Rejected Offers: Offer_list
Payment Error: Error
<<External>> FinancialCompany.payment()
[error]
Payment Error
[ok]
Selected Offer
<<External>> BrokerAgent.confirmBooking()
<<External>> BrokerAgent.CancelBooking()
Rejected Offers

Fig. 9 Purchase Offer Process

In particular, Fig. 9 depicts the definition of the process for the Purchase Order system action. This process accepts as input the credit card details, the offer to be purchased and the list with the rejected offers. With this information it will proceed with the payment to the corresponding Financial Company. If this activity is correctly performed, the process continues by confirming the selected offer and canceling the temporary booking of discarded offers.

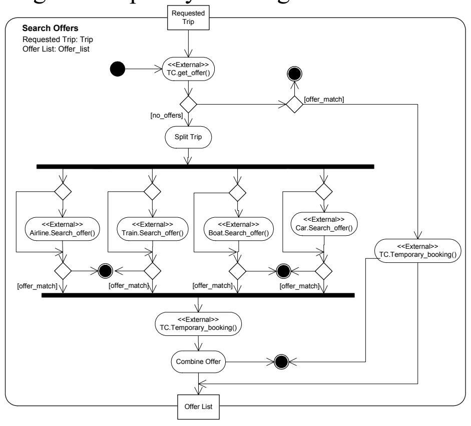

Requested Trip
Search Offers
Requested Trip: Trip
Offer List: Offer_list
<<External>> TC.get_offer()
[offer_match]
[no_offers]
Split Trip
<<External>> Airline.Search_offer()
<<External>> Train.Search_offer()
<<External>> Boat.Search_offer()
<<External>> Car.Search_offer()
<<External>> TC.Temporary_booking()
[offer_match]
[offer_match]
[offer_match]
[offer_match]
<<External>> TC.Temporary_booking()
Combine Offer
Offer List

Fig. 10 Search Offers Process

For the Broker Agent task taxonomy we have defined two tasks, which are *Search Offer* and *Proceed Booking*. Fig. 10 depicts the activity diagram that defines the *Search Offer* process. This process accepts as input a requested trip (including preferences and constraints) and manages to get a set of offers that match with the specified trip.

Fig. 11 defines the process in charge of paying external Broker Agents for the services supplied to our TAS. In this figure, the once-a-month accept time event action generates an output (signal) once a month that is received by the Pay Broker Agent process. At this moment, the process is performed.

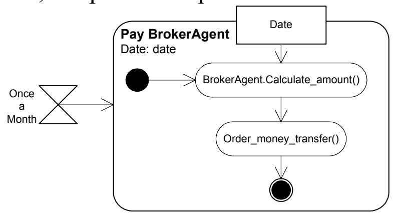


Fig. 11 Pay Broker Agent Process

5.4 Navigational Modeling

Until this point we have already specified the back-end of the system (structure, application logic and external functionality). Nevertheless, as our goal is to build Web applications, it is necessary to specify the front-end of the system. This is done through the Navigational Model.

In the Navigational view we build two models:

(1) the **user diagram**, This model expresses what kind of users can interact with the web application and the system visibility that they should have.

(2) the **navigational model**. This model defines the system visibility for each kind of user in terms of navigational constructs.

5.4.1 User Diagram

To define what kind of users can interact in the system we build the *User Diagram*. This diagram provides mechanisms to properly cope with additional user management capabilities, such as the user specialization that allows defining user taxonomies to improve navigational specification reuse.

As can be seen in Fig. 12, for the TAS case study we have specified two user types, an anonymous user (*Anonymous user*) that do not need to provide information about his/her identity to the system (depicted with a question mark symbol) and a registered user (*Registered customer*), who needs to be identified to connect the system (depicted with a padlock).

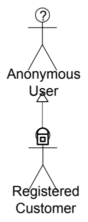


Fig. 12 TAS User Diagram

Once users have been identified, a structured and organized system view for each type must be specified. These specifications are shown next in the Navigational Model.

5.4.2 Navigational Model

The Navigational requirements of the system are defined in the *Navigational model.* In this model we provide a structured and organized view of the system for each user type defined previously in the User Diagram. Navigation requirements are captured in two steps: the "Authoring-in-the-large (global view) and the "Authoring-in-the-small" (detailed view).

For the definition of the navigational global view we take as reference the task taxonomy specified in the requirements modeling step (see Fig. 2). Only that leafs from the task tree whose associated interaction-actor is the user are transformed into Navigational Contexts4 in the Navigational map. In particular, those leafs targeted with *Enabling* relationships are defined in the Navigational Map as *Sequence Contexts* (contexts that can only be accessed via a predefined navigational path by selecting a sequence link). The rest of navigational contexts are defined as *Exploration Contexts*. Exploration contexts are accessible from any node of the application. Fig. 13 shows the global view of the system for the registered customer user type.

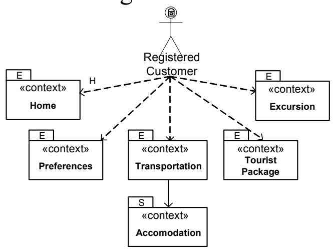


Fig. 13 TAS Navigational Map

Once the global view has been defined we should provide a navigational description for each navigational node (detailed view). Each navigational context is defined as a view over one of the three

[4] Navigational Contexts represent user interaction units that provide a set of cohesive data and operations to perform certain activity.

models presented in the previous sections: *the class diagram, the services model and the business logic model.* On one hand, a view over the class diagram (class view) is defined in terms of the visibility of class attributes, operations and relationships (class views defined by OOWS). On the other hand, when these views are built from operations (defined in the services model) or processes (defined in the business logic model) they are defined in terms of the data returned by these operations/processes (we call it Functional Views). These Functional Views are organized in three sections as follows:

(1) One section to specify the operation/process name, including its input and output parameters.
(2) A second section to specify which data from the data returned by the operation/process is going to be shown in the context.
(3) Finally, in a last section we include the Operations/Processes that can be performed with the data contained in the context.

An example of a view defined over the business process model is depicted in Fig. 14. The way in which a Functional View will be provided graphically is explained next:

(1) If the operation that defines the Functional view requires a set of input parameters these will be asked to the user by means of a input form. If not, the operation is directly executed without providing any data. In Fig. 14 the user should provide data about the trip as well as some constraints and preferences related to it.
(2) As a result of this invocation, this context would filter the data returned by the operation, showing only the data specified at the central section of the Functional view (which are origin, destination, dateDeparture, etc).
(3) Finally, the operations that can be invoked using the data contained in this context are located in the bottom section of the Functional view. In the exemplified Transportation context, the *ProceedBooking* activity is made accessible to proceed with the booking of the selected offer as well as the cancellation of those rejected offers that were temporarily booked. To accomplish this operation an input form is provided to the user in order to fill in the credit card details required for this operation.

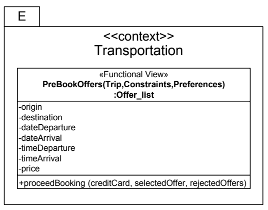


Fig. 14 Transportation Navigational Context

5.5 Presentation Modeling

Once the navigational model has been built, we specify presentational requirements using the *Presentation Model*. Presentation requirements are specified by means of properties that are associated to the primitives of the navigational context. This specification is strongly based on the Navigational Model. It allows us to specify the organization of data included in the Navigational Model.

To define this model we make use of the basic presentation patterns defined by the OOWS approach, which are *Information paging*, *Ordering criteria* and *Information Layout*. Fig. 15 shows the presentation defined for the Transportation context. It defines by means of the Ordering criteria that the data contained in the context must be presented ordered ascendant by price. It also defines the layout in which data must be organized. We decided to show the offer list following the register pattern.

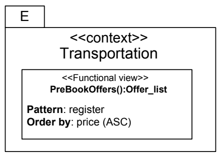


Fig. 15 Transportation Presentation Context

6. Building the PSM for the Services Model

At the PSM level Services Models should be defined as many as different technologies our application is going to interact with. For instance, we should have a Services Model for *Web Services* in case our application needs to interact with partners that provide their functionality by means of this technology.

The PSM for Services Models represent the specific technological aspects of the different technologies. To build these models we take the specific interface specifications that they provide (WSDL for WS or IDL for CORBA).

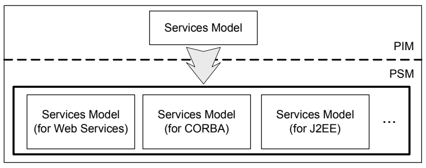
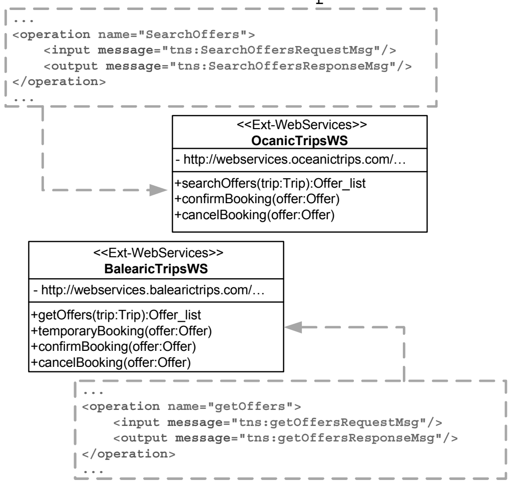

Services Model
PIM
PSM
Services Model (for Web Services)
Services Model (for CORBA)
Services Model (for J2EE)
...

Fig. 16 Services Model at PIM and PSM levels

Once the PSMs are built, in order to solve the differences that arise when integrating various applications (different interfaces, different protocols, different data formats, etc.) we should make use of adapters. These adapters should define the mappings between the operations defined at the PIM (abstract representation) and the ones included at the PSM (specific representation defined by providers). In Fig. 17 we show two Web Services imported from two different travel agencies (BalearicTripsWS and OceanicTripsWS). Both provide a set of operations that fulfil the ones modelled in our PIM Services Model. However, we still need to link the operation/s from each service with the generic ones. For instance, the `search_offer` operation defined for the Broker Agent at the PIM Services Model is fulfilled by the `searchOffers` operation from the OceanicTripsWS and by the `getOffers` and `temporaryBooking` operations from the `BalearicTripWS`.



```
...
<operation name="SearchOffers">
    <input message="tns:SearchOffersRequestMsg"/>
    <output message="tns:SearchOffersResponseMsg"/>
</operation>
...
```

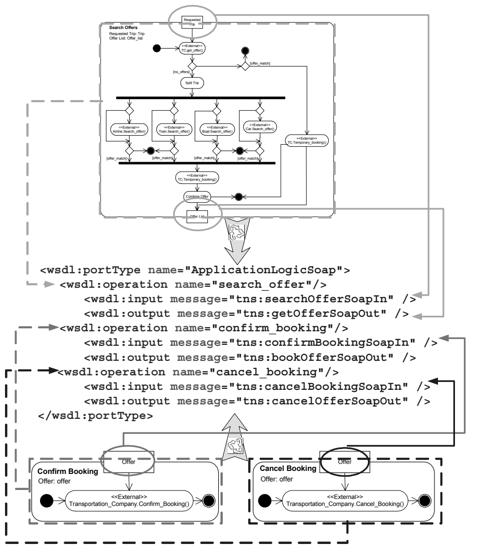
<<Ext-WebServices>>
OcanicTripsWS
- http://webservices.oceanictrips.com/…
+searchOffers(trip:Trip):Offer_list
+confirmBooking(offer:Offer)
+cancelBooking(offer:Offer)
<<Ext-WebServices>>
BalearicTripsWS
- http://webservices.balearictrips.com/…
+getOffers(trip:Trip):Offer_list
+temporaryBooking(offer:Offer)
+confirmBooking(offer:Offer)
+cancelBooking(offer:Offer)

```
...
<operation name="getOffers">
    <input message="tns:getOffersRequestMsg"/>
    <output message="tns:getOffersResponseMsg"/>
</operation>
...
```

Fig. 17 An excerpt of two Imported Web Services

7. Generating a Web Service for the Broker Agent

In order to make available to external parties the Broker Agent implemented in our system, we are going to generate a Web Service (WS) that provides access to its functionality.



```
<wsdl:portType name="ApplicationLogicSoap">
  <wsdl:operation name="search_offer"/>
     <wsdl:input message="tns:searchOfferSoapIn" />
     <wsdl:output message="tns:getOfferSoapOut" />
  <wsdl:operation name="confirm_booking"/>
     <wsdl:input message="tns:confirmBookingSoapIn" />
     <wsdl:output message="tns:bookOfferSoapOut" />
  <wsdl:operation name="cancel_booking"/>
     <wsdl:input message="tns:cancelBookingSoapIn" />
     <wsdl:output message="tns:cancelOfferSoapOut" />
</wsdl:portType>
```

Fig. 18 Application Logic WSDL for the Broker Agent

For the construction of the WSDL definition of the WS we take the business processes defined for the internal Broker Agent in the Business Processes Model. This model was made up of three processes which are going to be included in the WS. These are `search_offer`, `cancel_booking` and `confirm_booking`. These processes definitions include the necessary information for building the WSDL definition (including the input and output parameters of each process).
A graphical schema that defines how to obtain the WSDL definition is provided in Fig. 18.

8. Generating the Interaction with External Parties

Depending on the technology that a provider uses to export its functionality, we should provide an appropriate solution for each case. As Fig. 19 shows, for each instance from each PSM Services Model we should build the required client artifact in charge of interacting with the corresponding service provider.

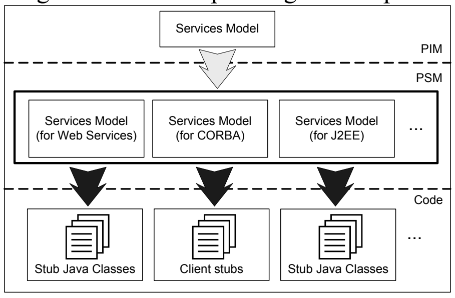

Services Model
PIM
PSM
Services Model (for Web Services)
Services Model (for CORBA)
Services Model (for J2EE)
...
Code
Stub Java Classes
Client stubs
Stub Java Classes
...

Fig. 19 Generating the appropriate Interaction

Based on the interfaces supplied by their respective providers we can generate the corresponding clients to fulfill the interaction with the external partner.

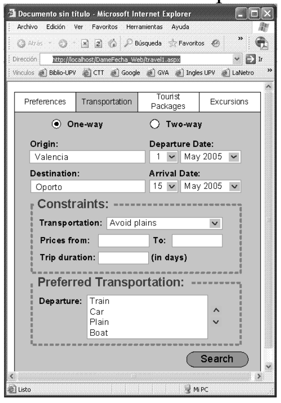
For instance, when functionality is provided following the Web Services model we generate from the associated WSDL definition the necessary client stubs to consume the supplied operations via SOAP, REST or XML-RPC depending on the characteristics of the available WS.

9. Generating the Web Interface

Web interfaces are generated taking as input the information modeled in the Navigational and Presentation Model. For the running example, we have modeled in section 5.4 the Transportation context as a functional view for the PreBookOffer operation.

The web interface that implements the Transportation context includes (as the top of Fig. 20 shows) direct access to those Navigational Contexts that we have defined as Exploration Contexts (Preferences, Transportation, Tourist Packages and Excursions).

The set of input parameters of this operation define the information that is required to the user in order to provide the corresponding data (an offer list provided by external broker agents). In Fig. 20 we can see how the input parameters (trip details, constraints and preferred transportation) are included in the Transportation context to allow the user to specify the parameters for the PreBookOffer operation.



Documento sin título - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Búsqueda Favoritos
Dirección http://localhost/DameFecha_Web/travel1.aspx Ir
Vínculos Biblio-UPV CTT Google GVA Ingles UPV LaNetro
Preferences
Transportation
Tourist Packages
Excursions
Search
Listo
Mi PC

(● One-way) ○ Two-way

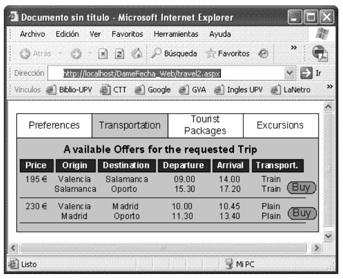
Origin: Valencia
Departure Date: 1 May 2005
Destination: Oporto
Arrival Date: 15 May 2005

Constraints:
Transportation: Avoid plains
Prices from: To:
Trip duration: (in days)

Preferred Transportation:
Departure: Train / Car / Plain / Boat

Fig. 20 Web Form for the Transportation Context

When this operation is executed, the context is shown as a Web page (see Fig. 21) that includes the offer list gathered from the Broker Agents. In Fig. 21 we can identify the register pattern applied to the retrieved offer list from the Broker Agents that work with our system.



Documento sin título - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Búsqueda Favoritos
Dirección http://localhost/DameFecha_Web/travel2.aspx Ir
Vínculos Biblio-UPV CTT Google GVA Ingles UPV LaNetro
Preferences
Transportation
Tourist Packages
Excursions
Buy
Buy
Listo
Mi PC

Available Offers for the requested Trip

Price	Origin	Destination	Departure	Arrival	Transport.
195 €	Valencia	Salamanca	09.00	14.00	Train
	Salamanca	Oporto	15.30	17.20	Train
230 €	Valencia	Madrid	10.00	10.45	Plain
	Madrid	Oporto	11.30	13.40	Plain

Fig. 21 Web Page with the Offer list

10. Conclusions and Further Work

In this work we have presented through the TAS case study the set of models that need to be built in order to develop a Web application that integrates functionality from external parties.

This approach is being incorporated to the OO-Method CASE tool (the software automatic production environment that gives support to the OO-Method [13]).

As further work we have planned to define the transformations that generate automatically the Web Services not only to provide the business logic of the application, moreover we plan to provide the information gathered also in the Navigational and presentation models.

In order to define transformations between models defined at CIM and PIM level we follow a strategy based on graph transformations. After completing the preliminary PIM models obtained from this first transformation, we plan to use the OlivaNova tool to obtain the application code for a specific platform.

10. References

[1] D. Schwabe, and G. Rossi, "An Object Oriented Approach to Web-Based Application Design", *Theory and Practice of Object System 4(4),* Wiley and Sons, New York, 1998, ISSN 1074-3224.

[2] O. De Troyer and C. Leune, "WSDM: A user-centered design method for Web sites", *In Proc. of the 7th International World Wide Web Conference*, 1998.

[3] N. Koch and M. Wirsing, "Software Engineering for Adaptive Hypermedia Applications", *In 3rd Workshop on Adaptive Hypertext and Hypermedia*, 2001.

[4] N. Koch, A. Kraus, C. Cachero and S. Meliá, "*Integration of Business Processes in Web Application Models*". Journal of Web Engineering. Vol. 3, No. 1 (2004)

[5] M. Brambilla, S. Ceri,, S. Comai, P. Fraternali and I. Manolescu, "*Model-driven Development of Web Services*

and Hypertext Applications", SCI2003, Orlando, Florida, July 2003

[6] P. Dolog, "Model-Driven Navigation Design for Semantic Web Applications with the UML-Guide". In Maristella Matera and Sara Comai (eds.), Engineering Advanced Web Applications

[7] R. Tongrungrojana and David Lowe, "*WIED: A Web Modelling Language for Modelling Architectural-Level Information Flows*". Journal of Digital Information, Vol 5 Issue2.

[8] J. Fons, V. Pelechano, M. Albert and O. Pastor, "Development of Web Applications from Web Enhanced Conceptual Schemas", *Proc. Of the International Conference on Conceptual Modelling, 22nd Edition, ER'03*, Chicago, EEUU, 2003, pp. 232-245.

[9] MDA Guide Version 1.0.1.

[10] Object Management Group. Unified Modeling Language (UML) Specification Version 2.0 Final Adopted Specification. www.omg.org, 2003.

[11] F. Paternò, C. Mancini and S.Meniconi, "ConcurTaskTrees: a Diagrammatic Notation for Specifying Task Models", INTERACT'97, Chapman & Hall, 1997, pp. 362-369.

[12] Wirfs-Brock, B. Wilkerson, and L. Wiener, "*Designing Object–Oriented Software*.", *Prentice–Hall*, 1990.

[13] O. Pastor, J. Gómez, E. Insfrán and V. Pelechano, "The OO-Method Approach for Information Modelling: From Object-Oriented Conceptual Modeling to Automated Programming", *Information Systems Elsevier Science*, 2001, Vol. 26, Number 7, pp. 507-534

The PIM to Servlet-Based PSM Transformation with OOHDMDA

Hans Albrecht Schmid
University of Applied Sciences Konstanz,
Brauneggerstr. 55
D 78462 Konstanz
xx49-07531-206-631 or -500
schmidha@fh-konstanz.de

Oliver Donnerhak
University of Applied Sciences Konstanz,
Brauneggerstr. 55
D 78462 Konstanz
xx49-07531-206-631 or -500
o.donnerhak@kinemotion.de

ABSTRACT
OOHDMDA generates servlet-based Web applications from OOHDM. An OOHDM application model built with a UML design tool is complemented with the recently proposed behavioral OOHDM semantics to serve as a PIM. This paper describes the transformation from the PIM to a servlet-based PSM, which have a great semantic distance. Therefore, the navigational transformation applies intelligent techniques in order to avoid a very great complexity. The resulting transformation rules are quite simple. The XMINavigational-Transformer implements each rule by a transformation class that uses the services provided by an XMI parser; it transforms the PIM XMI-file into a PSM XMI-file.

1. INTRODUCTION
The Object-Oriented Hypermedia Design Method OOHDM by Schwabe and Rossi [SR98] is a modeling and design method, which describes hypermedia-based Web applications by an object model on three levels: the conceptual level, the navigational level, and the interface level. OOHDM may be considered as a platform-independent domain-specific language for Web applications that provides an object model, in contrast to other Web application modeling languages.

A domain-specific language that is to be used as a platform-independent model (PIM) for a model-driven architecture (MDA) [OMG MDA], should have a well-defined formal semantics. Therefore, we use an OOHDM application model only as a base PIM, adding to it the behavioral semantics definition of OOHDM core features and business processes, proposed by Schmid and Herfort [SH04]. The behavioral semantics definition derives the application-related OOHDM classes from behavioral model classes with a fixed predefined behavioral semantics, so that they are well-defined and executable.

Schmid [S04] gives an overview over OOHDMDA, which covers all core constructs of OOHDM together with business processes [SR04]. This paper describes in detail the transformation of the PIM to a servlet-based PSM, and in particular the transformation rules used to perform the transformation. This paper presents the transformation of all core constructs of OOHDM, including fixed page, dynamic page and advanced navigation, which are considerably more complex than the usual MDA transformations [JCP01] [KWB04]. The transformation generates from an OOHDM application model and the behavioral semantics model a servlet-based Web application front-end, which accesses backend classes.

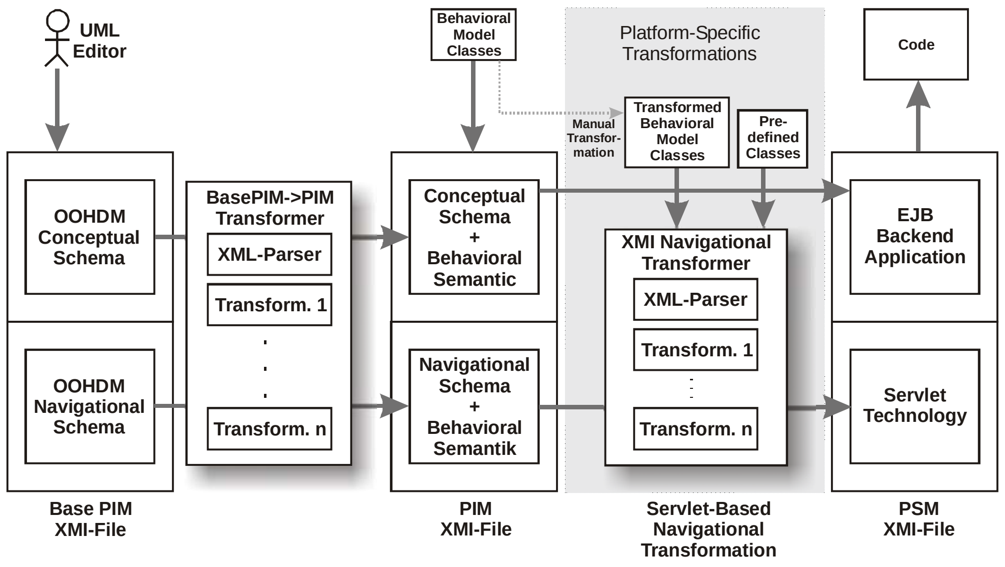
After an overview on the MDA-process with OOHDM in section 2, we describe the PIM for dynamic navigation with the behavioral semantics definition in section 3. Section 4 presents the transformation to a servlet-based PSM. Section 5 presents the transformation rules in detail. Section 6 shows how the transformations rules are implemented; section 7 presents an optimization of the transformation that reduces the number of the classes to be transformed, and section 8 presents related work

2. MDA PROCESS
2.1 Base PIM and PIM
A Web application designer designs with with OOHDMDA (see Figure 1) the OOHDM conceptual and navigational schema of a Web application as the base PIM for the MDA process, using any UML-based design tool, like Rational Rose or Poseidon, that produces an XMI-file as output. The designer has to mark the application-related OOHDM classes with a stereotype indicating the model class, from which the OOHDM class is derived (see [SH04]).

The **Base PIM to PIM transformation** transforms the output XMI-file of the design tool to a XMI-file with a modified UML class diagram. The main transformation rules are: replace navigational links by model classes for navigational links; derive the base PIM classes from model classes according to the stereotype; add the model classes to the PIM; add directed associations from nodes to the associated conceptual schema entities.

The OOHDM conceptual and navigational schemas represent two different, relatively independent aspects of a Web application, the Web front-end, and the application backend. Consequently, we partition also the PIM into a conceptual PIM sub-model and the navigational PIM sub-model (see Figure 1).

2.2 Navigational Transformation
The conceptual and navigational PIM to PSM transformation are completely independent, except for operation invocations of conceptual PSM objects from the navigational PSM, where the kind of invocation may vary. Thus, you may select and combine the implementation technology and platform of the conceptual PSM and the navigational PSM quite independently, as [S04] shows.



UML Editor
Behavioral Model Classes
Platform-Specific Transformations
Code
Manual Transfor-mation
Transformed Behavioral Model Classes
Pre-defined Classes
OOHDM Conceptual Schema
BasePIM->PIM Transformer
XML-Parser
Transform. 1
Transform. n
Conceptual Schema + Behavioral Semantic
XMI Navigational Transformer
EJB Backend Application
OOHDM Navigational Schema
Navigational Schema + Behavioral Semantik
Servlet Technology
Base PIM XMI-File
PIM XMI-File
Servlet-Based Navigational Transformation
PSM XMI-File

Fig. 1. Conceptual and navigational Base PIM, PIM and transformation to servlet-based navigational PSM with XMINavigationalTransformer

For example, you may transform the conceptual PIM into an EJB-based conceptual PSM in order to implement the application backend with Enterprise JavaBeans (EJB). This conceptual transformation, which is not very complex, transforms the different categories of objects from the conceptual schema into different kinds of Enterprise JavaBeans, as described shortly in [S04a].

This paper focuses on the navigational transformation from the navigational PIM into a servlet-based navigational PSM, both represented by files in XMI format. It is described by transformation rules.

Since we could not find a transformation tool to be parameterized with the transformation rules, we constructed an XMINavigationalTransformer that implements each rule by a transformation class that uses the services provided by an XMI parser; it transforms the PIM XMI-file into a PSM XMI-file.

The simple PSM-code transformation generates from a PSM XMI-file executable Java code, that is Java servlets and classes, which work quite efficiently.

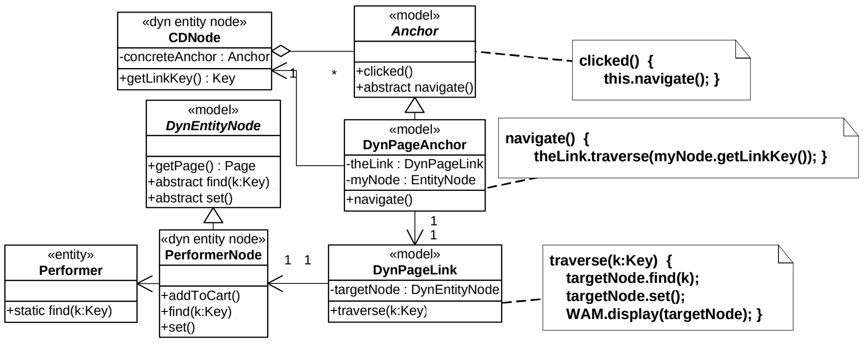
3. PIM MODEL CONSTRUCTS FOR FIXED, DYNAMIC AND ADVANCED NAVIGATION

The OOHDM behavioral semantics derives the OOHDM application model from behavioral model classes. That means conceptual schema entities, like CD, are derived from a model class, like Entity or subclasses, and navigational schema nodes, like CDNode, from a model class, like Node or subclasses. Model classes collaborate with a Web Application virtual Machine (WAM), which models basic Web-browser characteristics, i.e. HTTP-HTML characteristics, as seen from a Web application. Both model classes and WAM have a well-defined behavioral semantics [SH04].

Class Node defines the operations: getPage(): Page, getField(n:Name): Value, setField(n: Name, v: Value), getFieldNames(): Name [], which are mainly used by the WAM to display the content of a page. A Node refers to the entity or entities it displays, and contains an array of InteractionElements like Anchor’s or Button’s, and a Page. Node has subclasses FixedEntityNode and DynEntityNode that represent pages with a fixed content and dynamically generated content.

We distinguish two kinds of navigation, navigation to a Web page with fixed content and dynamic content, i.e. navigation to a FixedEntityNode and DynEntityNode. Advanced navigation allows a user to trigger an action that enters or updates information in entities from the conceptual schema.

We present two examples: the PIM model construct for dynamic navigation from CDNode to PerformerNode; and the PIM model construct for advanced navigation when a user triggers the execution of the addToCart-method of CDNode. For an easier understanding, we present both PIM model constructs independently from each other; that means we assume that CDNode allows either dynamic navigation or advanced navigation. When CDNode allows to do both dynamic navigation and advanced navigation, both PIM model constructs are superimposed such that there is only a single class CDNode with the union of the class members.

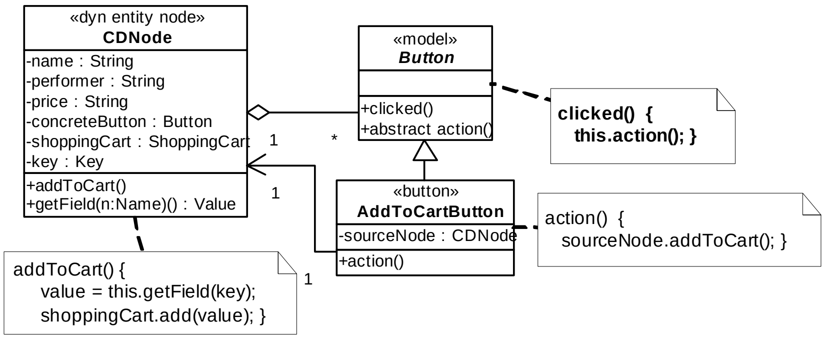


```
clicked() {
    this.navigate(); }
```

```
navigate() {
    theLink.traverse(myNode.getLinkKey()); }
```

```
traverse(k:Key) {
    targetNode.find(k);
    targetNode.set();
    WAM.display(targetNode); }
```

Fig. 2. PIM for dynamic navigation from CDNode to PerformerNode



```
clicked() {
  this.action(); }
```

```
action() {
  sourceNode.addToCart(); }
```

```
addToCart() {
    value = this.getField(key);
    shoppingCart.add(value); }
```

Fig. 3. PIM for advanced navigation: executing the addToCart-action triggered by AddToCartButton

3.1 Dynamic Page Navigation

Figure 2 shows the PIM model construct for dynamic navigation over a link from (the user-defined classes) CDNode to PerformerNode. The source node of the link, like CDNode, references a DynPageAnchor that references a DynPageLink, which references the target node of the link, a DynEntityNode like PerformerNode. These references are set by constructor parameters when the model classes are configured to work together.

The WAM has the attribute currentNode, which references the currently displayed Node. When a user clicks at an InteractionElement of the currently displayed Web page, like the anchor of a dynamic link on the CD Web page, the WAM calls the clicked-operation of the corresponding InteractionElement of the currentNode, like that of DynPageAnchor of CDNode, which forwards the call to the navigate operation. The navigate-operation fetches the key of the dynamic content that the target node should display, from the source node CDNode (referenced by attribute myNode), calling its getLinkKey-method that returns a key, like a Performer name. Then it calls the traverse-operation, passing the key as a parameter.

The traverse-operation of DynPageLink calls the find-operation of its target node, like PerformerNode with the key as a parameter, and then its set-operation so that the target node sets its dynamically generated content. Last, traverse calls the display-operation of the WAM with the target node as a parameter. The method display(n: Node) sets that node n as the current node and calls its getPage-operation to display. the page.

3.2 Fixed Page Navigation

Fixed page navigation is a simplified dynamic page navigation. The main difference is that no key is required to identify the content of the page, since the content is always the same. Therefore, the PIM for fixed page navigation is similar to the PIM for dynamic page navigation. The differences are that the class FixedPageLink (that replaces DynPageLink) has a traverse-method without a key-parameter, which calls directly the display-method of the WAM (without the find- and set-calls of the target node); and that the the navigate-method of class FixedPageAnchor (that replaces DynPageAnchor) does not pass a key parameter with the call of traverse.

3.3 Advanced Navigation

Figure 3 presents the PIM model construct for advanced navigation (for details, see [SH04]). Advanced navigation allows a user to trigger an atomic action by pressing a button on a Web page. An atomic action enters or edits information in a Web application, modifying the state of application objects that are modeled in the conceptual schema.

For example, consider the addToCart operation of the CDNode in Figure 3, which is triggered by the

AddToCartButton. The navigational PIM (see Figure 3) shows the model class Button with the operations clicked and action. A derived application-specific class, like AddToCartButton, implements the action-method, which calls an operation of the source node, like addToCart of CDNode.

When the WAM displays a Web page, i.e. a node, and a user clicks at a button on this page, the WAM calls the clicked-operation of the corresponding InteractionElement of the currentNode, like Button. The clicked-method forwards the call to the action-method, which forwards the call to the addToCart-method of the CDNode. This method fetches the value from the key-field of the (currently presented) CD and sends the message add(value) to the ShoppingCart object, which changes the state of the shopping cart. Note that the execution of an atomic action does not imply the navigation to another node.

4. SERVLET-BASED NAVIGATIONAL PSM

This section gives an overview on the servlet-based PSM for fixed page, dynamic page and advanced navigation. The semantic distance between PIM and PSM is very great, in contrast to what is usual with MDA. Therefore, the navigational transformation applies intelligent techniques in order to avoid a very great complexity. The resulting transformation rules are quite simple. Details on them are given in section 5.

A servlet connects the backend application with the Web; it runs on a Web server, receiving an HTTP request as a parameter of a doGet- or similar operation, and sending out a HTTP response as a result of the operation. The doGet-method analyses the user input and creates the new Web page as output.

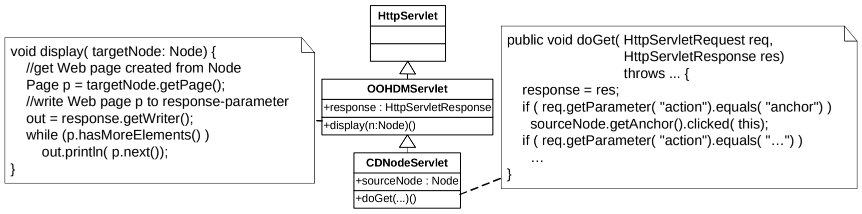
The processing performed by a servlet is similar to the processing performed by the WAM in the navigational PIM. The doGet-method of a servlet is triggered by a user interaction and reacts on that interaction by creating a Web page as a response, in the same way, as the OOHDM behavioral model is triggered by the WAM on a user interaction and creates and displays a mask for a Web page on the WAM.

As a consequence, the **navigational transformation** replaces the WAM by a servlet. The **EntityNodeToServlet** transformation rule, which applies to all entity nodes, generates from each PIM Entity Node class, like CDNode, a PSM servlet class, like CDNodeServlet (see Figure 5), that has a reference to the node, like PSM::CDNode, which is not modified from the PIM. When a user presses an interaction element of the Web page, the doGet-method of the generated servlet analyses the response parameter and calls the clicked-method of the pressed InteractionElement of the referenced node (see Figure 4).

The **navigational transformation** modifies also the navigational PIM classes like Anchor, PageAnchor, and Link, such that the new page is not displayed by the WAM, but put into the response-parameter of the doGet-method. Doing that straightforwardly would result in the navigational PSM being very different from the navigational PIM, which would make the navigational transformation a complex expenditure. To keep the transformation as simple as possible, we developed the solution that the servlet provides, similarly as the WAM, a display-method putting the node into the response parameter.

Since that responsibility is identical for all node servlets, we introduce with the **EntityNodeToServlet** transformation rule the PSM class OOHDMDAServlet, extending HttpServlet, as common superclass of all NodeServlet classes.(see Figure 4). Its method display(targetNode: Node) gets the associated Page from the parameter targetNode; since it has no direct access to the response parameter of doGet, it writes the Page to the member variable “response” that refers to the HttpResponse, after an assignment by the doGet-method (see Figure 4). Thus, the page contained in the parameter targetNode is put as content into the response parameter and displayed as Web page at the return from the doGet-method call.

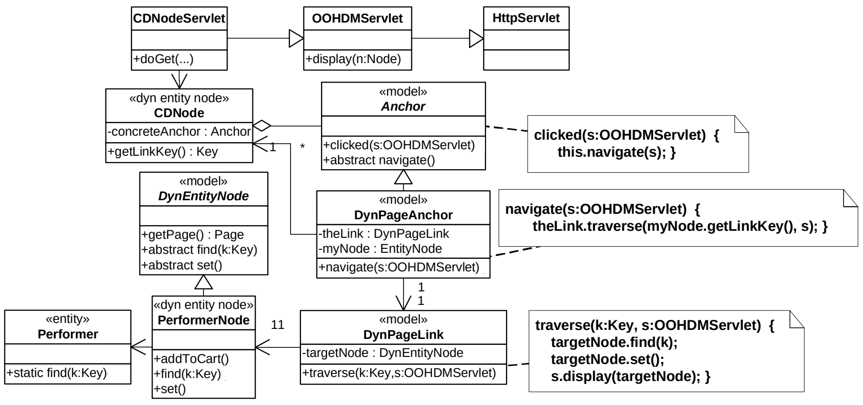
The navigational transformation rules **InteractionElement**, **Anchor**, **Button**, **PageAnchor**, **Link, Button** and **ActionUserButton** modify each the clicked-method of the class InteractionElement, Anchor or Button, the navigate-method of the class FixedPageAnchor or DynPageAnchor, the traverse-method of FixedPageLink or DynPageLink, and the action-method of UserButton, so that the traverse-method of FixedPageLink or DynPageLink or the action-method of UserButton can call the display-method provided by the servlet: a reference to the servlet is added as an additional parameter to these methods and forwarded from call to call.



```
void display( targetNode: Node) {
   //get Web page created from Node
   Page p = targetNode.getPage();
   //write Web page p to response-parameter
   out = response.getWriter();
   while (p.hasMoreElements() )
      out.println( p.next());
}
```

```
public void doGet( HttpServletRequest req,
                   HttpServletResponse res)
                   throws ... {
   response = res;
   if ( req.getParameter( "action").equals( "anchor") )
     sourceNode.getAnchor().clicked( this);
   if ( req.getParameter( "action").equals( "…") )
     …
}
```

Fig. 4. PSM-classes CDNodeServlet with doGet-method analyzing request parameter and calling clicked-method of the clicked-at InteractionElement, and OOHDMDAServlet

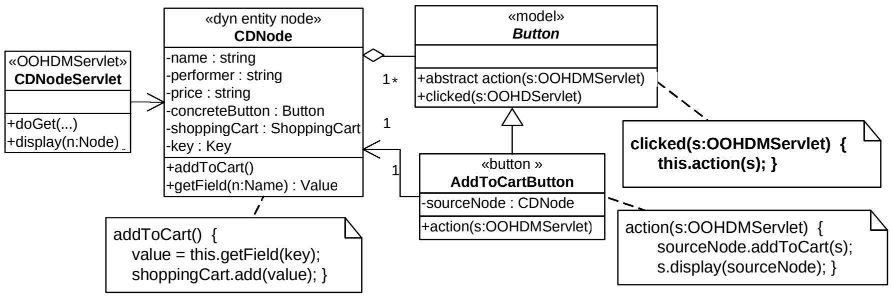


```
clicked(s:OOHDMServlet) {
    this.navigate(s); }
```

```
navigate(s:OOHDMServlet) {
    theLink.traverse(myNode.getLinkKey(), s); }
```

```
traverse(k:Key, s:OOHDMServlet) {
    targetNode.find(k);
    targetNode.set();
    s.display(targetNode); }
```

Fig. 5 Servlet-based PSM for dynamic navigation



```
clicked(s:OOHDMServlet) {
    this.action(s); }
```

```
addToCart() {
    value = this.getField(key);
    shoppingCart.add(value); }
```

```
action(s:OOHDMServlet) {
    sourceNode.addToCart(s);
    s.display(sourceNode); }
```

Fig. 6. Servlet Servlet-based PSM for advanced navigation

4.1 Servlet-Based PSM for Dynamic Navigation

Figure 5 shows the PSM construct for dynamic navigation that is the result of the transformation. The method doGet of CDNodeServlet, which has a reference to CDNode, calls the clicked-method of the corresponding interaction element of CDNode, which is a DynPageAnchor, passing a reference to the servlet as a parameter. The transformed behavioral model classes collaborate in the same way as described in section 3, passing additionally a reference to CDNodeServlet as a parameter. The traverse-method calls the find- and set-method of the target node so that PerformerNode gets the dynamic page content from the DynEntity Performer, and inserts it into the DynPage that contains already the static HTML page content. Then, traverse calls the display-method of CDNodeServlet with PerformerNode as a parameter.

The navigational PSM for fixed navigation is similar to the navigational PSM for dynamic navigation, except for the differences between the respective PIMs described in section 3.

4.2 Servlet-Based PSM for Advanced Navigation

Figure 6 shows the PSM construct for advanced navigation that is the result of the transformation. The method doGet of CDNodeServlet, which has a reference to CDNode, calls the clicked-method of the corresponding interaction element of CDNode, which is a Button, passing a reference to the servlet as a parameter. The clicked-method calls the action-method of AddToCartButton, passing a reference to the servlet as a parameter, which calls in turn the addToCart-method of CDNode. This method gets the key that identifies the CD presented to the user, and calls the add-method of the ShoppingCart object.

5. TRANSFORMATION RULES FOR THE NAVIGATIONAL TRANSFORMATION

The navigational transformation from the PIM to the servlet-based PSM is formed by a set of independent atomic transformations. We describe a transformation in this section for a better understanding by a semi-formalized transformation rule. A transformation rule has

1. a name
2. a source, which describes the PIM class or classes to be transformed. It indicates the class name and/or the stereotype of the class to be transformed, like <<DynEntityNode>>, and possibly additional selection criteria
3. a target, which describes the transformed PSM class or classes
4. a ToDo section that describes the modifications that are to be applied to the PIM class or classes to generate the PSM class or classes
5. a graph section. It represents on the left the source graph with the PIM classes and associations among them, and on the right the destination graph with the PSM classes and associations among them.

We present all transformation rules that the navigational transformation uses to transform the PIM for fixed page, dynamic page and advanced navigation. The same rules apply for fixed page and dynamic page navigation, as Table 1 shows, since the differences among them are expressed by different OOHDM and behavioral model classes, and not generated by the MDA transformations. Advanced navigation shares two rules with them and adds two more rules. Table 1 gives an overview on the transformation rules and for which kind of navigation they are used.

Area of Use Rule	FixedPage Navigation	DynPage Navigation	Advanced Navigation
EntityNode ToServlet	**X**	**X**	**X**
InterAction Element	**X**	**X**	**X**
Anchor	**X**	**X**	
PageAnchor	**X**	**X**	
Link	**X**	**X**	
Button			**X**
Action UserButton			**X**

Table 1 Overview on the transformation rules and their use

The transformation rules are presented in an abbreviated form in Figures 7 and 8.

The transformation rule **EntityNodeToServlet** (see Figure 7) is applied to all nodes derived from FixedEntityNode or DynEntityNode. It generates from a class PIM::<AnyClass>

- an unmodified class PSM::<AnyClass>
- a class PSM::OOHDMDAServlet (that is the same for all entity nodes)
- and a class PSM::<AnyClass>Servlet for which a doGet-method is generated as described in section 4 (see Figure 4): it analyses the request parameter to determine the anchor or button etc. clicked-at, and invokes the corresponding clicked-method of the associated node.

The transformation rule InteractionElement (see Figure 8 a) is applied to the abstract base class InteractionElement from which Anchor, Buton, etc. are derived. Note that the class InteractionElement was left off in the class diagrams.

The transformation rule Anchor is applied to the class PIM::Anchor, PageAnchor is applied to PIM::FixedPageAnchor or PIM::DynPageAnchor, Link is applied to PIM::FixedPageLink or PIM::DynPageLink (see Figure 8 b-d).

The transformation rules Anchor and PageAnchor add each a parameter of type OOHDMDAServlet to the methods: clicked of Anchor, respectively navigate of FixedPageAnchor or DynPageAnchor, and modify the behavioral semantics of these methods so that each forwards the newly added parameter to the navigate-, respectively traverse-method. The transformation rule Link adds a parameter of type OOHDMDAServlet to the traverse-method, and replaces the WAM as the receiver of the display-call by the servlet parameter.

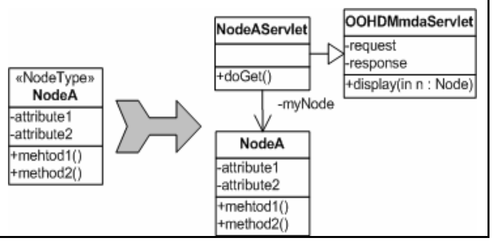
Name	**EntityNodeToServlet**
Source	PIM::<AnyClass> with stereotype <<DynEntityNode>> or <<FixedEntityNode>>
Target	PSM::<AnyClass> and PSM::<AnyClass>Servlet and PSM::OOHDMDAServlet
ToDo	no modifications in class <AnyClass> generate OOHDMDAServlet generate a servlet class <AnyClass>Servlet derived from OOHDMDAServlet, which has a reference to <AnyClass> generate case-distinction in doGet-method
Graph	 «NodeType» NodeA -attribute1 -attribute2 +mehtod1() +method2() NodeAServlet +doGet() OOHDMmdaServlet -request -response +display(in n : Node) -myNode NodeA -attribute1 -attribute2 +mehtod1() +method2()

Figure 7 Navigational transformation rule **EntityNodeToServlet**

There is only one transformation rule that is applied to transform both the navigate-methods of the classes FixedPageAnchor and DynPageAnchor, though both methods have a different number of parameters and a different behavioral semantics. The same holds for the traverse-method of the classes FixedPageLink and DynPageLink.

Name	**InteractionElement**
Source	PIM::InteractionElement
Target	PSM::InteractionElement
ToDo	add a parameter of type OOHDMDAServlet to method clicked()
Graph	not shown, since trivial

Figure 8 a Navigational transformation rule InteractionElement

Name	**Anchor**
Source	PIM::Anchor
Target	PSM::Anchor
ToDo	add a parameter of type OOHDMDAServlet to method clicked(); modify behavioral semantics of clicked to forward the added parameter with call of navigate-method
Graph	not shown, since trivial

Figure 8 b Navigational transformation rule Anchor

Name	**PageAnchor**
Source	PIM::FixedPageAnchor or PIM::DynPageAnchor
Target	PSM::FixedPageAnchor or PSM::DynPageAnchor
ToDo	add a parameter of type OOHDMDAServlet to method navigate; modify behavioral semantics of navigate to forward the added parameter
Graph	not shown, since trivial

Figure 8 c Navigational transformation rule PageAnchor

Name	**Link**
Source	PIM::FixedPageLink or PIM::DynPageLink
Target	PSM::FixedPageLink or PSM::DynPageLink
ToDo	add a parameter of type OOHDMDAServlet to method traverse() modify behavioral semantics code of traverse to replace the WAM as receiver of the display-call by the newly introduced parameter
Graph	not shown, since trivial

Figure 8 d Navigational transformation rule Link

Name	**Button**
Source	PIM::Button
Target	PSM::Button
ToDo	add a parameter of type OOHDMDAServlet to method clicked(); modify behavioral semantics of clicked to forward the added parameter with call of action-method
Graph	not shown, since trivial

Figure 8 e Navigational transformation rule Anchor

Name	**ActionUserButton**
Source	PIM:: <AnyClass> with stereotype <<Button>>
Target	PSM:: <AnyClass>
ToDo	add a parameter s of type OOHDMDAServlet to method action; modify behavioral semantics code of action-method to add at the end a display-call with s as a parameter and the sourceNode as receiver
Graph	not shown, since trivial

Figure 8 f Navigational transformation rule UserButton

6. IMPLEMENTATION OF THE TRANSFORMATION RULES

The navigational transformation is performed by the XMINavigationalTransformer (see Figure 1). It has as input and output each an XMI file. From the output file, executable Java classes are generated.

We programmed the described transformation rules in the XMINavigationalTransformer, since we could not find a transformation tool that could handle the described transformations, which are relatively complex.

The XMINavigationalTransformer contains an XML parser, and for each transformation rule a C# class implementing that rule. A transformation class uses query functions supplied by the XML parser to find a PIM class that meets the criteria described in the source part of the transformation rule. Then, a further query function is used to find the part of the class that is to be modified, like a certain method. When e.g. a parameter is to be added, a prefabricated XMI template with placeholders is used, that means modified and inserted into the XMI tree.

We selected C# as a programming language for the transformation classes since it provides a rich XML functionality. As a consequence, the transformation classes are short, simple and well structured.

Our experience is that it is much simpler to program the transformation rules than to use a tool to implement them.

7. TRANSFORMATION OPTIMIZATION

The PIM behavioral model classes like PIM::InteractionElement, PIM::Anchor, PIM::FixedPageAnchor, PIM::DynPageAnchor, PIM::FixedPageLink, PIM::DynPageLink and PIM::Button have a fixed predefined semantics. As a consequence, it is not necessary to transform each time with each transformation process, when we transform the OOHDM application-model classes.

Therefore, we create once pre-transformed PSM classes: PSM::InteractionElement, PSM::Anchor, PSM::FixedPage Anchor, PSM::DynPageAnchor, PSM::FixedPageLink, PSM::DynPageLink, and PSM::Button; and add them after the navigational transformation to the transformed OOHDM application-model classes. Since the class PSM::OOHDMDAServlet is the same for all nodes, we do not need to generate it with each transformation process, but provide it also as a pre-transformed PSM class.

8. RELATED WORK

Different Web application design methods, like WebML by Ceri, Fraternali, and Paraboschi [C00], W2000 by Baresi, Garzotto, and Paolini [B00], UWE by Koch and Kraus [KKCM03], OO-H by Cachero and Melia [KKCM03], and OOWS by Pastor, Fons and Pelechano [PFP03], generate code from the Web page design or a design model.

We distinguish a model-based process from a code-generation process. A model-based process transforms a formal PIM model into a formal PSM model. In contrast, a code-generation process does not use a formal PIM model, but an informal model, like e.g. a user design of a Web page and a verbal semantics of that design.

We classify the approach taken by WebML and W2000 as a code generation process, and the approach taken by UWE, OO-H, and OOWS together with our approach as a model-based process. UWE and OO-H use UML as a formal PIM model language, and OOWS captures functional system requirements formally to construct from them the Web application.

Taguchi, Jamroenderarasame, Asami and Tokuda distinguish and compare tow code-generation approaches, the annotation approach and the diagram approach for the automatic construction of Web applications [TJAT04].

9. CONCLUSIONS

We have presented how the OOHDMDA approach generates servlet-based Web applications from an OOHDM design model. OOHDMDA comprehends all core constructs of OOHDM and the business process extension [SR04]. OOHDMDA may be complemented easily with concepts currently not included, like navigational contexts, once a behavioral semantics is defined.

The base PIM to PIM transformation, which is a small effort, is done manually. The tools provided to perform the PIM to PSM transformation and the PSM to code transformation are running.

OOHDM is used together with the recently proposed behavioral semantics model as a PIM. The advantages of a model-based transformation process with a formally defined domain-specific language as a PIM are:

1. The semantics of the Web application model is well-defined by the domain-specific PIM language, and it may be extended when required. Consider e.g. dynamic navigation: if getting the key for the page content from the old page is more complicated than described by the behavioral semantics of the DynPageAnchor class (see Figure 2), a class that overrides the navigate-method may be derived from DynPageAnchor.
2. The software structure of the generated Web application is well-described by the transformation rules; it may be extended or modifed when required. Consider e.g. the **EntityNodeToServlet** transformation rule: it states clearly that an own servlet is generated for each entity node. If that would not be desired, it is quite easy to provide an alternative **EntityNodeToServletA** transformation rule which generates one servlet class for all entity nodes of an Web application, and to provide also an alternative transformation class implementing that modified rule.

This transparency is an important advantage if a tool is used to generate real-world applications for greater companies, since there are company-internal standards to be adhered to. Extension of the behavioral semantics or modification of the transformation rules makes it possible to adhere to different standards.

10. ACKNOWLEDGEMENTS

Our thanks are due to Gustavo Rossi for hosting Oliver in La Plata; the International Bureau of the BMBF, Germany, for the Bilateral Cooperation with Argentina support; and the Ministerium fuer Wissenschaft und Forschung, Baden-Württemberg for a partial support of the project.

11. REFERENCES

[B00] L. Baresi, F. Garzotto, and P. Paolini. ”From Web Sites to Web Applications: New issues for Conceptual Modeling”. In Procs. Workshop on The World Wide Web and Conceptual Modeling, Salt Lake City (USA), October 2000.

[C00] S. Ceri, P. Fraternali, S. Paraboschi: ”Web Modeling Language (WebML): a modeling language for designing Web sites”. Procs 9th. International World Wide Web Conference, Elsevier 2000, pp 137-157

[JCP01] Java Community Process Document JSR26:“UML/EJB Mapping Specification“

[KKCM03] N.Koch, A.Kraus, C.Cachero, S.Melia: “Modeling Web Business Processes with OO-H and UWE“. IWWOST 03, Proceedings 3rd International Workshop on Web-Oriented Software Technology, Oviedo, Spain, 2003

[KWB04] A.Kleppe, J.Warmer, W.Bast: “MDA Explained“, Addison-Wesley Pearson Education, Boston, USA, 2004

[OMG MDA] http://www.omg.org/mda/

[PFP03] O.Pastor, J.Fons, V.Pelechano: “OOWS: A Method to Develop Web Applications from Web-Oriented conceptual Models“. IWWOST 03, Proceedings 3rd International Workshop on Web-Oriented Software Technology, Oviedo, Spain, 2003

[S04] H. A. Schmid: “Model Driven Architecture with OOHDM“. Engineering Advanced WebApplications, Proceedings of Workshops in Connection with the 4th ICWE, Munich, Germany, 2004, Rinton Press, Princeton, USA

[S04a] H. A. Schmid: “Model Driven Architecture with OOHDM“. Proceedings IWWOST 04, Munich, Germany, 2004

[SH04] H. A. Schmid, O.Herfort “A Behavioral Semantics of OOHDM Core Features and of its Business Process Extension“. In Proceedings ICWE 2004, Springer LNCS 3140, Springer, Berlin, 2004

[SR04] H. A. Schmid, G. Rossi “ Modeling and Designing Processes in E-Commerce Applications“. IEEE Internet Computing, January 2004

[SR98] D. Schwabe, G. Rossi: ”An object-oriented approach to web-based application design”. Theory and Practice of Object Systems (TAPOS), Special Issue on the Internet, v. 4#4, pp.207-225, October, 1998

[TJAT04] M. Taguchi, K. Jamroenderarasame, K. Asami and T. Tokuda “Comparison of Two Approaches for Automatic Construction of Web Applications: Annotation Approach and Diagram Approach” In Proceedings ICWE 2004, Springer LNCS 3140, Springer, Berlin, 2004

Model Integration Through Mega Operations

Th. Reiter, E. Kapsammer, W. Retschitzegger

Department of Information Systems
Johannes Kepler University Linz
{reiter | ek | werner}@ifs.uni-linz.ac.at

W. Schwinger

Department of Telecooperation
Johannes Kepler University Linz
wieland.schwinger@jku.ac.at

Abstract

With the advent of the Model Driven Architecture, models are replacing code as the major artifact in software development. A critical success factor for this is the possibility to derive models from each other in terms of transformations. Existing approaches, such as the forthcoming QVT-standard, will provide a proper foundation for transforming models on a fine-grained level. They, however, do not provide appropriate abstraction mechanisms for different integration scenarios, such as integrating models representing cross-cutting concerns or integrating models even from different domains. This paper proposes so called mega operations representing an abstraction mechanism which allow to specify model integration at the meta-level, thus forming the prerequisite to automatically derive a set of directives carrying out the actual integration at the model level. To cope with different integration scenarios, tight as well as loose integration of models is supported on top of a QVT-like language.

1. Introduction

The OMG has initiated the Model Driven Architecture (MDA), a software design methodology emphasizing the construction of models and the subsequent generation of executable code on basis of those models [23]. Thus models are replacing code as the major artifact in software development [4]. One of MDA's main benefit is the abstraction of core business functionality from implementation specific details resulting in platform independent models (PIM) and platform specific models (PSM).

The derivation of a model from another model is carried out through a *transformation* - ideally automated. In order to standardize such a model transformation language, several proposals for a Query/Views/Transformations (QVT)-language have been submitted to the OMG [12]. Model transformations as envisaged by QVT focus on transforming a model m_a at the level M1 conforming to a meta-model M_a at the level M2 into a model m_b conforming to a meta-model M_b, where M_a and M_b may potentially be the same.

QVT definitely represents a major building block technology for basic model transformations in the MDA. It does not provide, however, appropriate abstraction mechanisms for different kinds of *model integration scenarios*, which are highly needed in practice and well-known from other research areas such as *federated information systems* [29], [32], *megaprogramming* [31], *web service composition* and [19] and *aspect-oriented programming* [18]. Such integration scenarios would require a series of basic model transformations which will simply not scale up when manually specified for complex models.

Following, for example, the basic principle of separation of concerns in the modeling realm would avoid the construction of large, monolithic domain models which are difficult to handle and comprehend. At the same time, these models, each of them describing a certain cross-cutting concern of a whole domain (e.g., security aspects and transactional aspects of a web-based tourism information system), need to be *tightly integrated* into one coherent model representing the entire domain, as required for MDA.

Integration is not only needed in the case of models representing aspects of the same domain, but also in case of models covering different domains. For example, it would be highly desirable to integrate web-based reservation systems covering different domains like transportation (e.g, car rental) and accommodation (e.g., hotel booking) in order to allow them to interoperate providing new services for customers. This scenario requires for loose integration, i.e., synchronizing both domain models in certain ways by explicitly representing the model's interrelationships, while providing their autonomy.

Although these two scenarios look quite differently at a first sight, they bear several commonalties in mind. Both call for integrations which can be defined in an *abstract and thus, scalable way*, without burden the modeler with transformation primitives. Integration should not have to be defined repeatedly each time when models should be integrated, but rather be specified once at a meta-level, thus *facilitating reuse of integration knowledge*. Finally, in order to prevent ad-hoc integration of models, the *actual integration at the model level* should be *governed* at the meta level and performed fully *automated*.

To deal with these requirements and based on our experience with various web-based model integration scenarios (cf. [15], [16], [20], [26], [28]) we introduce so-called *mega operations*[1] providing abstraction mechanisms for model integration, thus allowing modelers to develop web systems of several interrelated models. Mega operations offer a set of operators for dealing with model heterogeneity as well as synchronization and provide the possibility to specify integration constraints. Specified at the level of meta-models that are MOF-based [24], a set of integration directives can be automatically derived, carrying out the actual integration at the model level.

To fulfill the needs of the integration scenarios outlined above, integration done by mega operations are required to follow two strategies. Facilitating the integration of aspect models into a coherent domain model, a so called *weaving mega operation* is proposed, achieving tight model integration. Integrating independent domain models requiring, e.g., synchronization and loose coupling to preserve their autonomy is supported by a so called *sewing mega operation*. These mega operations are based on a common architecture using primitive QVT-transformations underneath.

This paper introduces these two mega operations in Section 2 and 3, together with a set of appropriate operators for each of them. In Section 4, an architecture supporting these mega operations is outlined. After a detailed discussion of the benefits of our approach with respect to other closely related approaches in Section 5, we conclude the paper pointing out further research issues.

[1] The term "mega operation" is influenced by the notion of *megaprogramming* - a DARPA research program conducted in the late 1980's and early 1990's (cf. [5], [31]) - and *megamodel* which is defined to be a model, whose elements represent models [4].

2. Weaving

Weaving provides for a tight integration of models. This means that a model m_a conforming to a meta-model M_a and a model m_b conforming to a meta-model M_b, can be woven to produce a model m_{ab} which in turn conforms to a woven meta-model M_{ab} (cf. Fig. 1).

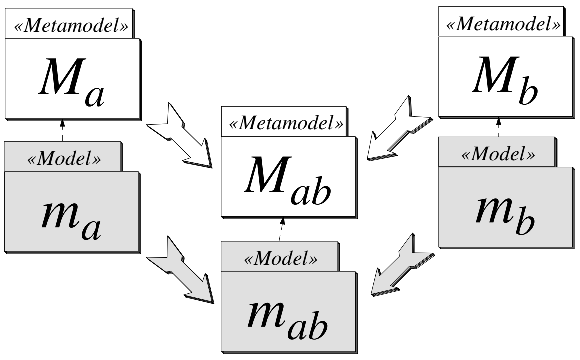


Figure 1. The Weaving Mega Operation

Note that the term weaving is adopted from aspect-oriented programming (AOP) [18], where it describes the process of weaving code representing a cross-cutting-concern into a base program. Transferring this basic idea into the modeling realm, but differently to the concept of weaving introduced by Bézivin et al. [3] (cf. Section 5), our weaving mega operation encompasses two steps:

(1) The weaving of *aspect meta-models* each of them describing a certain cross-cutting concern produces a *woven meta-model* (cf. Fig. 2)

(2) The subsequent weaving of *aspect models*, produces a *woven model* (cf. Fig. 3).

These two steps run automatically, provided that a particular *weaving specification* defines how to execute the weaving mega operation (cf. below).

Instead of only recording the semantic relationships between model elements, creating a woven model is necessary in case further processing or code generation mechanisms require so.

Furthermore, an advantage of defining a woven meta-model prior to the weaving of models is the ability to perform *conformance checks* with respect to the woven meta-model. Furthermore, having a meta-model for any given model is beneficial for *defining transformations* in the sense of MDA.

In the following we use a simplistic, though still sufficient running example to illustrate our basic ideas, stemming from the well-known domain of petri nets [25].

The meta-model M_{Petri} describes some basic structural aspects of a petri net consisting of places and transitions connected by arcs, whereas the meta-model M_{Mark} represents the aspect of markings, constituting places and marks (cf. Fig. 2).

The first step of the weaving mega operation - meta-model weaving - results in a woven meta-model $M_{PetriMark}$, representing a petri net with certain markings.

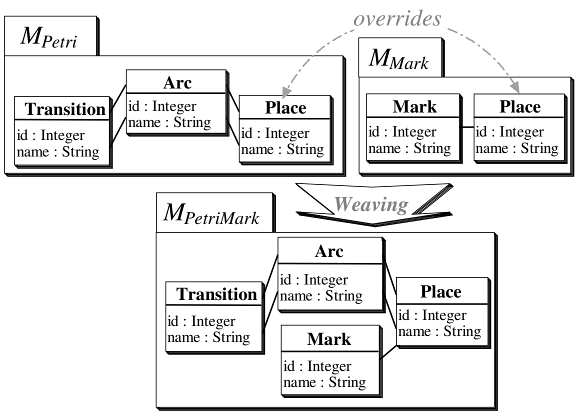


Figure 2. Meta-Model Weaving

The second step deals with the weaving of models (cf. Fig. 3). Governed by the new woven meta-model $M_{PetriMark}$, the woven model $m_{PetriMark}$ is produced, which consists of model elements from both source models m_{Petri} and m_{Mark}.

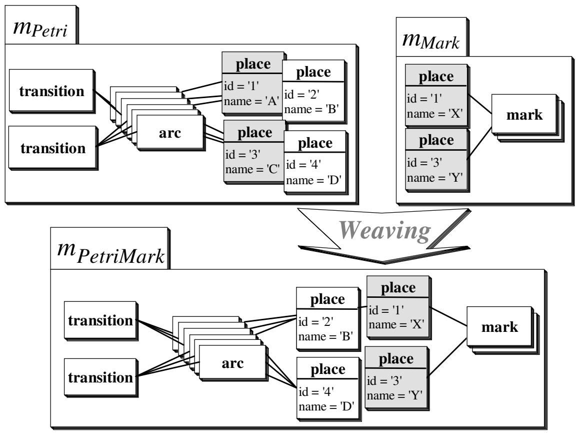


Figure 3. Model Weaving

Applying weaving mega operations as described above may yield the following benefits:

- Weaving allows the composition of domain meta-models, and thus enables the *re-use* of previously existing *domain knowledge*.
- Weaving allows several *teams* to model independently and weave their models as needed.
- Weaving allows the *evolution of a domain*, as new concerns can be woven in the form of aspect models, and thus supports incremental development of models.
- Weaving supports *scalability*, as there are no monolithic meta-models and models impairing comprehensibility.
- Weaving allows *libraries of models* to be built up for later re-use.
- Weaving makes modeling an activity of *assembling pre-existing "components"*.

The following subsections discuss the pre-requisites for putting the weaving mega operation into use, in terms of the weaving specification, comprising *weaving operators* and *model integration constraints*.

2.1. Weaving Operators

This subsection discusses several operators which are essential for defining weaving operations. Such operators need to address the reconciliation of overlapping concepts and allow basic model re-organization (cf., e.g., [30]). The set of operators comprises `overrides`, `references`, `prune`, and `rename` and does not claim to be complete.

Overrides. In case that two meta-models overlap in the form of elements representing the same concept, a weaving specification has to denote how to reconcile these model elements. Adopted from [30], but in contrast to them applied at a meta-level (cf. Section 5), we make use of an `overrides` operator which specifies that one meta-model element (qualified by "::") and its properties at the left hand side, take precedence over another meta-model element at the right-hand side.

With respect to the example shown in Fig. 3, the `overrides` operator expresses that meta-model element `Place` of the meta-model M_{Mark} replaces its pendant within the meta-model M_{Petri}.

M_{Mark}`::Place` **`overrides`** M_{Petri}`::Place;`

References and Inherits. If two meta-models do not conceptually overlap, a `references` operator denotes to connect meta-model elements via a new association. This operator also allows to specify the multiplicities of the association established between two meta-model elements. In our example, this expresses the fact that a place is able to hold an arbitrary number of marks.

M_{Mark}`::Mark` **`references(*,1)`** M_{Petri}`::Place;`

Similar to the `references` operator, but naturally not allowing for specifying multiplicities, we use an `inherits` operator to connect model elements via inheritance relationships.

Prune and Rename. As the previously introduced weaving operators "enrich" meta-models with elements, only, they cannot deal with the renaming or the deletion of possibly obsolete model elements, as portrayed in [30].

Therefore, a `prune` operator serves to rid all unnecessary meta-model elements in a meta-model. The example below shows the pruning of the obsolete `Mark` element.

M_{Mark}`::Mark` **`prune;`**

Renaming of meta-model elements can be done by applying a `rename` operator. As opposed to the previously mentioned weaving operators, `prune` and `rename` are unary in terms of meta-model elements. The example below shows the name change of the `Place` element.

M_{Petri}`::Place` **`rename`**`('State');`

2.2. Model Integration Constraints

Besides weaving operators, a weaving specification shall contain certain constraints, called *model integration constraints (MIC).* MICs are used to restrict the application of a weaving operator when integrating at the model level, thus forming some kind of precondition.

A MIC can be annotated for each application of a weaving operator. This means that the application of the operator at the model level is only carried out for those model elements, meeting the corresponding constraint. Thus, the MIC acts like a "filter", sorting out all invalid weaving operations and is indicated after the keyword "`MIC:`".

As shown in Fig. 3, only the `Place` model elements with `id='1'` and `id='3'` from the model m_{Mark} override the `Place` model elements in the m_{Petri} model with the matching values. For this, the previous example of the `overrides` operator is extended by the following MIC-specification:

`MIC:` M_{Mark}`::Place.id` **`==`** M_{Petri}`::Place.id;`
M_{Mark}`::Place` **`overrides`** M_{Petri}`::Place;`

2.3. Performing Meta-Model Weaving and Subsequent Model Weaving

A weaving operation can be reduced to a set of QVT transformations on the meta-model as well as on the model level. For the generation of the woven meta-model QVT transformations, derivable from the weaving specification, can be specified on the transformation's meta-level (M3) and applied to meta-models. In this way QVT populates the woven meta-model with model elements stemming from the meta-models to be woven. Likewise the subsequent weaving of models is specified in QVT on the transformation's meta-level (this time on M2) and applied to the model level. The actual QVT transformations to apply depend on the weaving operators involved and their attached MICs resulting in a certain transformation behavior.

According to the latest QVT 2.0 proposal [26] and to the best of our knowledge, Fig. 4 depicts an example transformation which could be derived from a weaving as shown in Fig. 2 and Fig. 3. Assuming that a transformation executed beforehand has populated the $m_{PetriMark}$ model with the model elements from m_{Petri}, the execution of the transformation below in the direction of the $m_{PetriMark}$ model, would enforce the overriding of `place` model elements and the creation of the according `mark` model elements.

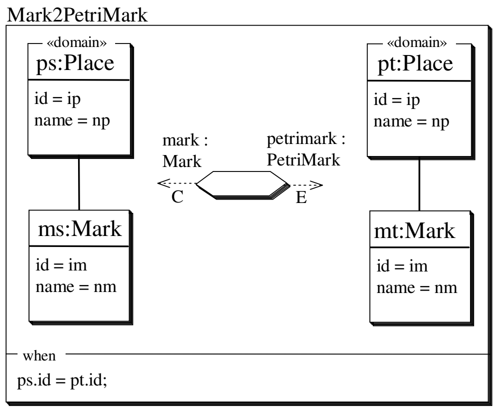


Figure 4. Example QVT Transformation

3. Sewing

As already mentioned, the weaving mega operation provides for a tight integration of models, by composing a coherent domain model from aspect models. Besides that, a loose coupling of models is required to integrate independent models pertaining to different domains and to keep them autonomous at the same time.

Therefore, apart from weavings, we see the necessity to introduce another mega operation called *sewing*. Sewing seems an appropriate analogy, as loose coupling can be seen as a form of stitching the involved models together, and thereby connecting without modifying them.

Analogous to weaving, a model m_a conforming to a meta-model M_a and a model m_b conforming to a meta-model M_b can be sewn to produce a set of *mediators* [32] realizing the integration, by "supervising" the sewn model elements (cf. Fig. 5).

Similar to a weaving specification, a *sewing specification* consists of operators annotated with MICs, and thus defines how to execute a sewing mega operation (cf. below). Specifying sewings on meta-models prior to the sewing of models, is deemed necessary to enable a meta-modeler to clearly define which model elements are valid to be sewn, and to henceforth rule out the ad-hoc creation of possibly ill-defined sewings.

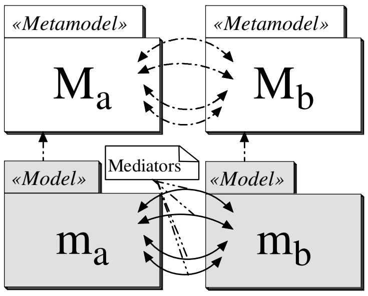


Figure 5. The Sewing Mega Operation

Continuing our running petri net example, let's imagine that we would like to have a graphical user interface (GUI) for a petri net simulation (cf. Fig. 6). The mega operator sewing could establish (similar to the model-view-controller paradigm) a loose coupling between the `name` attribute of the `Place` meta-model element belonging to the petri net model, and the `title` attribute of a `TextField` meta-model element belonging to the GUI model.

A tight coupling in the form of weaving the GUI model and the petri net model would not be adequate in this situation, as different domains are involved and weaving would simply entangle the different domain concepts.

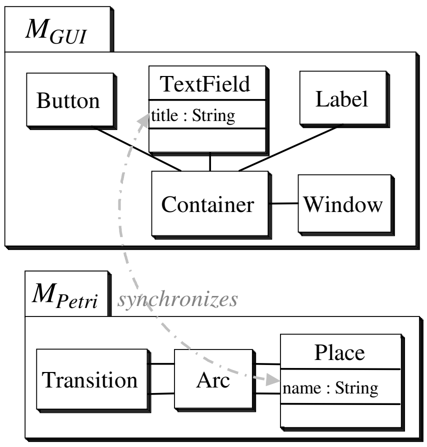


Figure 6. Meta-Model Sewing

As shown in Fig. 7, the application of a sewing mega operation at the meta-level results in the establishment of mediators between model elements, guided by MICs.

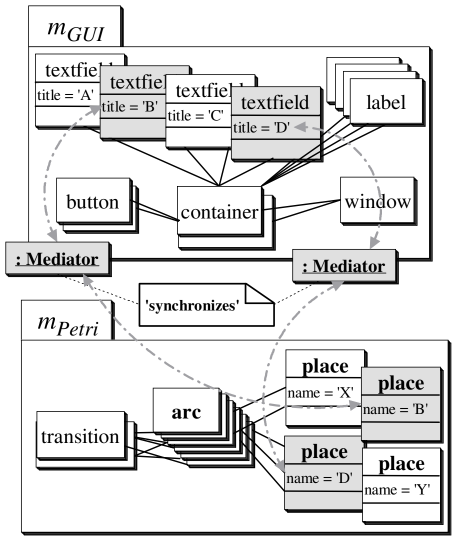


Figure 7. Model Sewing

Applying sewing mega operations as described above may yield the following benefits:

- Sewing integrates models, but still allows them to exist independently without affecting their structure and thus keeping their *autonomy*.
- Sewing serves to keep models *synchronized*.

- Sewing integrates models pertaining to different domains *without entangling* their concepts.

The following subsections discuss sewing operators together with their corresponding MICs and realization in terms of mediators.

3.1. Sewing Operators

The particular behavior of mediators depends on the specific operators in the sewing specification. The following subsection introduces such operators, namely `synchronizes` and `depends`, which are useful for the sewing of models. Such operators enforce to supervise the sewn model elements by observing their states and appropriately propagate changes.

Synchronizes. In case that, for instance, attributes of two model elements should be kept synchronized, a `synchronizes` operator can be used to denote that fact. With respect to the previous example (cf. Fig. 6 and Fig. 7) the `synchronizes` operator together with a MIC is employed as follows:

```
MIC:  MGui::TextField.title ==
      MPetri::Place.name;
MGui::TextField.title synchronizes
MPetri::Place.name;
```

According to the MIC, synchronizations are established between `TextField` model elements and `Place` model elements, only, if having equal values for their `title` and `name` attributes, respectively. Applied on the model level (cf. Fig. 7), changing the value of the `title` attribute would lead to a change in the value of the `name` attribute.

Depends. The `depends` operator is used to denote that the existence of one model element depends on the existence of another. If two teams are working on two separated, though related models, it can be useful to establish such correspondences between the related model elements. Thus, if one team decides to delete a model element, the related model element should immediately be deleted as to avoid inconsistencies among the teams' models. The example below shows a sewing specification for the meta-model element `TextField` depending on the meta-model element `Place`.

```
MIC:  MGui::TextField.title ==
      MPetri::Place.name;
MGui:TextField depends MPetri::Place;
```

It has to be noted, that sewing is focused on integrating existing models, not on creating them anew from another model, as QVT allows. Sewings therefore have a narrower domain and aim at simplifying certain integration tasks that would probably be more cumbersome to express using QVT alone.

3.2. Sewing realized by Mediators

The application of a sewing operator does not result in a newly produced, integrated model per se, as it is the case with weaving, where heterogeneities in the form of conceptual overlap can be eliminated through the establishment of woven meta-models and models. On the contrary, sewing has to handle, or better to say, transparently resolve existing overlap throughout sewn models. Thus, the outputs of the sewing mega-operation are mediating entities producing the desired integration behavior.

On the model level, mediators can manifest as QVT transformations propagating attribute changes or creating and deleting model elements accordingly.

Operators other than the two previously introduced `depends` and `synchronizes`, which would for instance allow model elements to be transparently connected via associations and generalizations across model boundaries, could be realized using the *Java Metadata Interface* (JMI) [10] and the *Eclipse Modeling Framework* (EMF) [9]. They provide an infrastructure for the generation of programming interfaces to instantiate and manipulate models as Java run-time objects. Such programs resulting from sewn models have to be adapted in a way, as to reflect the semantics and the mediating behavior of the specific operator. In case that it is not possible to influence the model code generation, an elegant solution would be to utilize an aspect-oriented approach and weave the necessary code fragments for the mediator pattern into the model code. The aspect code necessary would be derived from the sewing specifications.

However, when finally code is to be produced from models, the mediating behavior also has to be realized on the system level, specific to a certain platform. Sewings can of course manifest as models themselves, which describe the respective semantics and the integration behavior imposed on models. The generation of platform specific "bridge" code facilitating a loose coupling on the system level is thus rendered a common task like any other model driven development, as integration of heterogeneities is taken care of on the model level. The mediation on the system level could for instance be carried out by a web service, connected to different systems generated from sewn models.

4. Architecture

This section proposes a first sketch of an architecture for the implementation of a mega-operation toolkit and briefly discusses relevant technologies. Fig. 8 shows a GUI component as means for handling a Mega-Operations Controller, which orchestrates the toolkit's components as required. A MOF repository serves as basis for storing meta-models and models. To access and manipulate them programmatically, programming interfaces like JMI or EMF Java mappings can be used. Although EMF and JMI provide the necessary infrastructure for manipulating models, they are not capable of model transformations in the sense of QVT. Hence, a QVT-like model transformation tool such as Marius[2], which has been developed in the course of a former cooperation between the University of Linz and the University of South Australia, is employed for model transformation. Enforcing constraints on models can be accomplished by an Object Constraint Language (OCL) checker like [1].

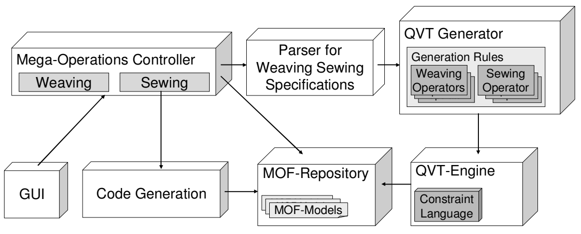


Figure 8. Architecture for Mega Operations

As already mentioned, the weaving mega operation can essentially be expressed as a series of QVT-like transformations, as can the sewing mega-operation concerning the model level. Thus, weaving and sewing specifications are parsed and input into a QVT generator, which can be seen as the toolkits core component, "compiling" weavings and sewings into QVT-code. The resulting code is in turn executed by a QVT-engine upon models stored in the repository to achieve the integration of models. A code generation component serves to create bridge code realizing the sewing mega-operation's loose coupling on the system level. The therefore necessary "glue" code can be incorporated into the code derived from models either directly through customisation of the generated code or through an aspect weaver like AspectJ [18].

[2] The name *Marius* stems from Gaius Marius, a Roman consul and general, best known for initiating a series of reforms 107 B.C., completely restructuring the organization equipment and tactics of the Roman army.

5. Related Work

This section gives an overview on other approaches most relevant with respect to our idea of mega operations. For this, the main focus of each approach is summarized briefly, followed by clearly pointing out similarities and differences to our own approach.

Table 1 summarizes the results by giving an overview on operators supported as well as whether the approach deals with arbitrary MOF-based models either on the M1 or the M2 level and if the specification on M2 is used for model integration on M1.

Table 1. Comparison of Related Approaches

							Weaving Operators				***Sewing*** Operators	
		MOF-based	M2 meta-level	M1 model-level	Meta-level based integration	MICs	overrides	reference	inherits	prunes / renames	synchronizes	depends
Approach	AMMA	✓	✓	✓	✓	▨	▨	▨	▨	▨	✓	▨
	Rondo	▨	✓	✓	▨	▨	✓	▨	▨	✓	▨	▨
	Model Composition Semantics	▨	▨	✓	▨	▨	✓	✓	▨	▨	▨	▨
	Model Composition Directives	▨	▨	✓	▨	▨	✓	✓	▨	✓	▨	▨
	GME	▨	▨	✓	~	▨	✓	✓	✓	▨	▨	▨
	C-SAW	▨	▨	✓	✓	▨	▨	▨	▨	▨	▨	▨
	Domain Composition Approach	▨	▨	~	▨	▨	▨	▨	▨	▨	✓	▨

Legend:		
✓	...	explicitly supported
▨	...	not explicitly supported
~	...	not applicable

AMMA. Bézivin et al. [3], [4], [21] are developing the *Atlas Model Weaver (AMW)* as part of the *AMMA model engineering platform*, which is soon to be released under the *Eclipse GMT project* [8]. The AMW aims at supporting modelers to establish semantic links between elements of different models or meta-models, which can serve as input for further tools. Model weaving in the sense of Bézivin et al. seems to be a manual operation specifying links between elements of different models or meta-models. The set of links produced by such a weaving operation is represented by a *weaving model*. A weaving model appears to be similar to a *weaving specification* in our approach, which specifies operators linking meta-model elements.

Our approach, however, extends the notion of weaving from an activity that establishes semantic links between meta-models, to a mechanism that actually interprets operators specified between meta-model elements and carries out operations accordingly. These operations involve the automatic generation of a new woven metamodel, which is an integration of the original metamodels. Furthermore, we provide a mechanism to automatically integrate models into a new woven model conforming to the new woven meta-model. In our understanding, weaving is treated as a distinct abstraction mechanism for the integration of both, models *and* meta-models.

Rondo. Within the *Generic Model Management* initiative, Bernstein et al., [2], [22] work on merging meta-data in the form of relational schemata and XML schemata. *Rondo* is an implementation thereof, providing model management operators that enable modelers to deal with models rather than model elements. Similar to our weaving and sewing operators, these operators include a *match* operator, which automatically establishes semantic correspondences between similar schema elements and a *merge* operator allowing to combine different model elements.

In contrast to them, however, we explicitly focus on MOF-models in the sense of MDA, keeping a later code-generation step following model integration in mind. Furthermore, in our approach, a meta-modeler is able to specify the integration of models and meta-models on a meta-level, instead of providing generic model management operators to manipulate models.

Model Composition Semantics. Clark [7] introduces a *composition mechanism* for UML class diagrams. This approach deals with the composition of models representing different separated concerns. Overlapping concepts are identified in these models and thus merged as specified by a composition relationship, following so-called *merge* and *override* strategies. Merge integration for example applies when equivalent classes appear in multiple design models, and conflicts need to be reconciled among these. Override integration can be used to substitute obsolete parts of a design with new modeling constructs. Based on these basic integration behaviours, *composition patterns* [6] are introduced as an extension to UML templates.

This approach, however, focuses on UML models, only, and does not provide for deletion of obsolete model elements after a weaving is performed, as required for our approach.

Model Composition Directives. Based on [7], Straw et al. [30] propose so called *composition directives* for composing UML class diagrams. These basically include name rewriting, adding and deleting of model elements, change of references, and control of execution order. Inspired by aspect-oriented programming concepts, so-called primary models are composed with aspect models, which represent a cross-cutting-concern to be interwoven.

Although composition directives are comparable to our weaving operators, their primary focus seems to be on model weaving but not on meta-model weaving. We believe that our mega operations could in turn be transformed into composition directives at the model level. Since we avoid an ad-hoc integration of models, with our mega operations, licit integrated models can be generated, only.

GME. The *Generic Modeling Environment (GME)* proposed by Karsai et al. [17] is a modeling and meta-modeling toolkit based on UML notation and a GME-specific meta-metamodel. GME allows for the composition of meta-models similar to our approach. The composition mechanisms comprise an *equivalence operator* creating a union of two model elements, similar to the *merge* semantics in [7] and two different inheritance operators, realizing implementation inheritance and interface inheritance.

One major difference to our approach is that GME is not based on the MOF standard. Furthermore, we believe that our approach goes beyond the functionalities for meta-model composition in the GME by introducing model integration constraints, allowing even fine-grained integration of models.

C-SAW. C-SAW, developed as a plug-in for the above-mentioned GME by Gray et al. [13], [14], is a so-called *cross-cutting-concern weaver*. Aspects are specified using the *Embedded Constraint Language (ECL)*, which is a superset of OCL, additionally providing imperative constructs for model manipulation.

The transformation capabilities of ECL are, however, limited to models of the same meta-model and it lacks support for abstract integration mechanisms as supported by our approach.

Domain Composition Approach. Estublier et al. [9] propose a *UML profile* to allow the composition of separately designed domain models, as required when facing the federation of immutable components off the shelf. UML associations and association classes are specialized by dedicated stereotypes to express feature correspondence and concept overlapping.

In principle, this approach is similar to our sewing mega operation. In contrary to this UML-based approach, our sewing mega operation is applicable to arbitrary MOF models. In addition, it seems that their focus lies not on tight integration of models, as done by our approach.

6. Conclusion and Outlook

This paper proposes mega-operations for model integration and shows the benefits that can be gained thereof. Apart from QVT-like mappings, which can be seen as the base requirement to the MDA approach, the introduced mega-operations weaving and sewing provide abstraction mechanisms to cope with complex modeling scenarios, allowing for a tight and loose coupling, respectively. Thus, enhanced scalability and further re-use capabilities of a model-driven approach are gained.

Future work will especially concentrate on clearly defining the integration behavior enforced by weaving and sewing operators.

Therefore, on the one hand the proposed operators have to be specified in detail, and on the other hand, further operators have to be conceived. Detailing would, e.g., include clarifying different reconciliation behaviors of the `overrides` operator, propagation behavior of the `synchronizes` operator, as well as detecting and resolving conflicts arising from the application of the mega-operations.

With respect to both, weaving and sewing, a clear syntax and means for representing the mega-operations as MOF models have to be developed.

Furthermore, an important issue to resolve will be to find ways to derive platform specific implementations for mediators.

Finally, a prototypical implementation for mega-operations shall be developed. Experiments with this prototype should yield valuable insight into the applicability of mega-operations as devised in this paper.

References

[1] D. Akehurst, O. Patrascoiu, "OCL 2.0-Implementing the Standard for Multiple Metamodels", *Proc.s of the UML'03 workshop*, Electronic Notes in Theoretical Computer Science, November 2003.

[2] P. A. Bernstein, "Applying Model Management to Classical Meta Data Problems" *Proc. of the Conf. on Innovative Database Research (CDIR03),* Asilomar, California, Jan. 2003, pp. 209-220.

[3] J. Bézivin, F. Jouault, P. Valduriez, "First Experiments with a ModelWeaver", *OOPSLA & GPCE Workshop*, Vancouver, October 2004.

[4] J. Bézivin., F. Jouault, P. Valduriez, "On the Need for Megamodels", *OOPSLA & GPCE Workshop*, Vancouver, October 2004.

[5] B. Boehm, B. Scherlis, "Megaprogramming", *Proceedings of the DARPA Software Technology, Conference,* 1992.

[6] Clarke, S., Walker, R.J. "Composition Patterns: An Approach to Designing Reusable Aspects", *Proceedings of International Conference on Software Engineering (ICSE)*, Toronto, Canada, 2001.

[7] S. Clarke. "Extending standard UML with model composition semantics", *Science of Computer Programming, Elsevier Science*, Volume 44, Issue 1, July 2002, pp. 71-100.

[8] Eclipse Foundation, *Generative Model Transformer (GMT),* http://www.eclipse.org/gmt/, 2005.

[9] Eclipse Foundation, *Eclipse Modeling Framework (EMF)*, http://www.eclipse.org/emf, 2005

[10] Java Community Process, *Java Metadata Interfaces (JMI)*, 2002, http://java.sun.com/products/jmi/

[11] J. Estublier, A. D. Ionita, G. Vega, "A Domain Composition Approach", *Proc. of the International Workshop on Applications of UML/MDA to Software Systems (UMSS),* LasVegas, USA, June 2005.

[12] T. Gardner, C. Griffin, J. Koehler, R. Hauser, "A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard", *Object Management Group (OMG)*, ad/2003-08-02.

[13] J. Gray, T. Bapty, S. Neema, A. Gokhale, "Generating Aspect-Code from Models", *OOPSLA Workshop on Generative Techniques for Model-Driven Architecture*, Seattle, WA, November 2002.

[14] J. Gray, T. Bapty, S. Neema, D. C. Schmidt, A. Gokhale, B. Natarajan, "An Approach for Supporting Aspect-Oriented Domain Modeling", *Generative Programming and Component Engineering (GPCE)*, Springer-Verlag LNCS 2830, Erfurt, Germany, September, 2003, pp. 151-168.

[15] G. Kappel, E. Kapsammer, W. Retschitzegger "Integrating XML and Relational Database Systems", *World Wide Web Journal (WWWJ), Kluwer Academic Publishers*, Vol. 7(4), December 2004, pp. 343-384

[16] E. Kapsammer, W. Schwinger, W. Retschitzegger, "Bridging Relational Databases to Context-Aware Services", *Proc. Of the CAiSE Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS)*, Springer LNCS, Porto, Portugal, June 2005.

[17] G. Karsai, M. Maroti, A. Ledeczi, J. Gray, and J. Sztipanovits, "Composition and Cloning in Modeling and Meta-Modeling Languages", *IEEE Transactions on Control System Technology*, special issue on Computer Automated Multi-Paradigm Modeling, March 2004, pp. 263-278.

[18] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, L. Videira, J.-M. Loingtier, J. Irwin, "Aspect-Oriented Programming", *Proc. Of the European Conference on Object-Oriented Programming (ECOOP), Springer LNCS 1241*, Finland, 1997.

[19] J. Koehler and B. Srivastava, „Web service composition: Current solutions and open problems.", Proc. of the *ICAPS, Workshop on Planning for Web Services*, Trento, Italy, June 2003.

[20] G. Kramler, E. Kapsammer, G. Kappel, W. Retschitzegger, "Towards Using UML 2 for Modelling Web Service Collaboration Protocols", *Proc. of the First Int. Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA)*, Geneva, Switzerland, February 2005.

[21] D. Lopes, S. Hammoudi, J. Bézivin, F. Jouault, "Mapping Specification in MDA: From Theory to Practice", *First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA)*, Geneva, Switzerland, February 2005.

[22] S. Melnik, "Generic Model Management: Concepts and Algorithms", *Springer LNCS 2967*, 2004.

[23] Object Management Group, "MDA Guide", Version 1.0.1, June 2003 [http://www.omg.org/docs/omg/03-06-01.pdf]

[24] Object Management Group (OMG), "MOF 2.0 IDL Specification", July 2004, [http://www.omg.org/cgi-bin/apps/doc?ptc/04-07-01.pdf]

[25] C. A, Petri, "Fundamentals of a Theory of Asynchronous Information Flow", *Proc. of IFIP Congress 62, Amsterdam: North Holland Publ. Comp.,* 1963, pp. 386-390.

[26] QVT-Merge Group, "Revised Submission for MOF 2.0 Query/View/Transformation RFP(ad/2002-04-10)", *Version 2.0, ad/2005-03-02,* March 2005

[27] Th. Reiter, "Transformation of Web Service Specification Languages into UML Activity Diagrams", *Master Thesis, Dept. of Information Systems, Johannes Kepler University Linz,* March 2005.

[28] M. Schrefl, M. Bernauer, E. Kapsammer, B. Pröll, W. Retschitzegger, T. Thalhammer, "Self-Maintaining Web Pages", *Information Systems (IS), International Journal*, Vol. 28/8, Elsevier Science Ltd., 2003, pp. 1005-1036

[29] A.P. Shet, J.A. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases", *ACM Computing Surveys*, Vol. 22, No 3., Sep. 1990, pp. 182-236.

[30] G. Straw, G. Georg, E. Song, S. Ghosh, R. France, and J. M. Bieman, "Model Composition Directives", *7th UML Conference*, Lisbon, Portugal, October, 2004.

[31] G. Wiederhold, P. Wegner, S. Ceri. "Toward Megaprogramming", *Communications of the ACM*, November 1992.

[32] G. Wiederhold, "Mediators in the Architecture of Future Information Systems", *IEEE Computers*, Vol. 25, No. 3, March 1992, pp. 38-49.

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 he 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.

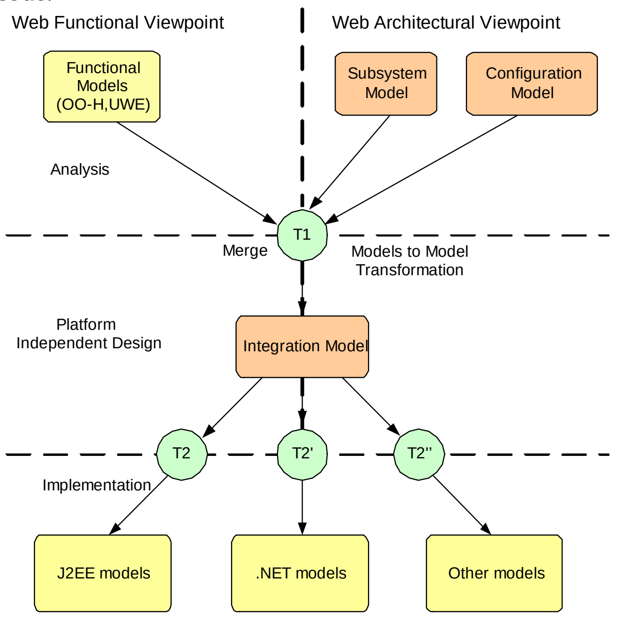


Fig. 1. The WebSA Development Process

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.

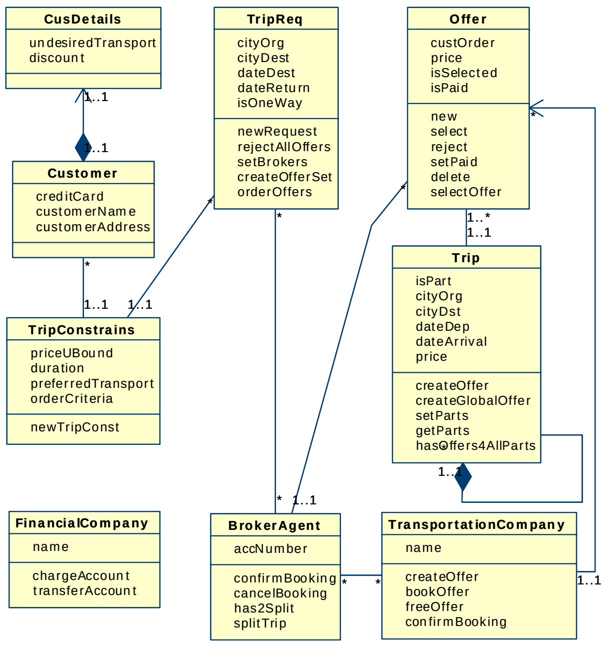


Fig. 2. Conceptual Model of the Travel Agency.

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.

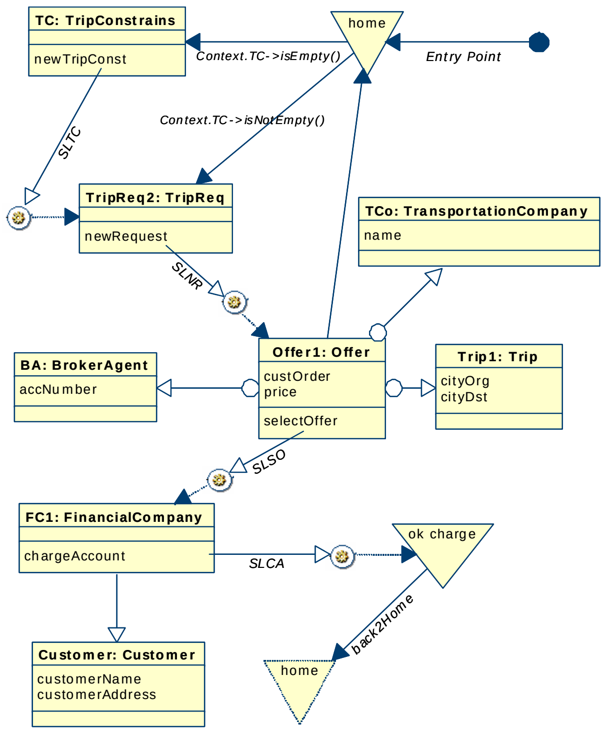


Fig. 3. Navigation Model of the Travel Agency.

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 funcional 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).

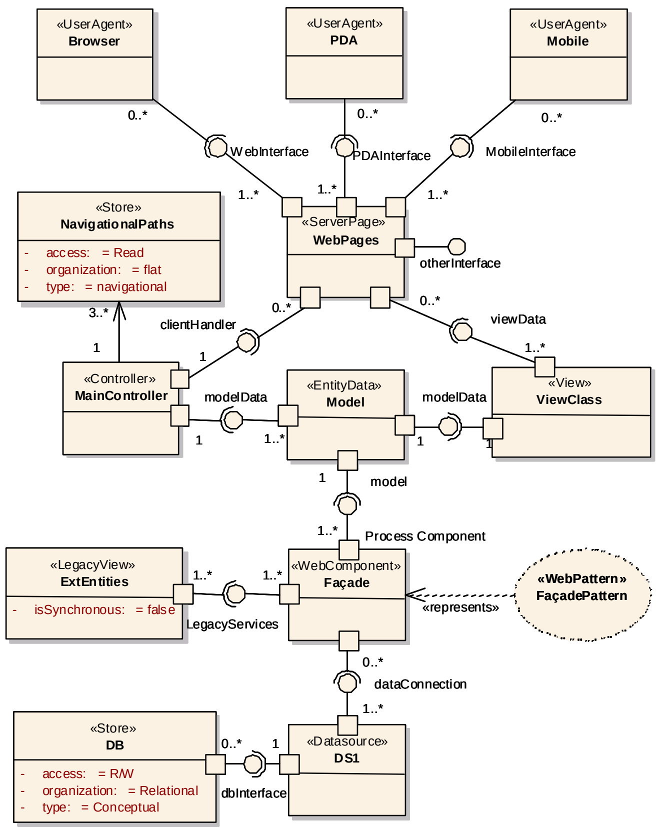


Fig. 4. Configuration Model of Travel Agency

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.

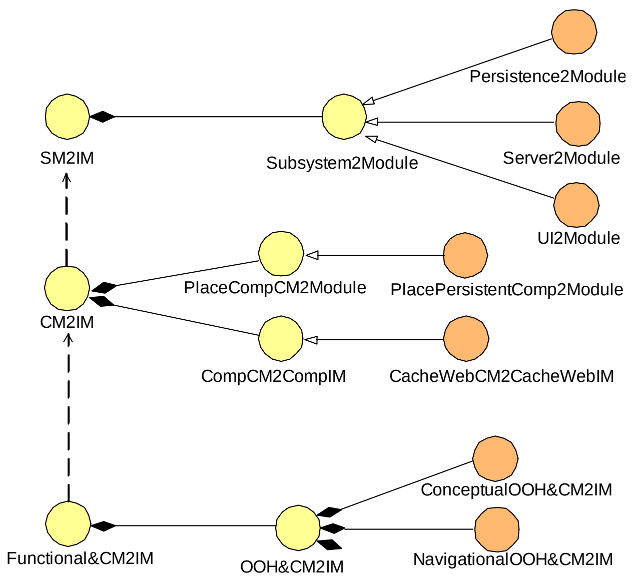


Fig. 5. T1 Transformation Map

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*.

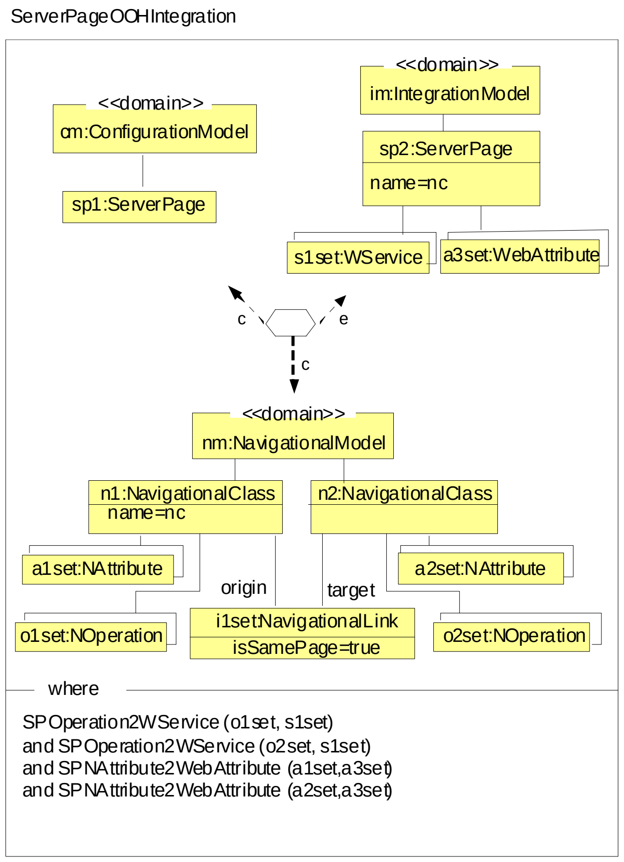


Fig. 6. Example of T1: NavigationalOOH&CMToIM

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).

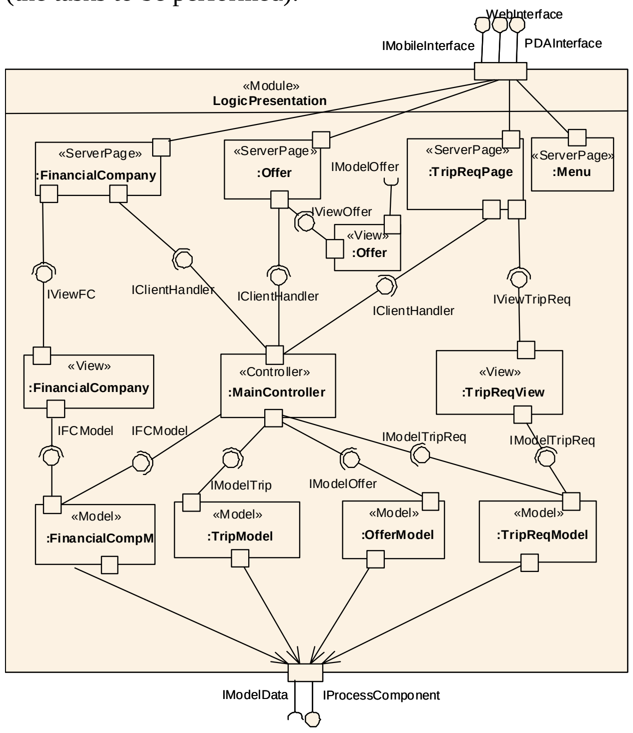


Fig. 7. Integration Model of Logic Presentation Module of Travel Agency

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.

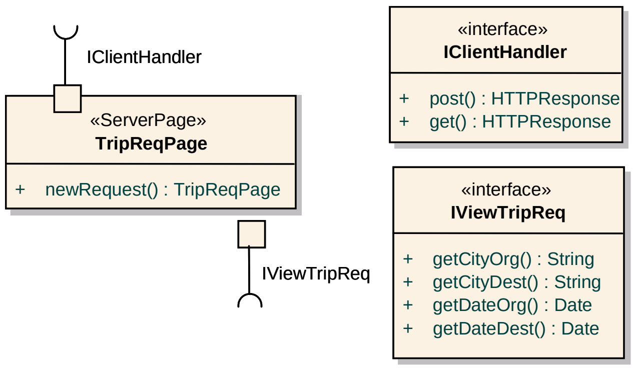


Fig. 8. Definition of the TripReq ServerPage and its interfaces.

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| WebService2Form (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* OCL 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 nº1, 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 submision: 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

Web Engineering does profit from a Functional Approach

Torsten Gipp and Jürgen Ebert
University of Koblenz-Landau
e-mail: (tgi|ebert)@uni-koblenz.de

Abstract

Founding web site development on models *is the state of the art. This paper aims at showing that* functional specifications *are a powerful means of modelling specific aspects of a web application and that it may be employed to gain overall coherency and an* integrated *set of models. The composition of pages from fragments, the incorporation of content based on queries, the definition of dynamics of the web application, as well as the transformation of pages into appropriate presentation level languages: all these aspects are different models, and they are specified and integrated using a functional language. All models constitute a coherent view on the web site as a whole.*

At the same time, using this functional approach proffers openness and extensibility to incorporate well-established tools and technologies.

1 Introduction

Initial creation and maintenance of websites are two sides of the same coin. A common approach for generating websites and for keeping dynamic websites up-to-date involves separating the different concerns of web site description into different yet integrated documents.

We assume a *model-based view* on web sites: a web site is described by a set of models, each emphasising a particular point of view and at the same time being part of one coherent and consistent 'big picture' of the web site as a whole. In other words, it does not suffice to regard the different models in isolation. Instead, it must be guaranteed that the models are *integrated*. It is important to have a coherent and precise description of all relevant web engineering aspects and of the artefacts (models) that are relied upon. Thanks to the clearly defined integration of the models, which also clarifies that they are *separate concerns*, each model can be tackled individually.

In addition to the models, the *process* of the creation and evolution of a web site has to be considered as well. This is an orthogonal aspect that applies to, and should reflect in, every model.

A prominent focus in our work is *maintenance*. One inherent characteristic of a web site is that it changes and evolves over time, and indeed rapidly so. Therefore, every model created during the development of a web site is potentially changed or even rewritten at any point of time.

One of the core contributions of this paper is to use a *functional programming* approach and a functional language to describe and specify the web site's single pages. We employ the functional programming language Haskell [1] as an example language. The functional specifications are executable and they are used by the run-time system to actually drive the web site. The approach is general enough to provide that the realisation is not constrained to a particular functional language or a particular run-time system. The focus clearly lies on integration, be it on the tools level or on the modelling level.

Relying on functional specifications for the definition of web sites is quite natural insofar as retrieving a web page via HTTP actually *is* a function call. Parameters may be passed in, and the final page is delivered as the result. This result is declaratively defined by a function, which in turn can include calls to other functions so to compose a page from smaller parts. One gains a true amount of coherence since everything in this specification is a function.

At the same time, this approach fosters modularity. The granularity of the fragmentation can be chosen with all flexibility of the functional language, and it can also differ on a page-to-page basis. This allows for doing template-based page generation, which we exploit by proposing an example template mechanism to build pages with a consistent layout. This mechanism can easily be substituted or extended if more complexity is needed.

A functional specification is a formal description. At the same time, it is an implementation that can be executed. This advantage can be exploited for prototyping and even for simulation of web sites. The specification represents a high-level view on the page definitions, but also lends itself to 'drilling down' due to its inherent rigourousity.

Common maintenance activities, e.g., established configuration management disciplines, can be applied almost naturally to the functional specifications.

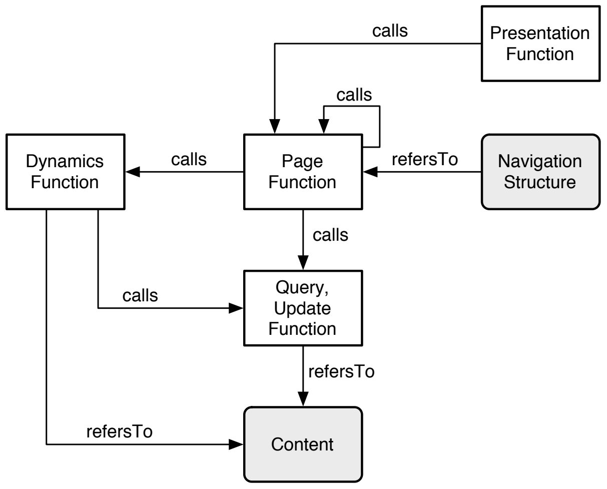


Figure 1. Artefact integration

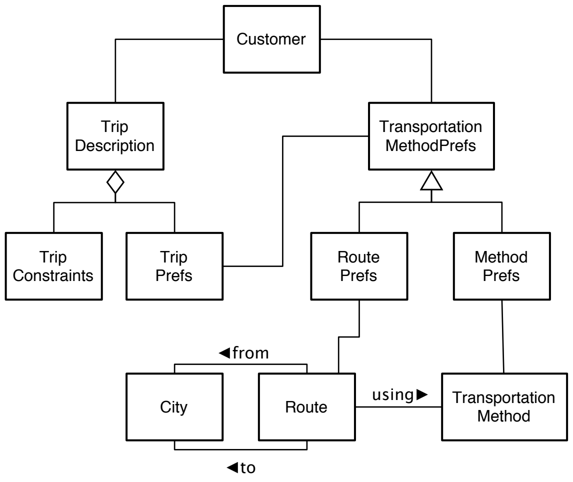


Figure 2. Content model for the TAS example

As suggested in the call for papers, the approach will be presented by using the travel agency system (TAS) as an example. Cf. [2] for a description of this system.

The remainder of the paper is structured as follows. The following section 2, backed up by the TAS example, introduces the models that yield the 'big picture,' giving a detailed explanation of the application of the functional specifications. Section 3 emphasises the model integration, section 4 gives an overview of related work, and section 5 concludes by summarising the core items of this text.

2 Models for the TAS example

According to the separation of concerns mentioned in the introduction, we provide a set of core models that, taken together, capture the application domain and its projection into a set of web pages. We consider: the content, the navigation structure (site map), the pages (navigation objects), queries and updates, the presentation, and the dynamics. The models are *integrated*, and sections 2.1 to 2.6 each describe one model and its integration with the others. Figure 1 visualises the interdependency of the models. After introducing the single models, section 3 will re-focus on their integration.

2.1 Content

The content model captures the concepts of the application domain and their relationships. A UML class diagram is used to write down the model (see fig. 2). For demonstration purposes, it is sufficient to consider a subset of the classes only. The diagram in figure 2 only contains classes that deal with the description of a trip and the storage of a customer's preferences in terms of transportation methods and routes.

The UML diagram describes the structure of the repository that keeps all contents. A query facility (see section 2.4) allows the extraction of information for the inclusion in the web pages.

2.2 Navigation Structure

The *navigation structure* determines the relation of the single pages with respect to the hyperlinks between them. It is modelled with a visual language that provides special features, like distinct page types and authorisation-dependent navigation, thus defining the *site map*. This language has been successfully applied in some of our web engineering projects.

Figure 3 visualises the site map for the TAS. The *primary* navigation structure, given by the solid arrows, defines a tree of page nodes. This tree assigns a unique path to every page. Also, this tree structure is easy to communicate to a web site visitor, who can create a mental image of the site map fairly quickly, which in turn is a very important ergonomic feature.

The *secondary* navigation structure is visualised by dashed arrows. They represent arbitrary links between pages, without heeding the tree structure.

There are four different types of pages, visually differentiated by four different page icons (cf. the legend in fig. 3). A lightning bolt marks a page as being *dynamic*, i.e., as a page whose relevant content is calculated (and thus potentially varies) at the time of access. In contrast, the content of *static* pages does not change at run-time. The classification of a page as being either static or dynamic is not necessarily unambiguous, because the definition of 'relevant content' is subject to interpretation. The distinction merely serves communicative purposes during modelling. There are no consequences on the implementation level.

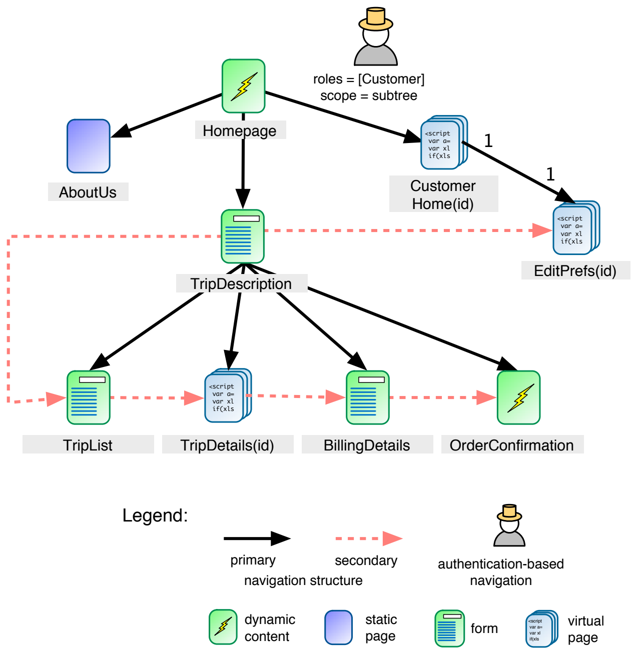


Figure 3. Navigation structure

Pages providing a form to let a web site visitor enter some data can be distinguished by a corresponding *form* icon.

A small piece of script code on a stacked page icon signifies a *virtual* page that is computed by a script. In contrast to the ‘lightning bolt’ pages with dynamic content, the script-generated pages are *entirely* calculated by a set of parameters, where one (the first) parameter defines the name of the page. The virtual pages do not exist under a pre-defined identifier, like the pages with dynamic content do. They are rather created and evaluated on-the-fly, every time the page is called. We will use the term *instance* to talk about concrete virtual pages. There is one instance for each possible identifier.

These four basic web page flavours can also be mixed on one page. A virtual, a static, or a dynamic page can contain a form (or more than one). Since non-dynamic virtual pages do not make much sense – because this would mean that every instance looked the same and did not make use of the identifying parameter – the lightning bolt adornment will not be applied to virtual page icons, and virtual pages will count as *always* being dynamic.

Technically, pages of all four page types are defined by a page function, and every page function has the *same* signature. Therefore, the page type chosen in the site map diagram is of no relevance implementation-wise (see section 2.3).

The navigation structure diagram may also contain information on *authorisation-dependent navigation*. In the example, the primary link to CustomerHome(id) is annotated with a role-icon. It states that a web site visitor must possess the role Customer in order to access the page. The scope=subtree declaration expands this constraint to the whole subtree rooted at this page. The alternative value thisPage for scope would prohibit this expansion. The actual mechanism for checking the authorisation of a given user, a given action and a given object is intentionally left open in our approach. We can encompass any matrix-based scheme that assigns any number of permissions to perform actions to a list of roles. Thus, the actual mechanism of the implementation platform can be used here.

The diagram can also capture the *multiplicity* of links to or from virtual pages. This is useful because virtual pages are like classes in that they represent a set of instances. Thus, we adopted a subset of the UML’s multiplicity symbols to lay down how many instances may be connected. In figure 3, each CustomerHome(id) instance is connected to exactly one instance of EditPref(id).

The actual checking of constraints and of the authorisation is contained in the associated page functions. They also contain the definition of the links for the secondary navigation structure. Thus, we can employ the full power of the underlying functional language to provide conditional links, whose behaviour or mere existence depends on the system state and other context information.

2.3 Pages

The pages that constitute the navigation space, i.e., the set of objects or nodes that can be visited, are called *navigation objects*. Each page is defined by a *page function* that returns an abstract regular structure which can be mapped to a concrete renderable page description using a presentation level language. Every time a page is accessed, the corresponding page function is evaluated, the result is transformed into the appropriate presentation language, and the result is sent to the client (see section 2.5).

We define a data type, the *abstract page description* (APD), that allows for defining a page on an abstract level in terms of nested, labelled, and attributed elements (analogous to XML). Here is the definition of this data type (in Haskell):

```
data APD =
       Text String
     | Element Name Attrs ElementList
     | Link Name Attrs ElementList Identifier Params
     | Form Name Attrs ElementList Identifier Params
     | Field Name Attrs String
     | Empty
```

There are six constructors for the APD type. An APD term can be a simple text node (Text); an element (Element) with a name, a list of attributes, and a list of child terms; a link or a form (Link and Form) with a name, a list of attributes, a list of child terms, the identifier of the destination page, and a list of parameters that should be passed to this page; a field in a form (Field) with a name, a list of attributes and a default field content; or it can simply be empty (Empty).

This declaration uses some type synonyms (syntactical abbreviations):

```
type Name = String
type Attrs = [(String, String)]
type ElementList = [APD]
type Params = Attrs
type Identifier = String
```

The name of an element is a string. Attributes and parameters are modelled as lists of key/value-pairs. Elements as well as forms can contain child nodes, so they use ElementList as a container for a list of arbitrary APD structures.

The type PageFunc is the function type for pages, mapping parameters to an APD structure. It is used for every page definition.

```
type PageFunc = Params → APD
type Id2PageFunc = Identifier → PageFunc
```

Links and form destinations are defined in terms of function identifiers. In this context, a function of the type Id2PageFunc maps identifiers to real page functions. This implies that links are represented by terms in an APD structure, attached with a reference to the page function they link to. This allows for link consistency checks.

First Example. As an example page function, consider the following (simplistic) definition:

```
greetingPage :: PageFunc
greetingPage [(”name”, name)] =
    Element ”pageheading” []
      [ Text ”Hello”
      , Element ”paragraph” [] [Text name]
      ]
```

This page is to be called with one parameter (called name) and results in an APD term that consists of a root element pageheading, which comprises a text element and a paragraph element. The latter finally produces another text element which corresponds to the value of the parameter that was passed to the page function. This is a first, albeit dull, demonstration of how to access page parameters and how to incorporate dynamic content.

Example with query. In general, the page functions are the place where we define the incorporation of the actual content into the pages. We suggest using a set of functions that encapsulate *queries* on the content repository. Since this is a separate level of abstraction, we treat these queries as a model of its own. Section 2.4 goes into more detail.

As an example, incorporating a list of a customer's transportation method preferences into the TripDescription page is done with a function

```
getMethodPrefs :: String → [String]
```

Given a customer's id as parameter, it returns a list of transportation method names. The actual query may be arbitrarily complex, but this does not matter at this point.

In the corresponding page fragment that calls this function, the delivered result is transformed into a list structure so it can be presented to the web site visitor:

```
−− generates a list of a customer’s preferences
tasPreferencesList :: PageFunc
tasPreferencesList params =
  let
    methodPrefs = getMethodPrefs                         (1)
      (fromJust (lookup ”customerId” params))
  in
    foreach
      methodPrefs
      (Element ”list” [] [])
      (λ(methodName) →
        Element ”item” [] [ Text methodName ]
      )
```

This listing shows the page function that builds an APD for the presentation of a customer's preferences (only the transportation methods are considered). The page function receives a customer id as its parameter. The main body of the function calls getMethodPrefs in the line marked (1) and then iterates over the resulting list of transportation method names. The iteration is achieved via the foreach construct. This is a simple iterator that applies a given function to each element in a given list and returns an APD with the result. Here, we can see how powerful the approach can get in respect to the specification complexity: iterators (and other control flow constructs, like higher level functions) can be defined and easily applied, thus lifting the specifications to a higher level of abstraction and allowing for more dynamics.

Example with template. For standard applications it appears to be appropriate to divide pages into several areas. Figure 4 visualises such a general page layout by showing a page template with *slots*. The slots are the places where the actual content is to be put in.

The presentation of a page as a function can be based on this template. As such, it serves as an additional abstraction of a page: the overall layout of the page in terms of its presentation can be defined without caring about the concrete content. The presentation of the content is again defined independently.

A page template is a function f : APD → APD. This signature emphasises that it is a *filter* that can be interposed in the transformation process (in the sense of a pipe/filter architecture). The functional approach allows for employing any number of these filters, resulting in further abstraction levels

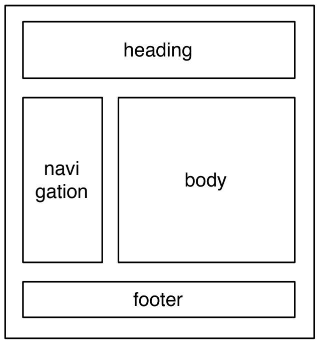


Figure 4. Page template with named *slots*

for pages. The levels can be established in correspondence to the complexity of the particular application at the time of modelling. There is no need to change the definition for the APD. The actual levels employed can even vary from page to page.

In order for a template to work as a filter, the *input* has to be an APD as well. Therefore, each filter traverses the given APD and searches it for content fragments that are marked with slot identifiers, thus signalling where to put them in the resulting page. That is, one can regard pages that make use of page templates as containing a *mapping* from slot identifiers to slot content. Calling a page template then involves creating the output APD and inserting the slot contents at the defined places by using this mapping.

We gain a further degree of coherence by defining the mapping as an APD as well. A page using a particular page template contains elements that have the same name as the slots defined in the page template. The children of these elements are then put at the appropriate place in the resulting APD.

The following listings show the definition of the TripDescription page. This page uses a page template (tasMainTemplate) whose Haskell code is given further below. The page template tasMainTemplate is called with an element named slots as its argument. This element serves as a container for the slot content mapping. The adherence to this convention is expected by the page template, which searches the given APD for the slots it is responsible for.

```
tasTripDescription :: PageFunc
tasTripDescription params = tasMainTemplate
  (Element ”slots” []
    [ Element ”heading” [] [Text ”Trip␣Description”]
    , Element ”navigation” [] [ tasNavigation params ]
    , Element ”body” []
      [ (tasTripDescriptionForm [])
      , (tasPreferencesList params)
      ]
    , Element ”footer” [] [ Text ”Footer” ]
    ])
```

Here is the main template:

```
−− Main template. Returns a complete HTML page.
tasMainTemplate :: APD → APD
tasMainTemplate (Element ”slots” a es) =
  Element ”html” a
    [ Element ”head” [] −− HTML head
      [ Text ”TAS” ]
    , Element ”body” [] −− HTML body
      [ Element ”div” [(”class”, ”heading”)]
          ( −− pageheading slot
            if (length headingContent) ≠ 0 then
              headingContent
            else
              [ ]
          )
      , Element ”div” [(”class”, ”body”)]
          ( −− body slot
            if (length bodyContent) ≠ 0 then
              bodyContent
            else
              [ ]
          )
      −− ( navigation slot, footer slot omitted )
      ]
    ]
  where
     headingContent =
        (filter (isElementWithName ”heading”) es)
     bodyContent =
        (filter (isElementWithName ”body”) es)
```

The function isElementWithName used in the **where** clause is a boolean function that returns true if, and only if, a given element has the name given.

The other functions necessary to constitute the complete TripDescription page are given below, to present a rather complete example. Note that they only build a very ‘naked’ version of that page. The page fragment building the navigation tree is omitted for the sake of brevity. The tree is defined by the primary navigation which is laid down in the navigation structure diagram.

```
tasNavigation :: PageFunc
tasNavigation _ = Empty

tasTripDescriptionForm :: PageFunc
tasTripDescriptionForm [] =
  let
    originCities = foldr
        (λa b → a ++ ”\n” ++ b) ”” getOriginCities
    destCities = foldr
        (λa b → a ++ ”\n” ++ b) ”” getDestCities
  in
    Form ”TripDescriptionForm” []
      [ Field ”originCity”
          [(”type”, ”optionlist”), (”values”, originCities)] ””
      , Field ”destCity”
```

```
        [(”type”, ”optionlist”), (”values”, destCities)] ””
      , Text ”Date␣of␣departure:”
      , Field ”dateOfDeparture” [] ””
      , Text ”Date␣of␣return:”
      , Field ”dateOfReturn” [] ””
      ]
      ”tasTripDetails” []
```

The get...-functions used here encapsulate queries to the content repository. For the example assume that getMethodPrefs returns [”Train”, ”Car”], getOriginCities returns [”Paris”], and getDestCities returns [”New␣York”, ”Rio”, ”Tokyo”] (section 2.4 will present an example query).

Besides the isElementWithName function, a whole set of auxiliary functions has been defined in order to make the definitions shorter and easier to maintain. Examples for tasks they perform is working with parameters lists, traversing APDs to find specific elements (especially links and forms), and displaying debug information.

One of the debugging functions prints a textual representation of an APD. Calling it with the results of the TripDescription page yields:

```
Element ”html” []
  [Element ”head” [] [”TAS”]
  ,Element ”body” []
    [Element ”div” [(”class”,”heading”)]
      [Element ”heading” [] [”Trip␣Description”]]
      ,Element ”div” [(”class”,”body”)]
        [Element ”body” []
          [Form ”TripDescriptionForm” []
            [Field ”originCity” [(”type”,”optionlist”),
                        (”values”,”Paris\n”)] ””
            ,Field ”destCity” [(”type”,”optionlist”),
                        (”values”,”New␣York\nRio\nTokyo\n”)] ””
            ,”Date␣of␣departure:”
            ,Field ”dateOfDeparture” [] ””
            ,”Date␣of␣return:”
            ,Field ”dateOfReturn” [] ””
            ]
            <page: ”tasTripDetails” []>
          ,Element ”list” []
            [Element ”item” [] [”Train”]
            ,Element ”item” [] [”Car”]
            ]
  ] ]  ] ]
```

Figure 5 visualises this result, hinting at the final presentation of the page. The abstract language used is quite close to XHTML already, but this is not a necessity. The final transformation into a concrete presentation level language is performed in a separate step, described in section 2.5.

2.4 Queries and Updates

Since the content model is given in terms of classes and relationships, it is possible to use almost any kind of representation for the underlying content repository. In our implementation we rely on a graph repository which keeps the data as *TGraphs* [3], i.e. typed, attributed and ordered directed graphs. This repository supplies a functional querying language that allows for accessing the graph at run-time, including updates.

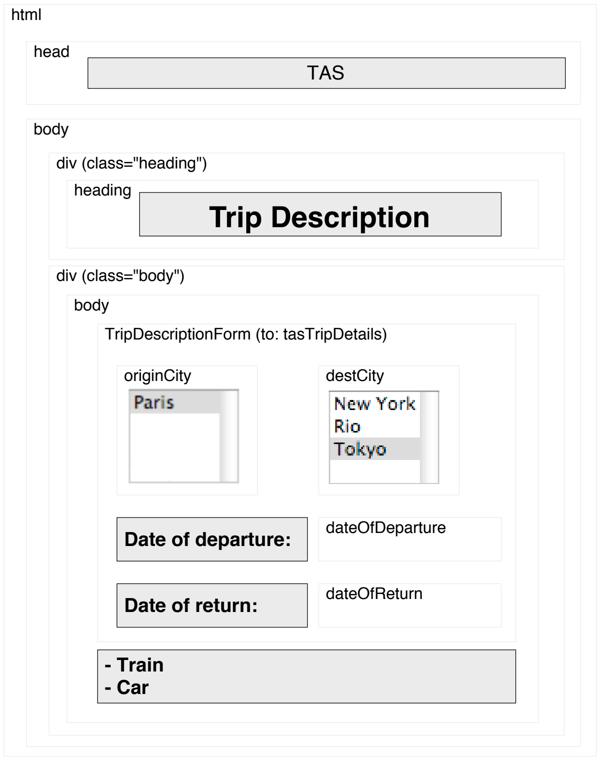


Figure 5. Visualisation of the abstract page structure for the TripDescription page.

The queries used for content integration can be of arbitrary complexity. While the query API may well be very simple, the full strength of the functional language can be used to work on the query results before incorporating them into the pages. Thus, each query is a proper function in its own right. This allows for defining arbitrarily complex *views* on the content. We prefer this amount of flexibility in favour of only allowing simple one-to-one mappings between the content model and the hypertext model. The goal is to keep the page functions as simple as possible and to avoid a too strong intermixing of content accessing functionality into the page definitions. This is perfectly achieved by using query functions that return simple objects like single ‘records’ or lists of records. The page functions then simply access the records or iterate over a list. Any necessary computation is encapsulated inside the query functions.

Consider, as an example, the following query that retrieves the list of all available cities:

```
queryAllCities :: AttributedGraph → [String]
queryAllCities g =
  nodesToValues
    g
    (λ lbl → getValue lbl "id")
    (query g (nodes g) [ constrainByType "City" ])
```

Without diving into the implementation details, we can note that this function returns a list of strings, given a concrete Graph g, by first selecting all nodes that are of type City, and then mapping a function that extracts the value of the id attribute over this list of nodes, resulting in the desired list of city names.

2.5 Presentation

The presentation model is given by defining one or more mappings (presentation functions) from an APD to the corresponding presentation level language. In the case of a web application that is to be delivered via HTTP and is destined to be rendered by a user agent that understands XHTML, a simple transformation of the regular APD into XHTML can be implemented as a Haskell function. Alternatively, the APD could be converted to any other XML dialect first, and subsequent transformations may be done with technologies like XSLT. All conceivable possibilities are open at this point, and the approach can be easily adapted to a great number of run-time systems. Note that the actual transformations can be selected at run-time, even on a page-to-page basis, or according to *context* information. This opens the path to customisation, personalisation, and multi-mediality.

For our current implementation, we use the *Zope* [4] web application server and we integrated a Haskell interpreter (Hugs, [5]) that evaluates page functions on-the-fly. The page functions (via a presentation function) return complete XHTML documents which can be delivered without further transformation.

2.6 Dynamics

The behavioural aspect of the TAS can be modelled elegantly by looking at the different functions that have to be carried out by the actors. Thus, the 'business logic' is broken down into well-specified functions that can be glued together in the page function definitions. Table 1 lists three example functions. It states the function provider and the name of the function in the first line, followed by a short description of its semantics. The rows marked 'In' and 'Out' describe the input and output parameters, respectively.

Since the page functions are executable formal specifications, the dynamics of the system can be subjected to simulation and testing. One can devise test cases by anticipating the results of relevant functions and test them by calling them with appropriate arguments.

Table 1. Function list

PTA.checkWellformedness	
	Checks a trip description for plausibility (e.g., return date lies before departure date)
In	trip description (TripDescription)
Out	true, if trip description is well-formed, i.e., plausible; false, otherwise
PTA.prepareTripList	
	Processes a well-formed trip description and presents a list of matching trips
In	trip description (TripDescription)
Out	ordered list of trips that match the trip description
TC.bookTemporarily	
	Temporarily books a transportation service, if possible. Reports any failure.
In	service request (Route with associated origin, destination, and transportation method; date of departure)
Out	confirmation of success (true or false)

PTA: personal travel agency; TC: travel company

3 Integration

The integration of the different models just described in section 2 is achieved by making functions call one another and by referring to the concepts from the content model in the functional specifications. Figure 1 depicts these relationships. A page function declaratively specifies what one navigation object consists of, thus being a model for it. At the same time, it is a callable function that returns the actual page upon evaluation. In other words, the *functions are executable models*. The page functions call query and update functions as well as dynamics functions, which makes them the integral and pivotal glue that holds the system together. They also implement the secondary navigation structure by including corresponding links.

By accessing instance data that complies to the content model, the functions refer to concepts from the application domain. Functional requirements are captured by specific dynamics functions that also contribute to building higher-level, integrating and integrated functions that can be used in the other specifications.

We showed how a complete website can be described by six different kinds of documents. Content is modeled by a conventional UML class diagram and the site's structure is modeled by a site graph. The site graph is written in a visual language that differentiates between four different page types (static, dynamic, form-based, and virtual pages) and allows to include authorisation-dependent navigation constraints.

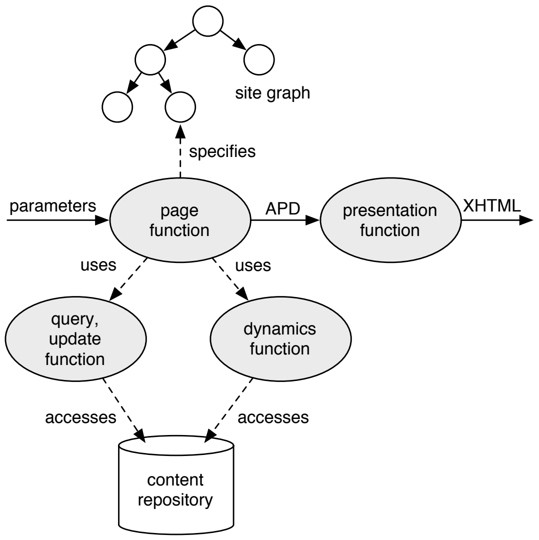


Figure 6. Artefact integration: data flow

All other information necessary to fully define the web site, i.e., the pages themselves, including all page fragments that are part of them, the queries supplying content information to the pages, the rendering of the pages for presentation, and the dynamics of the pages, are described as functions using a functional language.

All this effectively guarantees a high level of coherency. Figure 6 visualises the integration of the different artefacts, focussing on the flow of data, from the receipt of parameters to the production of a document in the desired output language. The page functions are the integrating unit, using queries, updates, and dynamics functions to define the incorporation of data into the pages and the execution of 'business logic'. The page functions also specify the site graph by creating APD terms that represent links.

4 Related Work

Relying on models for describing and specifying web sites has quite a long tradition. Overviews and comparisons of the most prominent approaches are given e.g. in [6], [7], and [8]. The approaches can be very coarsely classified by their 'foundations': some focus on object-oriented models, others rely on entity-relationship models, and again others put documents into the center of interest. The most influential 'schools' are the graph-based Strudel approach [9], the TSIMMIS project [10], the ER-based RMM [11], Araneus [12], HDM [13] and OOHDM [14], WebML [15], and UWE [16].

Significant effort has been put in developing and describing diverse methodologies for web site generation, of which none, to our knowledge, relies as much on functional specifications as we do. We envision a synergetic potential for the integration of our findings into existing approaches, or, vice versa, the integration of selected parts of the aforementioned approaches into ours. This vision was the reason for our approach being as abstract and as extensible as possible. The idea of integrating the models by making them functions, which is unique to our approach, clearly works best when *all* models are specified as functions.

It is interesting to compare the various notations used in the respective approaches. Some approaches rely on proprietary notations for some of the diagram types, especially for the hypertext models. A majority of the current approaches employs the UML (and its extension mechanisms) for the notation of diagrams. The main reasons stated for using UML are the availability of tools ([17, p. 2]), the fact that the UML is well-documented ([18, p. 2]), and the coherence gained by using UML for a web application that is connected to other systems that are already modelled using UML ([19, p. 64]). As of today, one can state that using UML class diagrams for the notation of entity-relationship views simply is standard practice.

It is not just recently that the community realises that *aspect-orientation* can very well be applied to web engineering problems. Many cross-cutting aspects have been identified (principally: authorisation, contextuality), and the models are defined in a way to allow for the inclusion of aspects (e.g., [20]). Our approach nicely fits into this line, as the functions offer well-defined cut points.

Our approach is based on functional specifications. We aim at integrating the advantages of this 'way of thinking' into existing web engineering practice. To the best of our knowledge, only very little effort has been put into this direction. Producing HTML and XML with a functional language in a type-safe way is, e.g., investigated in [21], [22]. A way of representing graphs in Haskell is proposed in [23], accompanied by a working implementation.

5 Summary and Conclusion

This text presented a coherent approach to web engineering by relying on functional specifications. Page functions are the formal foundation upon which whole web sites can be built.

At the same time, the approach is general and open enough to incorporate existing tools and technologies, like graph querying for the inclusion of content, web application servers or web content management software that provide session management and access control, or configuration management software to take care of the artefacts produced during development and evolution. Testing and simulation can be done by executing the specifications.

Note that abstracting pages, the dynamics, queries, and updates as functions effectively paves the way to the encapsulation of selected functions as *web services*.

Exploiting the ease of use and extensibility of the functional approach, we introduced a simple template mechanism to describe pages on a higher level of abstraction.

Once a consolidated library of functions is available, creating the functional specifications is not very hard work. However, there is also room for further improvement, especially when thinking of software tools that let the web engineer work in a more visual environment. These tools then deliver the functional specification as their *output*. It might make sense to constrain the expressive power of these tools, so as to be able to maintain a two-way consistency between the generated functional specifications and the visual documents shown to the user.

We regard our approach as a contribution that shall serve as a foundation, paving the ground for building upon and exploiting its possibilities.

References

[1] J. Peterson and O. Chitil, "The Haskell Home Page." http://www.haskell.org/, Dec 2004.

[2] "A travel agency system." http://www.lcc.uma.es/~av/mdwe2005/TheTASexample/, May 2005.

[3] J. Ebert and A. Franzke, "A Declarative Approach to Graph Based Modeling," in *Graphtheoretic Concepts in Computer Science* (E. Mayr, G. Schmidt, and G. Tinhofer, eds.), no. 903 in LNCS, (Berlin), pp. 38–50, Springer, 1995.

[4] "Zope.org." http://www.zope.org, May 2005.

[5] "Hugs 98 Web Site." http://www.haskell.org/hugs/, August 2004.

[6] N. Koch, "A comparative study of methods for hypermedia development," Technical Report 9905, Ludwig Maximilians-Universität München, November 1999.

[7] P. Fraternali, "Tools and approaches for developing data-intensive Web applications: a survey," *ACM Computing Surveys*, vol. 31, no. 3, pp. 227–263, 1999.

[8] G. Kappel, B. Pröll, S. Reich, and W. Retschitzegger, eds., *Web Engineering: Systematische Entwicklung von Web-Anwendungen*. Heidelberg: dpunkt.verlag, 2004.

[9] M. Fernández, D. Florescu, A. Y. Levy, and D. Suciu, "Declarative specification of Web sites with Strudel," *VLDB Journal*, vol. 9, no. 1, pp. 38–55, 2000.

[10] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom, "The TSIMMIS project: Integration of heterogeneous information sources," in *16th Meeting of the Information Processing Society of Japan*, (Tokyo, Japan), pp. 7–18, 1994.

[11] T. Isakowitz, E. A. Stohr, and P. Balasubramanian, "RMM: A methodology for structured hypermedia design," *Communications of the ACM*, vol. 38, no. 8, pp. 34–44, 1995.

[12] G. Mecca, P. Merialdo, and P. Atzeni, "Araneus in the era of XML," *IEEE Data Engineering Bulletin*, vol. 22, pp. 19–26, September 1999.

[13] F. Garzotto, P. Paolini, and D. Schwabe, "HDM – a model-based approach to hypertext application design," *ACM Transactions on Information Systems*, vol. 11, no. 1, pp. 1–26, 1993.

[14] D. Schwabe and G. Rossi, "The object-oriented hypermedia design model," *Communications of the ACM*, vol. 38, no. 8, pp. 45–46, 1995.

[15] S. Ceri, "Web Modeling Language (WebML): a modeling language for designing Web sites," *Computer Networks (Amsterdam, Netherlands: 1999)*, vol. 33, no. 1–6, pp. 137–157, 2000.

[16] A. Knapp, N. Koch, G. Zhang, and H.-M. Hassler, "Modeling business processes in web applications with ArgoUWE," in *UML 2004 - The Unified Modeling Language. Model Languages and Applications. 7th International Conference, Lisbon, Portugal, October 11-15, 2004, Proceedings* (T. Baar, A. Strohmeier, A. Moreira, and S. J. Mellor, eds.), vol. 3273 of *LNCS*, pp. 69–83, Springer, 2004.

[17] E. Gorshkova and B. Novikov, "Exploiting UML extensibility in the design of web information systems," in *Proc. Fifth International Baltic Conference on Databases and Information Systems*, (Tallinn, Estonia), pp. 49–64, June 2002.

[18] N. Koch, A. Kraus, and R. Hennicker, "The authoring process of the UML-based Web engineering approach." http://www.dsic.upv.es/~west/iwwost01/files/contributions/NoraKoch/Uwe.pdf, June 2001. (on-line).

[19] J. Conallen, "Modeling Web application architectures with UML," *Communications of the ACM*, vol. 42, no. 10, pp. 63–70, 1999.

[20] G. Zhang, H. Baumeister, N. Koch, and A. Knapp, "Aspect-oriented modeling of access control in Web applications," in *Proc. 6th Int. Wsh. Aspect Oriented Modeling (AOM)*, (Chicago, Illinois, USA), March 2005.

[21] P. Thiemann, "Modeling HTML in Haskell," in *Practical Aspects of Declarative Languages* (E. Pontelli

and V. S. Costa, eds.), vol. 1753 / 2000, (Second International Workshop, PADL 2000, Boston, MA, USA), p. 263, Jan 2000.

[22] P. Thiemann, "A typed representation for HTML and XML documents in Haskell," *Journal of Functional Programming*, vol. 12, pp. 435–468, July 2002.

[23] M. Erwig, "Inductive graphs and functional graph algorithms," *Journal of Functional Programming*, vol. 11, no. 5, pp. 467–492, 2001.

How to Model Aspect-Oriented Web Services

Guadalupe Ortiz
Juan Hernández
Pedro J. Clemente
Pablo A. Amaya
Quercus Software Engineering Group
University of Extremadura
Computer Science Department
gobellot@unex.es
juanher@unex.es
jclemente@unex.es
pabloama@unex.es

Abstract

Web Services provide a new and successful way of enabling interoperability among different web applications. In this paper, an MDA approach to modelling Web Services, in which aspect-oriented techniques are also applied, is provided. The UML profiles required to model aspects and Web Services independently from the platform (PIM) are presented. Once the system is modeled at this level of abstraction, transformation rules have to be applied in order to obtain the platform specific model (PSM). In this respect, two different approaches for the specific model are discussed, and benefits and shortcoming will be analyzed depending on whether the aspects' weaving is performed at design or implementation level.

1. Introduction

Web Services have become the new way to implement and compose applications through the Web, and they have had a great impact in the current way of developing applications.

Once Web Service technology seems to be highly consolidated, it is time to tackle how they can be modelled in order to be able to generate the necessary code automatically. Although it is influential to both evolution and maintenance in Web Services, the market has not proposed a suitable answer to this matter for the time being.

Furthermore, in spite of the importance of extra-functional properties' implementation, and interaction logic encapsulation in compositions in the area of Web Services, a unique and valid proposal has not yet been suggested. We have already proposed the encapsulation of both extra-functional properties and the composition interaction logic by using aspect-oriented programming (AOP) [1] as a suitable way to implement these aspects of Web Service development [2][3][4], but these issues must also be addressed in a more abstract level.

For this reason, we propose to model Web Services and their composition in a well modularized way, maintaining the logic decomposition of units by using a new profile to model aspects and another one to model Web Services altogether, independently of the platform. In addition, once we have the independent model, it has to be transformed into a platform specific one. Due to the presence of aspects, we find two different alternatives for the target specific model: firstly, to perform the weaving at transformation, so that no aspects remain in the specific model; secondly, to maintain the aspects in the specific model, keeping them abstract or translating them into a specific aspect-oriented language. Therefore, the main contributions of this paper are the definition of a PIM model for Web Services, in which aspect-oriented techniques have also been applied, and the analysis of two different approaches to the specific model depending on whether the aspects' weaving is performed at design or implementation level.

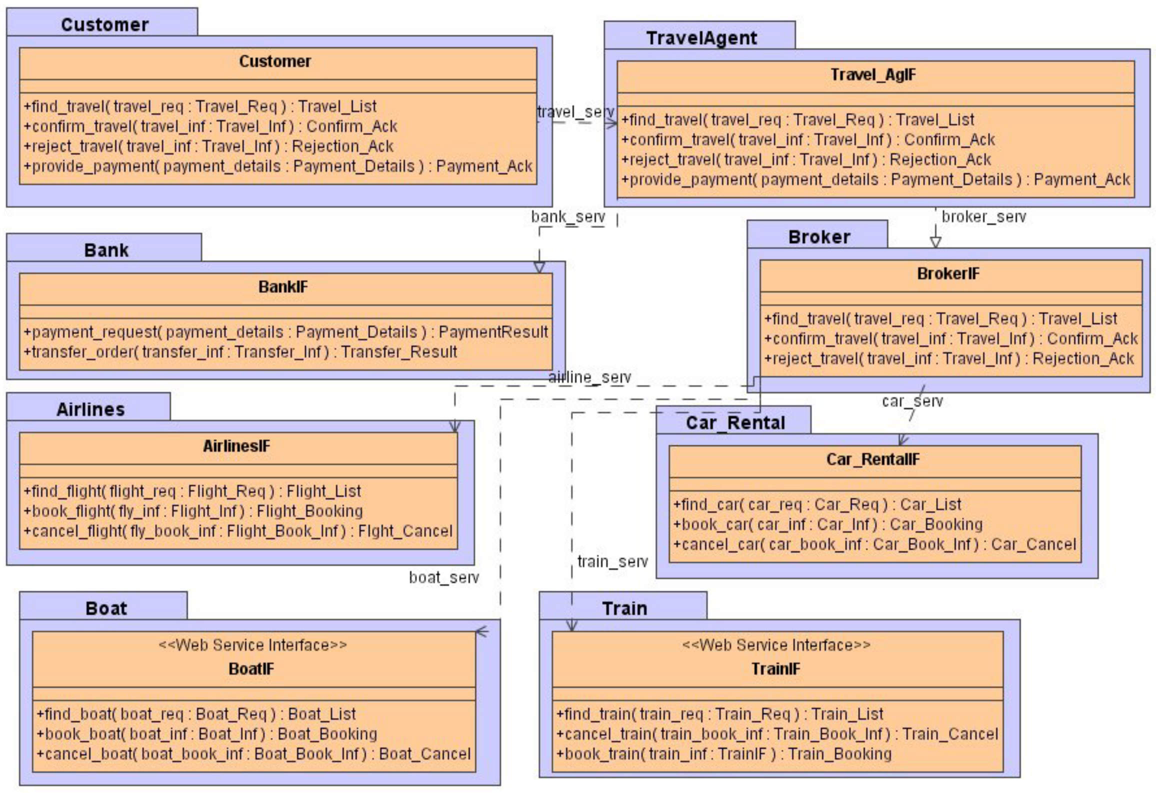
The rest of the paper will be arranged as follows: a case study PIM is presented in Section 2 to identify the various operations offered by the services and the way in which they are connected, highlighting how the aspects appear to deal with extra-functional properties and compositions' interaction logic. Section 3 specifies both the way to model Web Services and the one to model aspects and then moves on to outline the PIM appearance once it has been decided to include the aforementioned aspects. Section 4 presents two alternatives for the aspect weaving, and the influence of the decision in later stages will be examined. To finish with, we discuss our proposal and future work in Section 5.

2. The Case Study

The case study is not detailed in depth in this paper as a common example was proposed for all the position papers submitted to this workshop. With the sole purpose of clarifying the main operations in the different services and the relations among them, a very general PIM is presented in *Figure 1*. To avoid confusion, points which had no relevance to our proposal have been omitted from the diagram. .

Consider now that we want to add extra-functional properties to the case study. The tangling resulting from the added properties causes difficulties when designing the model in a conventional way. The same problem is encountered when trying to model the compositions' interaction logic in an independent unit. Both issues have been studied and solved at programming level, as mentioned in the introduction, but not at design level. In this sense, our proposal aims at solving this inadequacy.

Therefore, various aspects can be included in our case study in order to model extra-functional properties and compositions' interaction logic in a modular and structured way. The *customer*, the *travel agent* service and the *bank* service need to communicate in a secure way, as these are the three elements which send information, such as credit card numbers, which must travel securely over the net. This security can be provided by the aspect *Encryption*. We can also use the *Logging* aspect in the travel agency service to maintain a file with all the operations invoked. Finally we have also decided to include the T*ravel_agComp* aspect, which encapsulates the composition interaction logic for the different services required to offer the travel agent behaviour. Similarly, the B*rokerComp* aspect may be included; its task would be to encapsulate the interaction logic of the different services required to offer the broker behaviour.



Customer
Customer
+find_travel(travel_req : Travel_Req) : Travel_List
+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
+provide_payment(payment_details : Payment_Details) : Payment_Ack
travel_serv
TravelAgent
Travel_AgIF
+find_travel(travel_req : Travel_Req) : Travel_List
+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
+provide_payment(payment_details : Payment_Details) : Payment_Ack
bank_serv
broker_serv
Bank
BankIF
+payment_request(payment_details : Payment_Details) : PaymentResult
+transfer_order(transfer_inf : Transfer_Inf) : Transfer_Result
Broker
BrokerIF
+find_travel(travel_req : Travel_Req) : Travel_List
+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
airline_serv
car_serv
Airlines
AirlinesIF
+find_flight(flight_req : Flight_Req) : Flight_List
+book_flight(fly_inf : Flight_Inf) : Flight_Booking
+cancel_flight(fly_book_inf : Flight_Book_Inf) : Flght_Cancel
Car_Rental
Car_RentalIF
+find_car(car_req : Car_Req) : Car_List
+book_car(car_inf : Car_Inf) : Car_Booking
+cancel_car(car_book_inf : Car_Book_Inf) : Car_Cancel
train_serv
boat_serv
Boat
<<Web Service Interface>>
BoatIF
+find_boat(boat_req : Boat_Req) : Boat_List
+book_boat(boat_inf : Boat_Inf) : Boat_Booking
+cancel_boat(boat_book_inf : Boat_Book_Inf) : Boat_Cancel
Train
<<Web Service Interface>>
TrainIF
+find_train(train_req : Train_Req) : Train_List
+cancel_train(train_book_inf : Train_Book_Inf) : Train_Cancel
+book_train(train_inf : TrainIF) : Train_Booking

Figure 1. Case Study PIM

3. The Web Service and Aspect-Based PIM

In this section we are going to introduce our proposal in three steps: first of all, our proposal for modelling Web Services will be explained; secondly, our proposal for modelling aspects will be described and finally, the complete PIM proposed for the case study, based on previous premises, is presented.

3.1 Web Service Modelling

We can find many Web Service modelling-related proposals, such as [9] [10] [11]. From many of them it can be noted that most of the literature in this area tries to find an appropriate way to model service compositions with UML and most of them use the WSDL structure in order to model services.

The research presented by J. Bezivin et. al [12] is worth a special mention; in it Web Service modelling is covered in different ways, finally using *Java* and *JWSDP* implementations. Starting from this proposal, first of all, we decide to model our services representing the WSDL elements, but in order to simplify the model for our case study we will only show the *Web ServiceIinterface*, which would match the binding element in the WSDL metamodel representation in the said report. In this sense, we could extend the Web Services modelled in our case study to the rest of the elements in the WSDL metamodel.

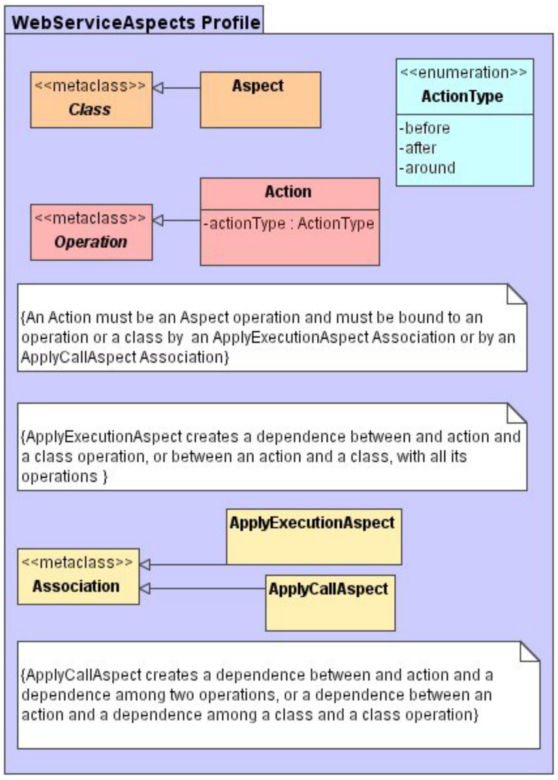
.

3.2 Aspects Modelling

AOSD (Aspect-Oriented Software Development) has been widely studied in the specialised literature, and we can find different proposals on how to model aspects with current products. We can find various proposals mainly oriented to a specific language, such as *AspectJ* [5] [6], or more general ones [7] [8]. The first ones show *pointcuts* and *advices* to be clearly distinguishable, whereas the second ones are not so oriented to *AspectJ* terms, but only to the points to be intercepted and their associated behaviours.

Our goal is to obtain a model which is independent from the platform, so we have followed the more general proposal to make ours, where we define the aspects in a completely independent way, as depicted in *Figure 2*.

Aspect-oriented techniques describe five types of elements to modularize crosscutting concerns: firstly, we have to define the *join point* model which indicates the points where new behaviours could be included. Then a way to indicate the specific *pointcuts* needs to be defined, to specify in which points of the implementation we wish for the new code to be inserted. Next, we ought to determine how we are going to specify the new behaviour to be included in the *join point* referred to. We would then encapsulate the specified join points and their corresponding behaviours into independent units. Finally, a method to weave the new code with the original one has to be applied [1].



{An Action must be an Aspect operation and must be bound to an operation or a class by an ApplyExecutionAspect Association or by an ApplyCallAspect Association}

{ApplyExecutionAspect creates a dependence between and action and a class operation, or between an action and a class, with all its operations }

{ApplyCallAspect creates a dependence between and action and a dependence among two operations, or a dependence between an action and a dependence among a class and a class operation}

Figure 2. Aspect-Oriented UML Profile

Therefore, in *Figure 2* we can see the new stereotypes defined in order to model the five aspect-oriented types of elements just described, which are briefly going to be outlined:

- Due to the fact that services are *black boxes*, there will be two possible types of interaction points in Web Service applications: firstly, the point in which a service operation is to be executed, regardless of who made the invocation; secondly, when a service client (which may also be another service) invokes a specific operation in the target service. The first ones will be referred to as *executionAspects* from now on in this article, as they are triggered when the service method is about to be executed; on the other hand, the second ones will be called *callAspects* since they are triggered when a specific operation is invoked by the client. Therefore, these two types of point in our application´s execution conform our join point model.

- In order to identify the pointcuts, two stereotyped associations have been added. In *executionAspects*, the association *ApplyExecutionAspect* binds the action, which implements the new behaviour to be included, to the operation whose execution is being intercepted; this association represents the pointcut in this type of aspect. For *callAspects* pointcuts, we have added the stereotyped association *ApplyCallAspect*, which binds *Action* to the invocation of a service method by a client one.
- The advices are represented by the *Action* stereotype which can be applied to operations and which will have a *tag Value* to indicate the type of action (*Before, After* or *Around*), that is, if the advice behaviour is going to be injected before, after or around the pointcut.
- The stereotype *aspect* contains *Actions*, which are linked to a service method by the *ApplyExecutionAspect* association or to a dependence between the client and a service method by the association *ApplyCallAspect*.
- Regarding *weaving*, two different alternatives will be studied in *Section 4*.

3.3 The Aspect-Oriented Case Study PIM

In this section, aspects and Web Service modelling are joined in the case study to illustrate our proposal. The complete case study model, in which various aspects have also been represented, is depicted in *Figure 7,* annexed at the end of this paper. Some details have been omitted in order to simplify model's presentation.

The structure of Web Services, as we mentioned before, limits the join point model [13] to the interface method execution on server-side and to their calls on client-side. The reason is that Web Services are implemented as *black boxes*, where only the interface with the offered operations is shown, thus limiting the logical join point model to those operations. Regarding actions associated to pointcuts, we have to indicate the action type, *before*, *after* or *around*, depending on the moment in which the action is to be executed in relation to the pointcut.

In order to illustrate how Web Services and aspect modelling are integrated in the case study, we have partitioned the complete PIM model into small pieces which allows us to explain the model definition in detail. In this respect, we are firstly going to study the application of an extra-functional property, which leads to the creation of a new *executionAspect*; then we will analyse the application of an extra-functional property which leads to the appearance of both an *executionAspect* and a *callAspect* and, finally, the application of one aspect which encapsulates the composition logic of one service, which, as we will see, leads to an *executionAspect* ocurrence.

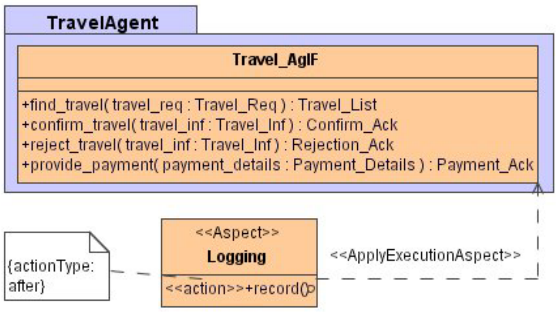

TravelAgent
Travel_AgIF
+find_travel(travel_req : Travel_Req) : Travel_List
+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
+provide_payment(payment_details : Payment_Details) : Payment_Ack
{actionType: after}
<<Aspect>>
Logging
<<action>>+record()
<<ApplyExecutionAspect>>

Figure 3. ***Logging*** **Aspect Application Modelling**

To start with, we can see how an extra-functional property, which solely affects one service, is modelled. As can be seen in *Figure 3*, a *logging* property is applied to the travel agent service. The Figure shows the stereotyped *Aspect Logging*, which contains *Action record*. This action has an associated note that specifies that it is an *after* action, that is, that the action will be executed after the intercepted method execution. Besides, the action is bound to *Travel_AgIF*, the service interface, by the stereotyped association *ApplyExeuctionAspect*, which means that the action will be applied to the whole method in *Travel_AgIF* class. As a result, every time any *Travel_AgIF* method is executed the record action will be triggered, after the method execution is complete.

Figure 4 shows how the *Encryption* property is modelled in our case study. This property, as can be observed in the above Figure, is applied to customer and travel agent through the aspect stereotypes *Server_Encryption* and *Client_Encryption*, with its associated stereotyped action *EncryptionAction.* As can be observed from the Figure, *Server_Encryption* is an *executionAspect*, thus it is modelled in the same manner as *Logging*: the action in *ServerEncryption*,

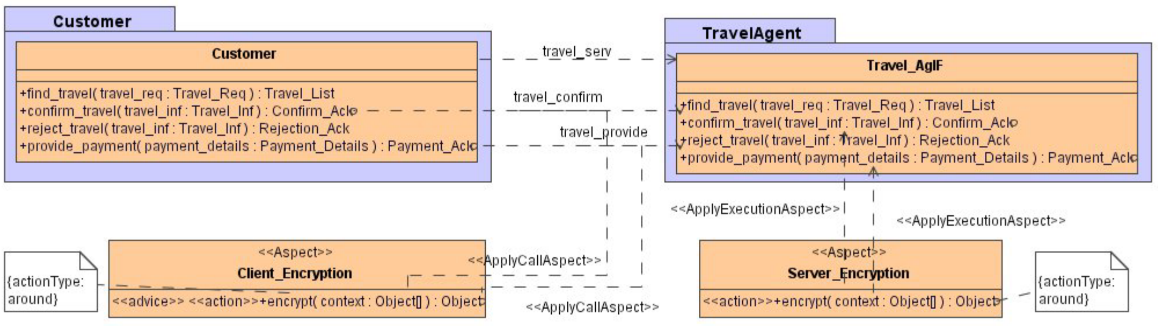

Customer
Customer
+find_travel(travel_req : Travel_Req) : Travel_List
+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
+provide_payment(payment_details : Payment_Details) : Payment_Ack
travel_serv
travel_confirm
travel_provide
TravelAgent
Travel_AgIF
+find_travel(travel_req : Travel_Req) : Travel_List
+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
+provide_payment(payment_details : Payment_Details) : Payment_Ack
<<ApplyExecutionAspect>>
<<ApplyExecutionAspect>>
{actionType: around}
<<Aspect>>
Client_Encryption
<<advice>> <<action>>+encrypt(context : Object[]) : Object
<<ApplyCallAspect>>
<<ApplyCallAspect>>
<<Aspect>>
Server_Encryption
<<action>>+encrypt(context : Object[]) : Object
{actionType: around}

Figure 4. *Encryption* Aspect Application Modelling

this time an *around* action, is linked by the use of an *ApplyExecutionAspect* association to the methods *confirm_travel* and *provide_payment* in travelAgent. This means that any time these methods are executed, the *ServerEncryption* action will be executed *around* it (This action would desencrypt the parameters received in the invocation, then allow method execution with the desencrypted parameters and finally would re-encrypt the method result) The case of *Client_Encryption* is different as it is a *callAspec*. *Figure 4* shows how the action in *Client_Encryption* is bound to the dependence between *Customer confirm_travel* and *Travel:Agent confim_travel* and between *Customer provide_payment* and *Travel_Agent provide_payment*. In fact, it could be reduced to the dependences between *Customer* and methods *confirm_travel* and *provide_payment* in *TravelAgent*. This means that any time these two methods are invoked by the customer, the *Client_Encryption* action will be applied *around* (The call is intercepted, then invocation parameters are encrypted, after that the call goes on and, to finish with, when the result comes back, it is desencrypted before it is reused). We can now see how the same properties are reused in the case study to manage security between the travel agent and the bank. As can be observed in *Figure 5*, now the *Server_Encryption* action is also applied to *Bank payment_request*, thus any time this method is executed, the associated action will be executed *around*. Moreover, the named Figure shows that a

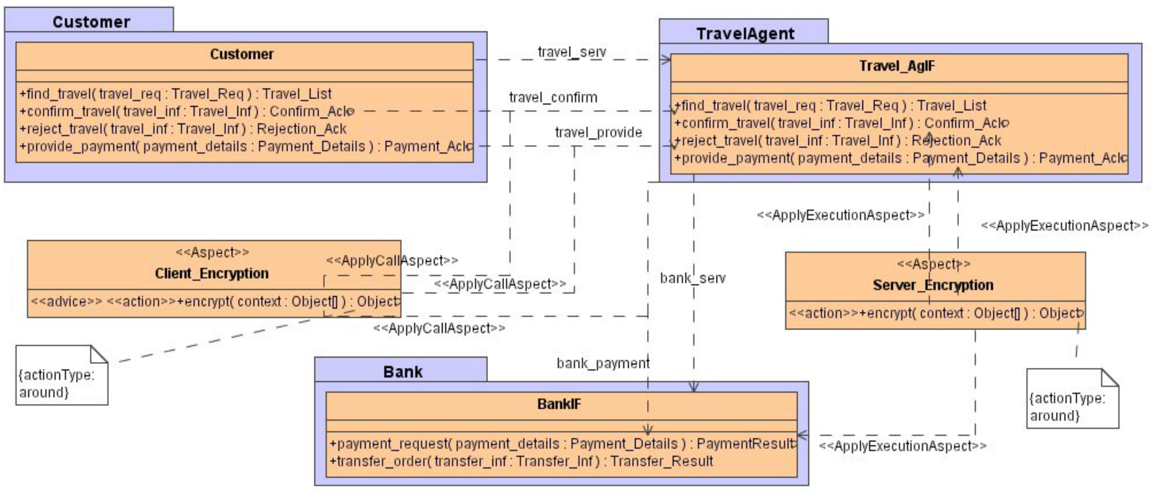

Customer
Customer
+find_travel(travel_req : Travel_Req) : Travel_List
+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
+provide_payment(payment_details : Payment_Details) : Payment_Ack
travel_serv
travel_confirm
travel_provide
TravelAgent
Travel_AgIF
+find_travel(travel_req : Travel_Req) : Travel_List
+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
+provide_payment(payment_details : Payment_Details) : Payment_Ack
<<ApplyExecutionAspect>>
<<ApplyExecutionAspect>>
<<Aspect>>
Client_Encryption
<<advice>> <<action>>+encrypt(context : Object[]) : Object
<<ApplyCallAspect>>
<<ApplyCallAspect>>
<<ApplyCallAspect>>
bank_serv
<<Aspect>>
Server_Encryption
<<action>>+encrypt(context : Object[]) : Object
{actionType: around}
bank_payment
Bank
BankIF
+payment_request(payment_details : Payment_Details) : PaymentResult
+transfer_order(transfer_inf : Transfer_Inf) : Transfer_Result
<<ApplyExecutionAspect>>
{actionType: around}

Figure 5. *Encryption* Aspect Reusing Modelling

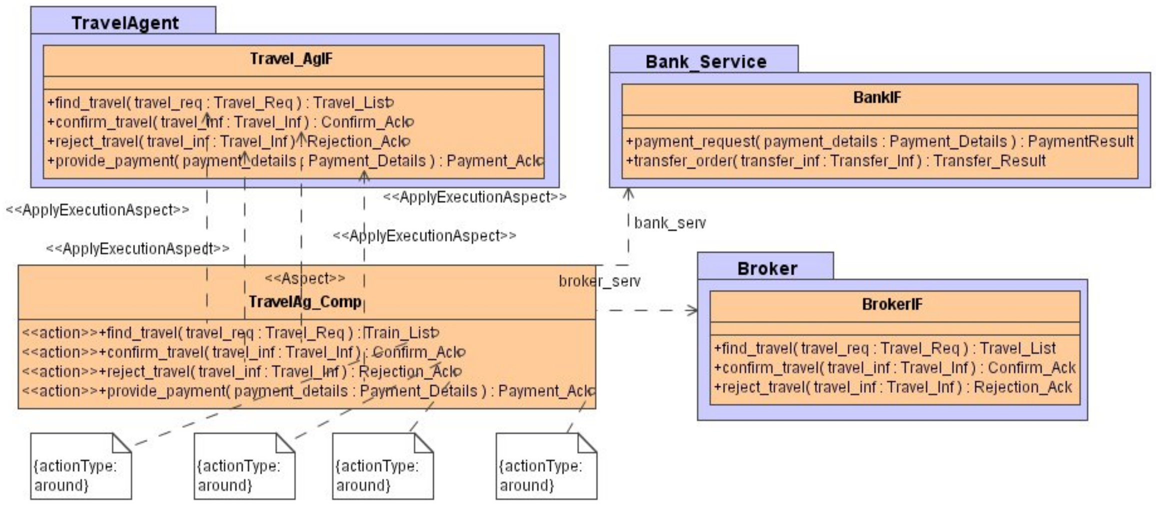

TravelAgent
Travel_AgIF
+find_travel(travel_req : Travel_Req) : Travel_List
+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
+provide_payment(payment_details : Payment_Details) : Payment_Ack
Bank_Service
BankIF
+payment_request(payment_details : Payment_Details) : PaymentResult
+transfer_order(transfer_inf : Transfer_Inf) : Transfer_Result
<<ApplyExecutionAspect>>
<<ApplyExecutionAspect>>
<<ApplyExecutionAspect>>
<<ApplyExecutionAspect>>
bank_serv
broker_serv
<<Aspect>>
TravelAg_Comp
<<action>>+find_travel(travel_req : Travel_Req) :Train_List
<<action>>+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
<<action>>+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
<<action>>+provide_payment(payment_details : Payment_Details) : Payment_Ack
Broker
BrokerIF
+find_travel(travel_req : Travel_Req) : Travel_List
+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
{actionType: around}
{actionType: around}
{actionType: around}
{actionType: around}

Figure 6. ***Travel_AgComp*** **Aspect Application Modelling**

Client_Encryption aspect is also associated to any invocation made by the *Travel_Agent* service to the *provide_payment* method in *Bank_Service*. In this case, the travel agent behaves as a client as it is the one making the call.

Finally, as we have previously explained, not only are extra-functional properties modelled into individual units, but also compositions' interaction logic, as depicted in *Figure 6*, where it can be seen that the *Travel_AgComp* Aspect is applied to the travel agent service. This aspect will define the interaction logic for composing the different services in the case study for the *Travel_ag* service to offer its behaviour. As depicted in *Figure 6*, the aspect actions are linked to different methods in *TravelAgent*, as there will be different actions for every method execution. Each action refers to the execution of one of the operations on offer: one will be associated to each method to offer its composition interaction logic. Therefore, four different actions can be found in *Travel_AgComp*.

To illustrate this idea, we can assume that the *confirm_travel* operation is invoked by the customer. The behaviour associated to the action *confirm_travel* in *Travel_AgComp* will be the one to trigger the bank service invocation to check the credit card provided and, depending on the result, invoke the broker to confirm this trip and cancel the rest, or ask the client to reenter credit card information. By modelling this interaction logic as an aspect action, all dependences among the composed services will be avoided, the action being the one which will compose them in a decoupled way.

Similarly, not represented in this Figure but available in the Appendix, the broker service has the aspect *BrokerComp* applied to it, where three different actions are defined for the execution of the three different operations on offer. Similarly to *Travel_AgComp Aspect*, *BrokerComp* also has one action associated to each method in order to develop the behaviour offered by the broker service through the composition of various services' invocations (*airline* service, *car_rental* service, *train_rental* service and *boat_rental* service).

4. Weaving Alternatives

Once we have our system aspect-oriented PIM we have to decide when we desire the weaving to be performed. We have two alternatives, which are going to be discussed in the next subsections: doing the weaving in the PIM-PSM transformation, and therefore eliminating aspects from the specific model, or letting aspects be in the specific model in order to undergo weaving at a later stage.

4.1 PSM Without Aspects

We may decide not to have aspects in our platform specific model, therefore the PIM aspects weaving would be taking place in the transformation from one model to the other.

The main advantage of this proposal is that the obtained PSM does not need to also specify the metamodel of an aspect-oriented platform or language, thus resulting in a simpler model. We would just have to specialise the model to the platform and language used to define the services. Consequently, the specific model would look simpler than if we still had unweaved aspects; this allows the possibility of reusing any process of automatic code generation which may already have been developed for Web Service modelling; hence, we could use any previously implemented tool to generate code from the sequence diagram automatically, which could not be achieved with the second alternative shown in *Section 4.2*.

On the other hand, simplicity is also occasionally a drawback, as by obtaining a simpler model traceability problems arise, that is, we cannot recover aspects at a later stage by separating them from the rest of the elements. Furthermore, although we have no aspects in the specific platform class model, on executing aspects weaving their behaviour must somehow be reflected in the sequence or interaction diagrams. This must be performed in order not to lose dynamic information, which can be sometimes difficult at this level of abstraction.

4.2 PSM With Aspects

Once we decide to also have aspects in our platform specific model, we find two alternatives. The first one is to remain modelling them in a general form, in order to define them in a specific aspect-oriented language in a more refined PSM; the second option is to proceed to the last step straight away.

If the aspects are in the PSM, but they have not been linked to a specific language, we have a more complete model which allows us to maintain our model concerns, which crosscut the system, in a modularized way.

In any of the cases, extra-functional properties may be maintained in the PSM, therefore facilitating traceability and keeping our system well modularized. This fact provides us with the possibility of deciding if we want to use an aspect-oriented language to implement these properties at a future stage or not, depending of the appropriateness of the specific case study in the particular platform. The longer our extra-functional properties are kept modelled in a separate element, the easier it will be to code them in a modularized way, also opening the possibility of reusing them in different parts of the application.

Furthermore, the fact that there are aspects defined in this model in general terms offers us the possibility of using different tools for code generation, in order to obtain the target application for different aspect-oriented languages, thus offering great reusability and a wide range of target applications.

Moreover, in the case of the composition interaction logic modelling with aspects, we can defend the same viewpoint. If we maintain the interaction logic encapsulated in the specific model, it will offer us the possibility of implementing a more modularized application, in which we do not need to couple the different services composed from the earlier design stages. Besides, we do not have problems with traceability as the line between the code and all design models can be traced perfectly.

4.3 PSM Comparative Report and Discussion

We have made a comparative table of the different elements we can find when modelling Web Services within aspect-oriented techniques in three different models: the model that is independent from the platform (PIM), the one that is specific to the platform doing the weaving of the aspects in the transformation (PSM without Aspects) and the model specific to the platform when the aspects have not been weaved at transformation (PSM With Aspects). In the comparative report we have assumed that the transformation is done in *Java* and, when aspects appear, in *AspectJ* language.

As we can see in the table, the stereotypes defined for the main elements in Web Service modelling in the platform independent model, remain in both PSM models, just transformed into the target language, therefore obtaining the *Java Web Service Interface* and *Java Web Service Implementation* stereotypes.

For the aspect stereotypes, it is obvious that in the aspectless PSM they do not appear any more, but become behaviour which would be reflected in the sequence or interaction system diagrams, where the aspects would intercept the methods' execution to provide a new behaviour. When creating the PSM with aspects for the *AspectJ* specific target, the stereotypes are transformed into the new stereotypes defined for *AspectJ* syntax, maintaining the diagram structure.

Table 1. Comparative Table Between PIM, PSM without aspects and PSM with Aspects

Compared Element	PIM	PSM WITHOUT ASPECTS	PSM WITH ASPECTS IN ASPECTJ
WEB SERVICE INTERFACE	Represented by the stereotype <<web service interface >>	Represented by the interface oriented to target language and platform, i.e. <<java web service interface>>	Represented by the interface oriented to the target language and platform, i.e. <<java web service interface >>
JOIN POINT MODEL	*Although* not represented explicitly in the diagram, as mentioned before, it is limited to service method execution and method calls from any element in the model to a service method.	This element would not appear in this model	This element would not appear in this model.
POINTCUTS	Represented by the stereotypes << ApplyExecutionAspect>> and <<ApplyCallAspect>>	This element would become part of the model behaviour representation. It could be represented as the connection point between services and aspects	Represented by stereotypes <<Execution-Pointcut>> and <<CallPointcut>>, where the different methods' execution or invocations would be linked to <<advices>>
POINTCUT RELATED BEHAVIOUR	Represented by the stereotype <<action>>	This element would become part of the model behaviour representation (idem). It would specify aspect behaviour.	Represented by the stereotype <<advice>>, which would be linked to a specific pointcut and would indicate the new behaviour to be injected in the intercepted pointcut.
ASPECTS	Represented by the stereotype <<aspect>>	It would be transformed into part of service specification and into the relations and interaction logic between them.	Represented by the stereotype <<aspect>>, formed by <<advice>> elements, which are bound to the rest of the elements by <<ExecutionPointcut>> and <<CallPointcut>> dependences.
ASPECT-SERVICE LINK	Represented by the pointcut stereotypes <<ApplyExecutionAspect>> and <<ApplyCallAspect>>, which link the action to <<Web Service Interface>>	It would be transformed into part of the behaviours services specification and into the relations and interaction logic between the named services	Represented by the pointcut[s] stereotypes <<ExecutionPointcut>> and <<CallPointcut>>

5. Discussion and Future Work

This paper has shown that we can model Web Services and their compositions by defining new stereotypes in UML. In this sense, we have defined the new necessary stereotypes for aspect inclusion at this stage of the development, in order to maintain our models well designed and our system well modularized.

Once demonstrated the importance of encapsulating Web Service composition code and the usefulness of AOP for this task, we affirm is time to face how to model it whilst maintaining the separation of concerns. As we have seen, our proposal permits Web Service modelling with a standard modelling language such as UML, by defining the necessary stereotypes, as well as aspect ones, regardless of the platform mode. Hence, it has been demonstrated that aspects can be easily modelled in service-oriented systems as extra-functional properties, and composition interaction patterns can also be encapsulated in aspects since modelling stage within the same domain.

We have also proposed two alternatives for the platform specific models, where we could choose between performing the aspects' weaving before or after we create the PSM. In this respect, there is still a lot to be done. As we have previously explained, when the weaving has been done at transformation the PSM is simpler than the one in which aspects remain. Once weaving is finished the aspects are reflected in behaviour diagrams, be it the sequence or the interaction diagram. However, in the PSM where the aspects' weaving has not been done yet it is not that simple as we have to maintain all the elements for the aspect classes and association description, but provides the privilege of allowing perfect traceability between the different models, although they may have undergone transformation. We leave this matter open for discussion in the workshop forum, in order to analyze the advantages and shortcoming of both proposals more in depth.

Once agreed on the best option for weaving, we still have to define all the process rules for automatic transformation from platform-independent into specific models. Transformation rules for translating from the specific model to the target code also have to be analysed. These are important parts of our work in the near future.

Acknowledgments: This work has been developed thanks to the support of CICYT under contract TIC2002-04309-C02-01.

7. References

[1] Kiczales, G. *Aspect-Oriented Programming*, ECOOP'97 Conference proceedings, Jyväskylä, Finland, June 1997.

[2] Ortiz, G., Hernández, J., Clemente, P. J. *Decoupling Non-Functional Properties in Web Services: an Aspect-Oriented Approach*. Workshop EOOWS, ECOOP Conference, Oslo, Norway, June 2004

[3] Ortiz G., Hernández J., Clemente, P.J.. *Web Service Orchestration and Interaction Patterns: an Aspect-Oriented Approach*, Short Papers Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC). New York, USA, November 2004.

[4] Ortiz G., Hernández, J. Clemente, P.J. *Building and Reusing Web Service Choreographies by Using Aspect-Oriented Techniques*. Proc. of the WorkShop on Best Practices and Methodologies in Service-oriented Architectures: Paving the Way to Web-services Success at the Object-Oriented programming, Systems, Languages and Applications Conference (OOPSLA), Vancouver, Canada, October 2004

[5] Aldawud, O., Elrad, T., Bader, A. *A UML Profile for Aspect Oriented Modeling*. OOPSLA 2001 Workshop on Aspect Oriented Programming.

[6] Stein, D., Hanenberg, S. and Rainer, U.: *A UML-based Aspect-Oriented Design Notation for AspectJ*. Proc. 1st Int. Conf. on AOSD, Enschede, The Netherlands, 2002

[7] Baniassad, E. Clarke, S. *Theme: An Approach for Aspect-Oriented Analysis and Design*. 26th Int. Conference on Software Engineer, Edinburgh, Scotland, UK, 2004

[8] France, R., Ray, I., Georg, G., Ghosh, S.. *An Aspect-Oriented Approach to Early Design Modeling*. IEEE Proceedings – Software, 2004

[9] Bordbar, B., Staikopoulos, A. *Modelling and Transforming the Behavioural aspects of Web Services.*

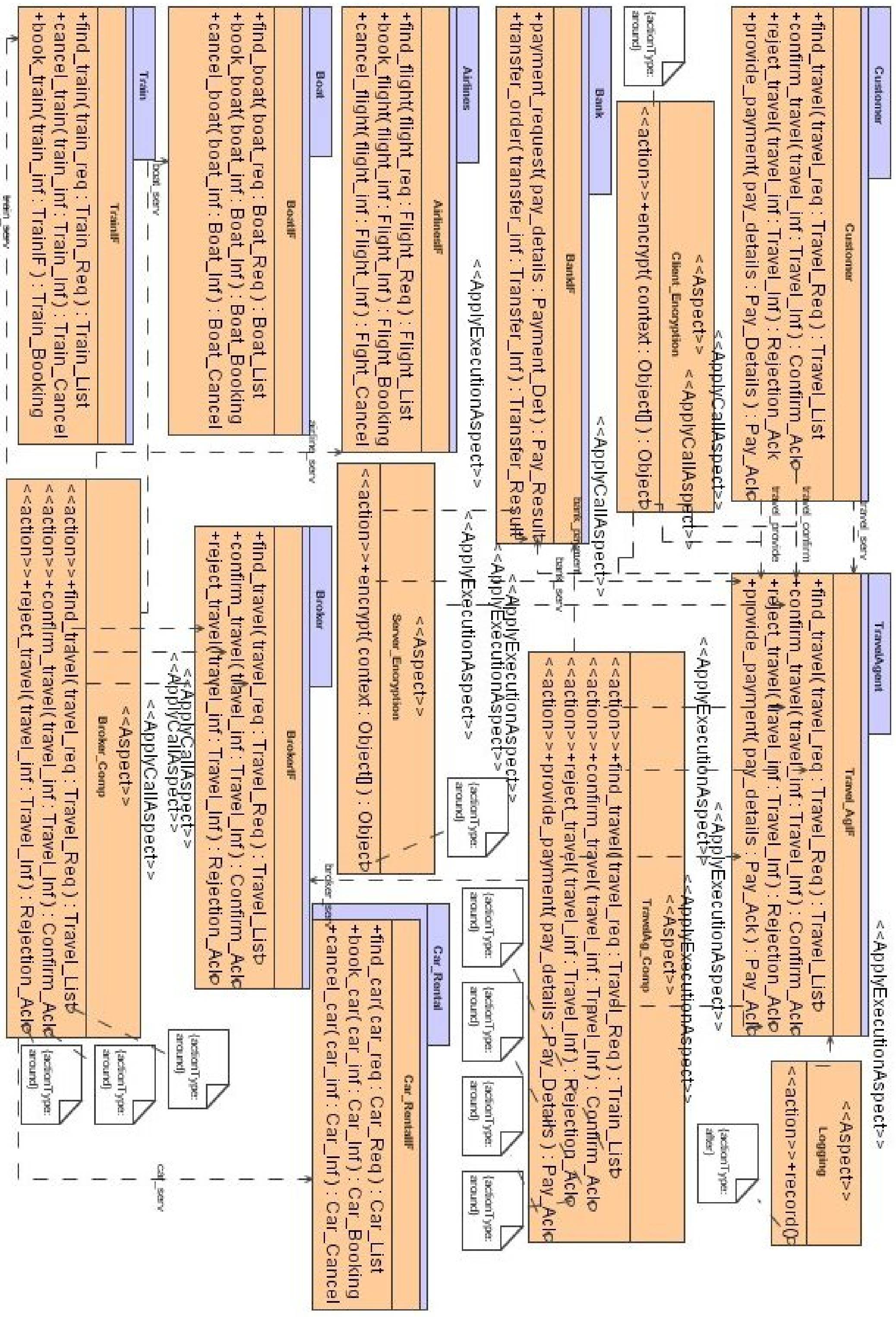
[10] Grønmo, R., Solheim, I *Towards Modeling Web Service Composition in UML.* 2nd International Workshop on Web Services: Modeling, Architecture and Infrastructure, Porto, Portugal, 2004.

[11] Thöne, S.,. Depke, R, Engels, G.. *Process-Oriented, Flexible Composition of Web Services with UML.* International Workshop on Conceptual Modeling Approaches for e-business: A Web Service Perspective, Tampere, Finland, 2002

[12] Bézivin, J., Hammoudi, S., Lopes, D. Jouault, F. An Experiment in Mapping Web Services to Implementation Platforms. N. R. I. o. Computers**:** 26, 2004

[13] Elrad, T., Aksit, M., Kitzales, G., Lieberherr, K., Ossher, H.: *Discussing Aspects of AOP*. Communications of the ACM, Vol.44, No. 10, October 2001.

Appendix . Case Study Web Service and Aspect-Based PIM



Customer
Customer
+find_travel(travel_req : Travel_Req) : Travel_List
+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
+provide_payment(pay_details : Pay_Details) : Pay_Ack
<<Aspect>>
Client_Encryption
<<action>>+encrypt(context : Object[]) : Object
Bank
BankIF
+payment_request(pay_details : Payment_Det) : Pay_Result
+transfer_order(transfer_inf : Transfer_Inf) : Transfer_Result
Airlines
AirlinesIF
+find_flight(flight_req : Flight_Req) : Flight_List
+book_flight(flight_inf : Flight_Inf) : Flight_Booking
+cancel_flight(flight_inf : Flight_Inf) : Flight_Cancel
Boat
BoatIF
+find_boat(boat_req : Boat_Req) : Boat_List
+book_boat(boat_inf : Boat_Inf) : Boat_Booking
+cancel_boat(boat_inf : Boat_Inf) : Boat_Cancel
Train
TrainIF
+find_train(train_req : Train_Req) : Train_List
+cancel_train(train_inf : Train_Inf) : Train_Cancel
+book_train(train_inf : TrainIF) : Train_Booking
TravelAgent
Travel_AgIF
+find_travel(travel_req : Travel_Req) : Travel_List
+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
+provide_payment(pay_details : Pay_Details) : Pay_Ack
<<Aspect>>
Logging
<<action>>+record()
<<Aspect>>
TravelAg_Comp
<<action>>+find_travel(travel_req : Travel_Req) : Train_List
<<action>>+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
<<action>>+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
<<action>>+provide_payment(pay_details : Pay_Details) : Pay_Ack
<<Aspect>>
Server_Encryption
<<action>>+encrypt(context : Object[]) : Object
Broker
BrokerIF
+find_travel(travel_req : Travel_Req) : Travel_List
+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
<<Aspect>>
Broker_Comp
<<action>>+find_travel(travel_req : Travel_Req) : Travel_List
<<action>>+confirm_travel(travel_inf : Travel_Inf) : Confirm_Ack
<<action>>+reject_travel(travel_inf : Travel_Inf) : Rejection_Ack
Car_Rental
Car_RentalIF
+find_car(car_req : Car_Req) : Car_List
+book_car(car_inf : Car_Inf) : Car_Booking
+cancel_car(car_inf : Car_Inf) : Car_Cancel
<<ApplyCallAspect>>
<<ApplyExecutionAspect>>
{actionType: around}
{actionType: after}
travel_serv
travel_confirm
travel_provide
bank_payment
bank_serv
airline_serv
boat_serv
train_serv
broker_serv
car_serv

Figure 7. Case Study Web Service and Aspect-Based PIM.

Integrating Web Systems Design and Business Process Modeling

Prof. Mario A. Bochicchio, Eng. Antonella Longo
SET-Lab, Dipartimento di Ingegneria dell'Innovazione - University of Lecce - Italy
mario.bochicchio@unile.it, antonella.longo@unile.it

Abstract

Business processes models and methodologies are effectively adopted by business process engineer, organization experts and bureaucrats, to describe important aspects peculiar of many modern organizations (banks, government, utilities and, generally, service organizations). The expressive power of these models is desirable to design and deliver Web Applications and Web-enabled services for these organizations. On the other hands, models and methodologies for the design of Web systems, primarily focused on software engineering and technical aspects, may fail in capturing important business aspects of the services to be delivered.

In the paper we propose a framework which extends a specific methodology for modeling Web applications with concepts of business process design, to bridge the gap between the business process modelers and the Web system designers.

1. Introduction

In last years the growth of digital communication networks and of distributed systems permitted the development of new business-to-consumer, business-to-business and government-to-citizen relationships. The current trend is to exploit the Web to provide services in such an unexplored fashion, driving organizations to rethink continuously the ways in which they do business and the type of business they do. In this scenario, an organization needs to be flexible enough so that it can cope with the complexity of the new technologies and its business systems while not disregarding all the opportunities created by internal or external changes of context.

Therefore, an organization should be aware about how it operates at both business and information system levels, and should constantly assess their reciprocal dependencies. These dependencies are twofold:

- the design of Web systems should reflect the model of business processes, where with a business process we mean a sequence of activities that take one or more inputs and create an output that gives value to a customer. Business process modeling includes the description of the structure and behavior of an organizational activity, such as process activities flow, the role of its actors, the rules actors use, the information needs actors have;
- Web systems should enable innovation in business processes and business models, providing the process customer with new services. In other words, Web technologies provide significant strategic leverage to businesses, with a strong impact on business models and consequently on business processes.

The first dependency requires a reasonably seamless modeling chain from business process modeling to web system design. The latter dependency requires Web system architects to have a clear vision of what technology can do for business.

The necessity to unify these areas is a hot research topic today [21-23, 28]; it derives from the fact that existing methodologies to design business processes are naive in modeling Web system aspects (like information, transactions, navigation patterns). Conversely current frameworks to design Web applications are still basic in treating business processes which must be supported by Web applications.

In the paper we propose a conceptual framework, referred to as UWA+, for describing, linking and tracing these concepts at multiple levels of detail merging two separate areas of concerns: business processes and Web systems. UWA+ extends UWA (Ubiquitous Web Application) conceptual framework [19], developed by a joint effort of European researchers, with concepts from the Information System research area, in order to cope with UWA's lack about the modeling of business processes. The adoption of standard models and tools (in a broad sense) from the business process research scenario facilitates to linkage between business process models and Web systems design – a characteristic that is crucial in Web-development, where the systems under development often lead to fundamental changes in business processes and models.

This paper is structured as follows. Next section provides the context of our approach; section 3 presents the requirements of the new framework; section 4 presents the main concepts of UWA+ referring to a sample application. Finally in section 5 we discuss our conclusions and work directions.

2. Background

Although business modeling has been a significant challenge for business and IS practitioners for more than one decade, relatively little has been written about how to express interdependencies among business processes, information system components and actually adopted technologies. In net-enhanced organizations these dependencies are even more stressed because the introduction of a Web system has a fundamental role on the nature of business processes and the business models which are being supported. This section provides background on business process modeling and Web system modeling and on linkages between them.

2.1 Web Information System and Web applications Modeling

More and more, information systems (IS) relies on system architecture models. The system architecture describes the relationship among components and the guidelines governing their design and evolution. MDA [28] is the main effort done in divorcing implementation details from business functions from a software engineering standpoint. Associations between business concepts and systems are usually embedded in the architecture's development process [6].
This implies that identifying which parts of business are supported by which parts of the system is not a straightforward task. One major issue that enterprises face today is ensuring IS architecture be business-driven and adaptive to changing business needs [8]. Nonetheless Web Information Systems (WIS) single components, consisting of the integration of complex functionality with rich information handling, must be designed with the focus on users in order to maintain their experience effective and satisfactory. To model these systems there are a number of elements that we would like to represent. In terms of information design, typically we wish to model not only the information itself, but also the relationship between the underlying content and the user-perceived views of that content, the interactions with those views (such as behavioral and navigational aspects), and the ways in which the information is represented and presented to users. This modelling tends to be much more complex than traditional "data modelling" (e.g. E-R models and data flows). Whilst existing modeling languages (such as UML) can be used to represent the functional aspects, they are not so effective to represent these informational aspects. Although some attempts have been made to adapt UML to support information models (e.g. the interesting work by Conallen [7]), these are still relatively simplistic and suffer from a notational confusion. For example classes, normally adopted to represents data structures and related methods, are now proposed to represent information elements and/or other modelling constructs, that is incorrect or inconsistent [24].
Actually the problem to explicitly include business processes in Web Systems design, treating them as "first class citizens" along with navigational and informational aspects, is emerging insistently [20, 21]. Usually these approaches are based on the definition of business processes which reminds the concept of database transactions; in [25] business processes are regarded as heavy-weighted flows of control consisting of activities and transitions. The definition adds functional aspects to informational and navigational design and it treats business processes as a single user perspective's workflow related to a chosen interaction channel. On the other hand business processes, as defined in Organizations and Information Systems literature [1-6], are a sequence of activities that take one or more inputs and create an output that gives value to a customer. This definition involves several user types with different points of view, each accomplishing different activities with specific communication style. In our experience processes are fundamental for the design of WIS in complex organizations, because they add contextual information to the following design of operations and business transactions. Hence we consider business process design an essential portion of WIS requirement analysis. More recently a number of approaches has been developed to design Web applications that utilize and adapt software modelling design, and in particular UML [12-15]. Of particular interest in this paper, as an example of these modeling approaches, is UWA (Ubiquitous Web Applications) conceptual framework [19]. UWA is a requirements-driven, user - focused approach, incorporating a graphical notation based on UML. Whilst these kinds of framework (like many other modern approaches) claim to address the full development spectrum – from the requirements to the detailed design of websites -, we contend that their focus is primarily on the design and development of hypermedia aspects (like for navigational sites, which mainly allow access to huge amount of information) of Web applications. Moreover in the requirement elicitation step UWA's authors describe organization context through goal-stakeholders diagrams and textual constraints. This approach is acceptable in the design of small Web systems, but it is less effective for the conceptual modeling of Web applications to be set up in complex organizations (like banks, e-government services, utilities, …), where more formal contemplation of the scenario (made up of organizational structures, processes, constraints, rules and norms) is required. Thus models like UWA's requirement elicitation, describing the organization by semiformal rules and textual constrains need to be improved.

2.2 Business Modeling: Processes and Goals

In current competitive global economy, the demand of high quality, low costs and fast delivered products is forcing organizations to become process-focused in order to maximize the performance of their value chain and business process engineering.

Countless research studies have underlined that in designing a new process, one should investigate the interaction between the "systems" that drive business performance and condition organizational change. Business and organizational view and systems view have been variously integrated ([2], [3], [4]). In particular with the advent of Business Process Reengineering (BPR) the necessity of a systemic view has required analysts to document, understand, manage and efficiently model business processes. Numberless modelling languages describe at various extents these aspects of processes (e.g. IDEF0 highlights activities and DFD is centered on the information flow). However, their stereotypes and their abstract types are rather far from the one used to model web systems. Bridging the gap between these business and systems models is a hard task, because they use proprietary types and lack systems modelling notations and concepts necessary to conceptual modelling Web systems.

On the other side UML provides a standard, non-proprietary, general purpose modelling language, currently used to model business processes.

There are currently two mainstream directions to extend UML for business modeling in order to bridge the gap with the system design. One direction is proposed and standardized into the UML by OMG [5]. This approach tends to preserve existing UML diagram elements, duplicating the user view layer and the structural view layer into the business modelling domain and broadening the scope to the business analysis domain. This results into an approach which is comfortable for system analysts, but it is rarely used by business analysts. The latter approach, Eriksson-Penker's Business Process Extended Modelling (BPEM) [6], is wider and more process oriented, but twists somehow the UML pre-ordered arrangement and diagrams to adapt them better to the way business analysts think about processes and business process redesign. By this extension one can describe both statically and dynamically a variety of process elements. BPEM also proposes a new diagram, the assembly line diagram, which is the core tool to bridge the business process analysis and IS design. Through assembly line diagrams the analyst captures both the domain of functionality and the domain of information and connects the two through use cases in object-oriented modeling. The relationship between assembly line packages (describing the information aspect) and swimlanes (describing the functional part) shows the information necessary to each user type during the process activities, while the reference to the assembly line packages comprises the information flow to and from the information system.

Even if this view is very helpful to link informational resources to the user type through the swimlanes, in general this functional approach, appropriate for some traditional information systems, is not valid] in Web system design because

- Informational resources are modeled through classes, which are a construct normally associated with functional elements and usually used in the system detailed design. They show little emphasis on the model of information structure, which is very complex in WIS design as in all hypermedia applications.
- Web systems handle richer information than traditional data systems, with additional informational navigational issues to the traditional business process workflow;
- Assembly lines link process activities to their required informational resources, not showing any semantic association among the informational resources. This last feature provides the "infrastructure" for potential content navigation;
- By its nature and purpose conceptual modeling of hypermedia applications is different from business process modeling and, in particular, navigational aspects cannot be straight derived from business process models, without considering the user experience and other communication aspects ;
- Business informational resources are at higher level of abstraction than information objects composing software systems and a straight match to software components or classes is too coarse.
- The relationship between business goals and business processes are expressed in a semiformal language with no reference to systemic properties [17], like process performances or properties concerning the impact of Web technologies on business goals and processes.

An interesting attempt to exceed this lack is PIM (Process Information Model), a further extension of Erikkson-Penker's BPEM, developed at the Business Engineering of Politecnico of Milan. The systemic aspect [17] is modeled through Key Performance Indicators (KPI) a concept used in management landmark, the Balanced Scorecard [1]. In this model, generally used in BPR projects, business goals are quantified, planned and controlled through Business Process KPIs, which measure both internal performances and the performance of customers from different standpoints - quality levels, service level, efficiency and cost. Therefore, the analyst selects the appropriate KPIs to describe the performance of the process being modeled, and uses these KPIs to benchmark the actual performance and/or the performance expected by management and/or the best practice.

In net-enhanced organizations, where the link between business goals, business models and Web systems models is even more crucial than in the past, KPIs are fundamental to satisfy the user expectations, to monitor business performances and to trace requirements and changes from goals to Web systems and viceversa.

3. Integrating Web Systems Design and Business Process Design: Requirements

The background, depicted in the previous section, shows that existing methodologies to design business processes are naive in modeling Web application aspects (like information, transactions, navigation patterns etc.). Conversely current frameworks to design Web applications are still basic in treating business processes and goals. A conceptual framework integrating these views is not just the union of two tasks performed in isolation. It sums up the complexity of the two activities with the additional task of designing their coupling. As such, extending conceptual modeling of web applications cannot simply pile up existing techniques of hypermedia design (borrowed from the hypermedia/web communities) with methods and notations of "traditional" business process modeling (borrowed from the information systems or organization communities). The crucial point is to integrate and extend models, design methodologies and techniques to meet the new design challenge. Given stakeholders goals, defined in order to exploit the potential of Web technologies, the integration aims at providing a formal way of design processes and Web information systems and the relationships between them.

Although the new framework is based on a clear separation of concerns (business processes designers and Web systems designers), necessary to handle the complexity of the Web System design, the dependencies and relationships between the different views are evident. The research literature and the previous discussion has shown that a conceptual framework aiming at bridging the gap between the two perspectives requires the following properties:

Req. 1. **Ability to design business processes "per se"** (i.e. not just as artifacts to design the corresponding Web sites, like dynamic diagrams in UML). Business processes are essential to describe several key aspects of a given business context: they are useful to define the key performance indicators (KPI), to optimize the workflow, to solve possible conflicts etc. Therefore, they are not just design artifact to better define some navigation step or some page details of the Web site supporting that process. Business processes have the responsibility to support all the key aspects of the organization's business model. These crucial business functions can be implemented either through Web interfaces to existing legacy applications, or through new Web-based systems. Therefore the new framework, must enable the representation of these functions and the related design artifacts. Examples include the modelling of business workflows, information and order tracking, transaction processing etc.

Req. 2. **Traceability between the business process model and the WIS model**. The traceability problem can be expressed as an extension of the same issue in software engineering. Paraphrasing Palmer's assertion [16] the traceability between processes and WIS gives essential assistance in understanding the relationships existing within and across Web system requirements and design, and within and across the business process modeling. Therefore, for example, all transactions and operations on Web pages should be mapped to the corresponding business process (or subprocess, or activity); the relationships between workflows and navigation structures should be captured and the correspondence between business processes models and Web information models should be established. By definition, net-enhanced organizations are constantly evolving, so that the traceability requirement is essential to continuously evolving the model, adapting it (and the related Web System) to the changing context, without restarting a whole new design/implementation cycle. The requirement is that the framework must support the modelling of both information and functional architecture and, more important, their integration in a cohesive and consistent manner. Once understood and documented, the business process model needs to be effectively mapped and integrated into a Web Systems design, enabling the implementation and delivering of the system functionality. To support this requirement, the framework must provide the ability to identify the linkage between the business model and the technical aspects, and between the model elements in the business model and the model elements in the technical architecture. This interconnection needs to be represented at various abstraction levels.

Req. 3. **Skill hiding**. Business-related development and modelling artifacts are usually created and used by developers from both IT and business backgrounds. As a result, the modelling of business domain concepts must be designed considering a wide range of users; in this way the modelling artifacts can be easily understood, communicated and modified within and across development teams and business units. It can be very helpful to customize the modelling artifacts from different perspectives, so that concepts and methods are used by people with different background. Paraphrasing the well–know "information hiding" concept, the new framework must enable teams with homogeneous skills (i.e. process designers, Web system designers) to encapsulate their work, with an interface that only provides access to concepts and diagrams useful to the other teams. The requirement aims at allowing Web system engineers and business analysts to work together in a common environment, each preserving the specific focus on either processes or Web system design respectively. In this way the communication gap between business process and Web systems designers is bridged. The skill hiding requirement is also important because in a continuously changing scenario, like that of Web Information Systems, both process modelers and Web application designers need to manage the "in-progress" aspects, and to continuously review the design, each from his/her own perspective.

Req. 4. **A user centered rationale**. The common trend in Web Systems approaches is "user-centered". Instead, business designers usually define their methodologies, and handle the supporting Web Information Systems design, according to a "process-based" approach. Actually the definition of Web Information System is at higher level of abstraction than Web application and the characterization of the first as business driven doesn't exclude the latter to be user centered. Moreover the definition of the process focuses on the characterization of its customer. Hence, to support this requirement, the methodology must be based on a user centered rationale.

Req. 5. **A unique modelling language**. Several modern Web design methodologies adhere to the semantics and notation provided by UML. The scenario is less uniform in business process modelling, even if the use of UML is already increasing. UML is also the standard modelling language in both research and industry environments and it is already applied to different domains through its extensibility mechanism. In this perspective, the new framework must be based on UML, assuming that business process aspects and Web system aspects must be supported with the proper notation and tools.

4. UWA+

In order to clarify the enhancement of UWA+, the following section provides a summary of UWA framework.

4.1 UWA framework in a nutshell

UWA framework (the right hand side of Figure 1 without the integration design) organizes the web application design process into four activities:

- *Requirements Elicitation:* defining stakeholders of the application (anyone who has interest in the application and not just application users), their goal and their requirements.
- *Hypermedia and Operation Design:* this activity is accomplished by W2000 methodology; it is composed by the *Information Design* (structuring contents of the application and their organization in terms of access structures), the *Navigation Design* (defining navigation paths in the contents of the application in order to provide effective navigation), the *Publishing Design* (defining how the application will be organized into pages) and
- *Transaction Design*: defining user activities, system transactions and how operations are involved in them.
- *Customisation design:* defining customisation rules to enable ubiquity of the application access (anytime/anywhere/anymedia) by adapting a web application towards a particular context which reflects the environment the application is running in.

If *Hypermedia and Operation Design* activities are "typical" of Hypermedia design, *Requirements Elicitation, Transaction Design* and *Customisation design are* new ones and their definition and integration with the W2000 methodology is one of the main contributions of UWA. Using the concepts and the corresponding notations, a designer can write down the *application schema*, i.e. a description of what the application will be (not how it will be implemented). The application schema should be the result of the *design activity* that takes into consideration the overall *application requirements* (both typical business requirements and requirements about integration with legacy systems requirements).
For each design activity, schemas can be specified at two levels of depth and precision: *in-the-large*, to describe general aspects, sometimes informally, without many details, and *in-the-small,* to describe the several aspects in fine details.

4.2 Extending UWA with Business Processes

The requirements depicted in the previous sections are included in the framework briefly presented in this section. UWA+ (Figure 1) extends the UWA framework [19] with models for business process design coming (Req. 1) from PIM methodology [17].
The choice has been guided by the characteristics of these methodologies, which match some requirements of those described in the previous section.

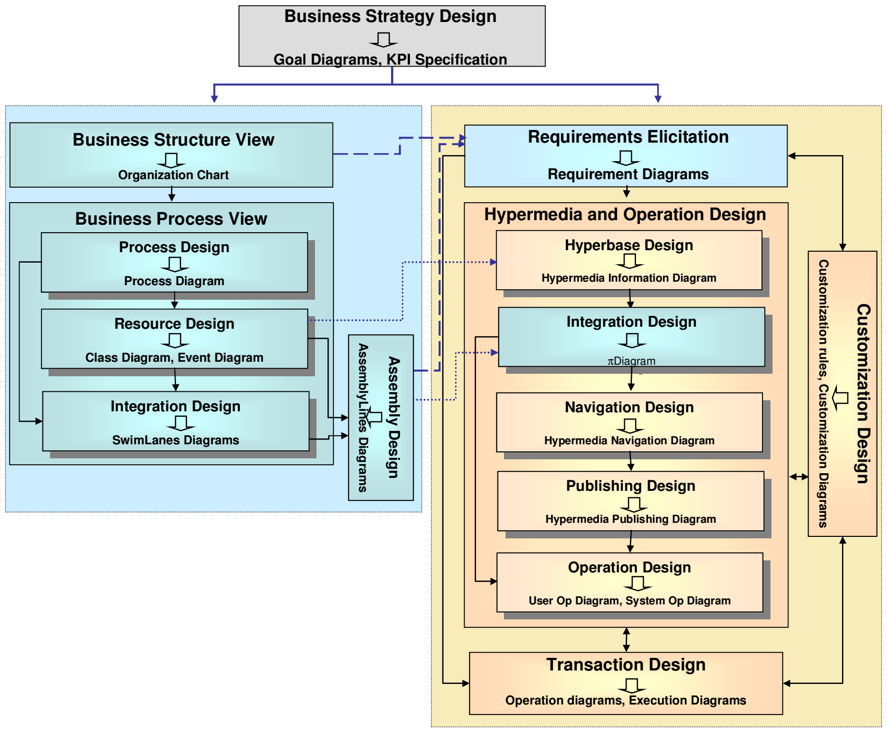


Figure 1: UWA+ framework

We adopted UWA framework among other "hypermedia-based" conceptual models, because of both its strength in user-centered modeling (Req. 4) and its UML-based notation (Req. 5). On the other side PIM is a process–driven methodology (Req. 4), based on UML primitives (Req. 5). Both these approaches have been successfully tested in real life projects, and they are, at the moment, some of the most convincing frameworks for designing web-applications and Business processes.
In agreement with Req. 4, UWA+ has the common rationale in the user satisfaction/process focus, but the conceptual component that models the process performances, and drives a part of the requirement design of the Web system, is based on KPIs (Key Performance Indicators). It helps in improving the overall performance of the process and, simultaneously, complies with the objectives of users.
UWA+ is focused on net-enhanced organizations with a systemic approach, so it assumes goals and KPIs are defined considering the use of Web technologies. The methodology supposes a heterogeneous teamwork of business-process analysts and Web-application engineers collaborating on complementary parts of the same framework and sharing some relevant aspects (Req. 2).
The starting point of the overall approach is the definition of strategic goals and KPIs, applicable to the process and to the customer to be achieved by the support of the Web system. In UWA+ we distinguish between procedural and communication goals, which correspond to organization’s critical success factors. Procedural goals describe time dependencies and logical steps, in a process-driven fashion, while communication goals define the user experience and the content aims.
In order to integrate the two perspectives, a process engineer must provide the Web system designer with models describing the organization where the application is going to be set up, namely the organizational chart, business process models (showing the process flow, input and output, actors, constraints), models of resources and structured constraints drawn during the organizational analysis. Special diagrams, like swimlanes and assembly lines [6], are very effective to define the scenario and are a useful input to the requirement elicitation of the Web system. Conversely these models are used to trace potential changes of the Web system requirements back to the business process perspective. The conceptual modelling of the Web system follows the main steps of UWA, with some integration described in the following.
The elicitation of the Web system requirements starts from the identification of stakeholders, goals, i.e., objectives that the application must satisfy in the stakeholder desires, and scenarios. Through the goal-stakeholder diagram goals are linked to the corresponding stakeholders, who are also responsible of their

performances. In particular procedural goals, together with swimlanes diagrams, elicit operation and transaction requirements, while Contents, Structure-of-Content, Navigation, Presentation, Customization requirements derive from communication goals derive . KPIs, linked to both the types of goals in the Business Strategy Design, are inherited by corresponding requirements. This allows traceability of goals and monitoring of performances till the We system requirements.
Swimlanes, assembly lines diagrams and organization chart contribute to the identification of the Web System's stakeholders. In particular swimlanes and assembly lines diagrams define internal users (Personal Travel Assistant, Broker Agent, etc), customer profiles and external organizations dealing with the Web system, while the Organization chart helps to identify stakeholders who are hierarchically superior to the system's internal users and interested in the system for strategic and advisory purposes. The stakeholders so located are a subset of those a requirement engineer must take into account. The relationships among goals, stakeholders, process structure and system requirements enable the traceability (Req. 2) among business strategy, business processes and Web system architecture.
To model goals and derived sub-goals coming from the business process analysis we use an arrow shape, whose content is the process phase at the top and the child's activity at the bottom, while communication goals are still represented by oval shapes.
Let's use as running example the process followed by a Travel Agency for selling travels to its customers, where we consider the point of view of the Personal Travel Assistant. In the macro-process we can single out two business processes: the purchase of the travel by the customer and the monthly payment of the external Broker Agents. Let's consider the first process.
Figure 2 shows a partial view of the goal/stakeholders diagram. It represents the goals of the Customer who wants to "Buy a travel", which corresponds to the goal of the Personal Travel Assistant to "Sell a travel". The Customer also wants to "Get Information about the arrival city" and "Access his/her previous trips".
From the customer's perspective, the "Buy a travel" goal can be specialized into a few sub-goals: "Submit a Request" goal, "Refine/Update request", "Choose among travel alternatives". Since these sub-goals derive from the hierarchical decomposition of the process activities, their shape is the arrow The label in the upper part of the diagram is the process to which the activity belongs.
These subgoals are also the specialization of Personal Travel Assistant's "Sell a travel" goal, together with the "Selection of the broker", the "Aarrangement of the trip" and the "Presentation of the different offers" to the Customer. The "Presentation of the different offers" subgoal can be "operationalized" into requirements: the description of each single alternative and the sort of the offers according to the customer's requests.

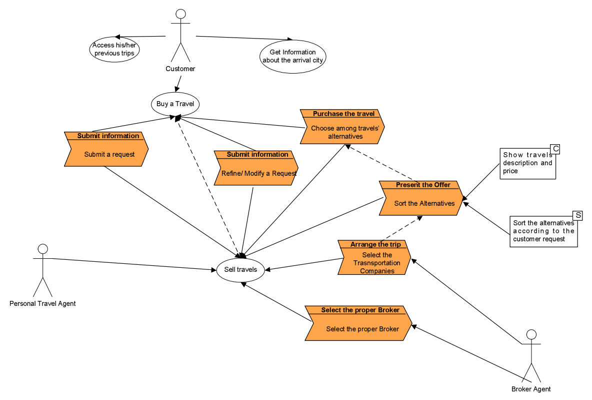


Figure 2: Goal / Stakeholder diagram

The shape used for the requirements is a rectangle with a label indicating its nature (in our sample they are Content and System requirements, respectively).
It is evident that the whole set of requirements to design the Personal Travel Assistant Web system comes both from the specialization of the business processes into transactions and operations and from content and navigation features, derived from the context and from other interviews wit the users.
Resources diagrams and assembly lines diagrams are input of the Information design, where the main purpose is the identification of the relevant information to be handled by the application, and the provision of an overall organization of the information structures, independently from any specific intended usage. Assembly line packages [6] are evaluated for the design of entity types in the hyperbase in the large [19], while Resources Diagrams [6] are an input of the hyperbase in the small. The hyperbase model is designed at a high level of abstraction and it is independent from the single user perspective and the specific interaction channel. This lets the model be persistent along the time and the different user views.
The main enhancement of UWA+ towards business process methodologies is the step of the Integration Design. The aim of this step is to correlate the process structure model and the information model, in order to have the overview of actors, processes and information resources involved in the design of the Web system. The output of this phase is the production of the πDiagram (Process Information Diagram), which bridges Business process and Information. The diagram represents the swimlane referring to a process phase at the top and the hyperbase in the large diagram at the bottom part. πDiagram shows the semantic relationships among informational resources and between them and the related process activities.

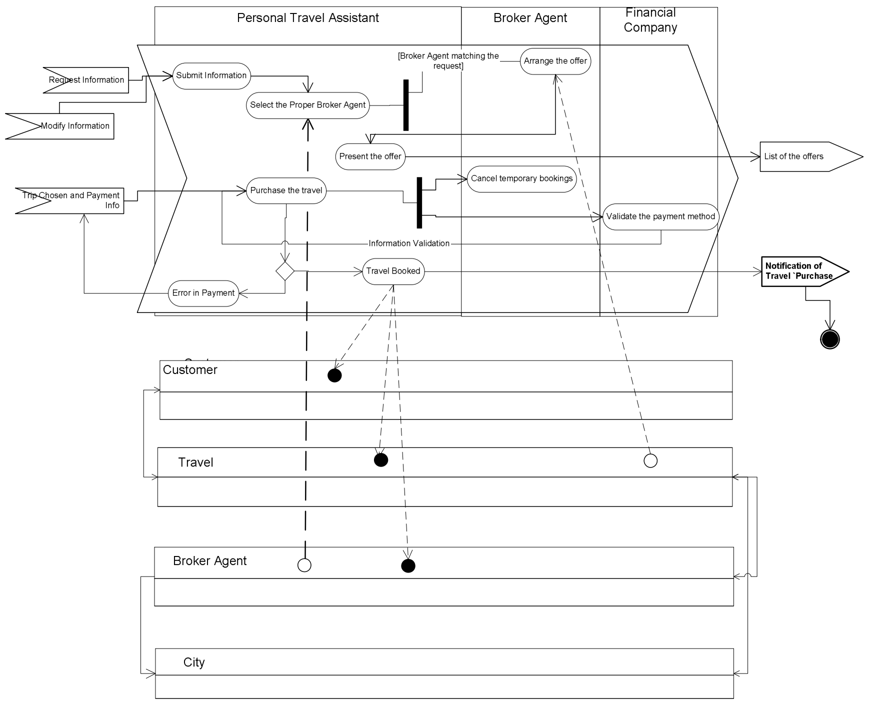


Figure 3: πDiagram

It outlines the potential informational navigation and business process execution patterns to be implemented by the Web system. The connections between the swimlanes and the hyperbase comprise the information flow to and from the information system and show the interface between the business process and the Web information system. As a rule of thumb the transaction design uses the upper part of the diagram to design WIS transactions, while the relationships between the swimlane diagram and the hyperbase represent the operations the Web System must deliver. πDiagram is also crucial to trace the dependencies of processes and information resources down to the transactions and operation design steps.

In our Travel Agency example Figure 3 shows the πDiagram representing the business process of selling a travel to a customer and the corresponding candidate information entity types. The upper part of the πDiagram shows the process and the swimlanes representing the involved stakeholders (the Customer, the Personal Travel Assistant of the Travel Agency, the Broker Agent -which can be internal or external to the Travel Agency-, the Financial Companies) and the corresponding activities.

When the Customer needs to buy a trip, he/she enters the travel agency and describes it, sometimes establishing constraints and conditions.

The Travel Agency's Personal Travel Assistant receives the requests from the Customer, checks that it is well formed, and selects the Broker Agents that could work with them. The Personal Travel Assistant interacts with each Broker Agent, asking for an offer that fulfils the Customer's requested trip. Each Broker Agent asks the Transportation Companies to provide an offer for the requested service. If the offer matches the customer requirements, the Broker Agent will pass the offer (with the corresponding overhead in case of external Brokers) to the Personal Travel Assistant of the Travel Agency. 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 the sorted list to the Customer. The Customer may either select one of the offered trips, reject them all and quit, or refine requirements and start the process again. If the Customer selects one of the arranged trips, the Customer will provide the credit card details to the Personal Travel Assistant, who will process the payment through the corresponding Financial Company. Once the payment is correct, the Personal Travel Assistant will notify the corresponding Broker Agent to confirm the booking(s). 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 to cancel their bookings.

The bottom part of the πDiagram shows the information packages and the candidate semantic relationships among them. In our sample the Customer package includes

general information about the Customer (name, address, etc.) his/her preferences, his/her purchase history, the broker agent package consists of information about his/her specialty, his/her overhead, the Cities package says about the airports and other travel facilities, the Travel package expresses information about the specific trip of a customer from the departure city to the arrival town, offered by a Broker Agent.
For clearness of the diagram we show only a few relationships between the swimlanes and the information packages; the most suitable Broker Agent for a given travel is derived from the Broker Agent descriptions stored in the corresponding information package; once the payment is accepted the travel information and the customer information is stored in the corresponding information packages.

Navigation, Publishing, Operation and Transaction Design are similar to UWA's framework. They gather inputs and information from the requirement elicitation and the information design.

UWA+ is more efficient and effective with respect to UWA because:

- The modeling of business processes has reduced the time of requirement elicitation to define the Web systems requirements
- The documentation quality has been improved due to the use of a common modeling language based on UML
- The design documentation has been reduced and harmonized between the business process and the Web system designers because of the use of a common framework
- Web system and documentation maintenance is easier due to the traceability between the two views

As shown, in UWA+ the traceability requirement (Req. 2) between the two views is satisfied, thanks to the formal definition of the steps and the integration diagrams existing between the views.
Moreover business process design and Web systems design remain separated and are made by people with different backgrounds, according to the "skill hiding" requirement (Req.3).

5. Conclusion and Further Works

The framework proposed in this paper aims at solving some of the problems haunting the relationship between business process modeling and Web systems design. Having a unique and standard language to describe different aspects of business and systems is fundamental to create a common ground for discussing both business and the supporting Web system. The emphasis of UWA+ is on providing the basis for creating such a common representation and simultaneously providing a way for addressing the traceability between the different views. The first view of the framework is about business goals and business processes. The relations between the process and Web system view allow the representation of how Web information systems support business, which is one of the main issues in today's organizations. Web system modeling is based on a user centered approach, which is the cornerstone of today's Web system architecture.
UWA+ overcomes the main shortcomings of the parent methodologies, PIM and UWA. In fact PIM is very effective in designing business processes, it is less successful in the steps toward the design of the information system. For instance, it designs information systems from business processes models and models informational resources as classes, using activity diagrams to model the behavioral aspect of business processes. This approach is information systems -oriented, whilst the key viewpoint should be process oriented. On the other hand in the requirement elicitation step UWA's authors describe organization context through goal-stakeholders diagrams and textual constraints. This approach is acceptable in the design of small Web systems, but the conceptual modeling of Web systems in complex organizations (like banks, e-government services, utilities, …) require the contemplation of the scenario (made up of organizational structures, processes, constraints, rules and norms), thus models like UWA's requirement elicitation, describing the organization by semiformal rules and textual constrains are little effective. In one word the first flattens the user communication and interaction design, the latter needs to extend the requirement to hook up the organization.
UWA+ framework integrates these views and manage the complexity of the two activities with the additional task of designing their coupling.
Future lines of work are planned to extend the framework for better capturing the evolving needs of real organization into evolving business models, and corresponding development process that leads from business to systems (of which the matching between different layers is a starting point).

Acknowledgments

The framework presented in this paper is partially funded by a grant of the Italian Ministry of Education and Research (VICE project).

6. References

[1] Kaplan R.S., Norton D.P., Balanced Scorecard, HBS Press, Boston, 1996

[2] IBM, Business Systems Planning, GE 20-0257-1, 1975

[3] Martin J., Information Systems Engineering, Prentice Hall, 1990

[4] Scheer A.-W., ARIS - Business Process Modelling, Springer, 2000

[5] Object Management Group, UML Extensions for Business Modelling, v.1.1, September 1997

[6] Eriksson H-E., Penker M., Business Modelling with UML: Business Patterns at Work, Addison Wesley, 1999

[7] J. Conallen. Building Web Applications with UML. Addison Wesley Object Technology Series. Addison-Wesley, 2nd edition, 2003.

[8] Sims, D., "EA Best Practices", 2000. http://www.eacommunity.com/articles/art28.asp

[9] P. Dolog and M. Bieliková, "Hypermedia Modelling Using UML," presented at ISM 2002 - Information Systems Modelling - ISM'2002, Czech Republic, 2002.

[10] L. Mandel, N. Koch, and C. Maier, "Extending UML to Model Hypermedia and Distributed Systems," 1999

[11] B. Henderson-Sellers, A. J. H. Simons, and H. Younessi, The OPEN Toolbok of Techniques. UK: Addison-Wesley, 1998.

[12] L. Baresi, F. Garzotto, and P. Paolini, "Extending UML for Modeling Web Applications," presented at 34th Hawaii International Conference on System Sciences, Hawaii, USA, 2001.

[13] J. Conallen, Building Web Applications with UML. Reading, Mass: Addison-Wesley, 1999

[14] R. Hennicker and N. Koch, "Systematic Design of Web Applications with UML," in Unified Modeling Language: Systems Analysis, Design and Development Issues, K. Siau and T. Halpin, Eds.: IDEA Group Publishing, 2001.

[15] N. Koch and A. Kraus, "The expressive Power of UML-based Web Engineering," presented at Second International Workshop on Web-oriented Software Technology (IWWOST2), Malaga, Spain, 2002

[16] J. D. Palmer. Traceability. In R. H. Thayer and M. Dorfman, editors, Software Requirements Engineering, Second Edition, pages 412.422. IEEE Computer Society Press, 2000.

[17] R. Vidgen, D. Avison, B. Wood, T. Wood-Harper, Developing Web Information Systems, 2002 Elsevier Science

[18] G. Rossi, H. A. Schmid, F. Lyardet: Customizing Business Processes in Web Applications. EC-Web 2003: 359-368

[19] http://www.uwaproject.org/

[20] L. Baresi, F. Garzotto, P. Paolini, "From Web Sites to Web Applications: News Issues for Conceptual Modeling", Proc. Int'l Workshop on the World Wide Web and Conceptual Modeling, LNCS 1921, S.W. Liddle, H.C. Mayr, B. Thalheim, eds., Springer, 2000, pp. 89-100

[21] H. A. Schmidt, G. Rossi, "Modeling and Designing Processes in E-Commerce Applications", IEEE Internet Computing, Vol. 8, Number 1, 2004, pp. 19-27

[22] W. Lam, V. Shankararaman, "An Enterprise Integration Metholodogy", IEEE IT Professional, March/ April 2004, pp. 40-48

[23] AA. VV. "Business processes on the Web", Special Issue of IEEE Internet Computing, Vol. 8, Number 1, 2004, pp. 28-54

[24] Tongrungrojana, R. and Lowe, D., 2003, WebML+: connecting business models to information designs, SEKE: Fifteenth International Conference on Software Engineering and Knowledge Engineering, (San Francisco, USA, 2003), Knowledge Systems Institute, Skokie, IL, USA, 17-24

[25] Koch N., Kraus A., Cachero C., Meliá S.,. Modelling Web Business Processes with OO-H and UWE. In Third International Workshop on Web-oriented Software Technology (IWWOST03). D. Schwabe, O. Pastor, G. Rossi, and L. Olsina, editors, 27-50, July 2003

[26] Longo A., Bochicchio M. A., "UWA+: bridging Web systems design and Business process modeling", in Web Engineering Workshop, Hypertext '04, Santa Cruz, August '04

[27] Longo A., Bochicchio M. A., "Public-Private Partnership to manage Local Taxes: Information Models and Software Tools", in EGOV 2002, Aix-en-Provence, sept. 2002

[28] Booch G., Brown A., Sridhar I., Rumbaugh J. Selic B.: "An MDA Manifesto", MDA Journal (www.bptrends.com), May 2004

Incorporating Cooperative Portlets in Web Application Development

Nathalie Moreno, José Raúl Romero and Antonio Vallecillo
Dpto. de Lenguajes y Ciencias de la Computación
Universidad de Málaga, Spain
{vergara,jrromero,av}@lcc.uma.es

Abstract

Portlets-based software development is gaining recognition as a key technology for the construction of robust and evolvable Web applications. Emerging portlet standards like JSR-168 or WSRP—along with commercial development tools and portal servers—enable designers to quickly develop, integrate and run "composite" Web portals, which integrate specific platforms and partners. Despite this significant progress, there is a lack of guidelines and models for addressing the development of portlet-based Web applications from a technology-independent viewpoint. This work tries to identify the main concerns involved in modeling portlets, determine the major models required to capture these concerns, and propose a way to address their integration into Web applications, from a platform-independent point of view. In so doing, we also introduce mechanisms for modeling inter-portlet communication and cooperation capabilities independently of the target portal server product.

1. Introduction

A current tend in the development of distributed Web applications is to reuse and assemble pre-produced components—such as Web services or portlets—for reducing development costs and improving software quality. The assembly of these third party systems for building Web applications has been successfully applied in practice. In fact, Web services are commonly used as the main building blocks for generating Web applications and portals.

One of the limitations of Web services is that they focus on the service *functionality*, but without dealing with presentation issues. This forces the re-creation of the presentation layer in each client application that uses the Web service. Besides, the corporative image and many of the marketing aspects of the service are lost if presentation is not considered, something which is very important for some service providers (their brand name is a key factor for their business—think for instance for Adobe, IBM, or Coca-Cola, whose corporate image and logo are crucial for selling their products and services). To overcome this limitation, *portlets* provide integration in both the business logic and the presentation layer allowing end-users to interact directly with the service.

From an application perspective, a portlet is *an individual Web-based component that typically handles requests and generates only a dynamic fragment of the total markup that a user sees from his or her browser* [3]. The content of a portlet is normally aggregated with the content of other portlets to form the final portal page. That is the reason why portlets are rarely run in an isolated way, but together with other portlets. However, when a user navigates within one portlet, the others usually remain unchanged ignoring what is being rendered by it. In order to transfer data from one portlet to another, users have to manually copy and paste key data from sources to targets portlets. This means that each portlet has to be searched individually for relevant information.

The need for effectively modeling, integrating, communicating and sharing data among cooperative portlets has been addressed by many portal servers (Oracle, IBM, BEA, etc). They provide proprietary extensions to industry portlets standards such as the *Web Services for Remote Portlets* (WSRP) specification by OASIS [13], or the JSR-168 development model proposed by JCP [6]. However, these extensions are: (i) not portable to other servers, (ii) often require the use of concrete development tools closely tied to a particular platform technology and architectural style (e.g., the WebSphere Portal tool only supports the MVC design) and, (iii) implement a simplified means of data sharing among portlets. As a consequence, there is current a lack of guidelines and modeling concepts to address the portlets-based portal development from a technology-independent viewpoint.

Web Engineering proposals have traditionally aided the industry Web software development to further improve its productivity, quality and longevity. Although the majority of those proposals provide excellent methodologies and tools for the design and development of Web applications, the study of integrating Web applications within cooperative third party systems (such as Web services, portlets or legacy systems) has been particularly overlooked until recently. It is very likely to be introduced in future extensions, but currently there is current a lack of guidelines about how portlets should be modeled, how they should be integrated in a Web application or how inter-portlet communication capabilities should be offered.

In this paper we try to identify the main concerns involved in modeling portlets and determine the major models required to capture these concerns (see Section 2). Using the travel agency example, Section 3 proposes a strategy for modeling portlets and their integration into Web applications from a platform-independent viewpoint. In so doing, we introduce also mechanisms for modeling inter-portlet communication and cooperation capabilities independently of the target portal server product. Finally, Section 4 sketches some conclusions and outlines some further research activities.

2. Reference Models for Portlets

Broadly speaking, the main difference between a Web application and a portlet stems from the fact that the former is an aggregate of pages whereas the latter is an aggregate of fragments. Apart from that, both the Web application design and the portlet design share many features and concerns. Based on their similarities, we will make use of the general framework presented in [11, 12] for describing Web applications in order to identify the major required models involved in the modeling of portlets[1]. Next subsections look at framework viewpoints briefly and describe the concepts and models that rise up during the portlet-based applications construction. How to use them for addressing portlet specific requirements will be discussed in Section 3.

2.1 The User Interface Viewpoint

Typically, portlets are considered as user-facing applications which offer more than just content display to their users. They also allow them to interact with the content by means of forms, entry data fields, radio buttons, check boxes, etc. The User interface viewpoint is directly concerned with how the client interacts with the portlet and how information is structured for providing a user-friendly interface through a coherent look and feel of its visual elements.

[1]That framework was specifically designed to integrate third party applications and legacy systems into Web systems by separating independent concerns into a set of views on the system, each one addressing one particular viewpoint (user interface, business logic, persistent data, distribution, etc)

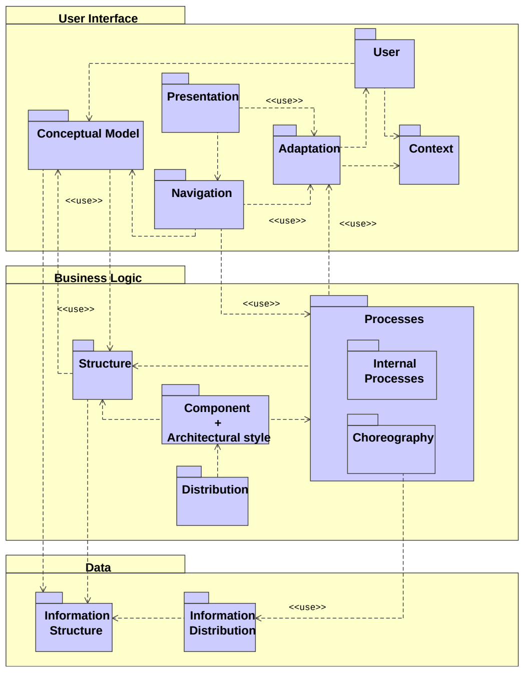


Figure 1. Reference models for portlets

This viewpoint consists of six main models: *Conceptual*, *Navigation*, *Presentation*, *User*, *Context* and *Adaptation* (see Figure 1). For modeling an individual portlet we need at least its *Conceptual*, *Presentation*, and *Navigation* models at the User Interface level. In case of adaptation requirements, the others could be needed, too. Generally, all these models are not usually supplied from the portlet providers; rather, they need to be built from the portlet information (which, by the way, tends to be imprecise, scarce, and insufficient). In any case, they are required for any model-driven development approach.

- The **Conceptual** model encapsulates the information handled by the rest of the models at the User Interface level. Since each user looks at the same information and navigates through the application in a different way, different views of this model can be defined. *UserInformationUnits*, *Attributes* and *Associations* comprise the major elements of this representation.

- The **Navigation** model describes the application navigational requirements building the navigational structure of the final application—a portlet in this case. This implies expressing how users can visit a collection of related data in a non-linear way. To this end, *NavigationUnits*, *NavigationLinks*[2], *Events* and *AccesStruc-*

[2]Not to be confused with UML "links". NavigationLinks represent as-

tures have been defining among other concepts for representing the logic of this model.

– The **Presentation** model captures the presentational requirements in a set of *PresentationUnits* (text, image, pages, sections, forms, etc.) and *Transitions*, which sketch aesthetic aspects and the look&feel of the portlet.

– The **User** model describes and manages the user characteristics [5, 4] with the purpose of adapting the content and the presentation of information to their needs and preferences. Generally, the *User* model is expressed in terms of the following concepts: *User*, *UserFeature*, *Role*, *Preference*, *PreviousKnowledge*, *History* and *Session*.

– The **Context** model. Following [7], context is defined as *the reification of certain properties, describing the environment of the application and some aspects of the application itself which are necessary to determine the need for customization*. Context deals with *Device*, *Network*, *Location* and *Time* aspects.

– The **Adaptation** model is performed based on user's knowledge, preferences or context features to obtain appropriate Web content characteristics and target markup for each device Generally, the Adaptation Model is expressed as a set of mathematical expressions or ECA rules associated with navigation and presentation concepts in adaptation models [5, 4, 9].

2.2 The Business Logic Viewpoint

This level describes the behaviour of the portlet, independently of its user interface, and the persistent data it handles. To provide detailed descriptions to perform business processes, actions, events handling and errors management is delivered to this viewpoint.

From our point of view, modeling the behaviour of a portlet can be done using the following five models (see Figure 1): *Structure*, *Internal Processes*, *Choreography*, *Distribution* and *Architectural Style*. Among them, the models that comprise the description of the portlet are: the *structure*, *internal processes*, and *choreography* models.

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

– The **Internal Processes** model specifies the precise behavior of every *BusinessProcessInformation* or component as well as the set of activities that are executed in order to achieve a business objective.

– The **Choreography** model defines the valid sequences of messages and interactions that the portlet may exchange [17]. The choreography may be externally oriented, specifying the contract a component will have with other components (*PartialChoreography*) or, it may be internally oriented, specifying the flow of messages within a composition (*GlobalChoreography*).

– The **Distribution** model describes how functionality is distributed in *Nodes*, connected by means of point to point connections or *Links*. *Nodes* can be static, dynamic or mobile. *StaticNodes* represent *Devices*, *Places* or *Actors* of the system. Dynamic nodes model points of computing (*ComputingNode*) where *Activities* are performed. Both *StaticNodes* and *ComputingNodes* may be *MobileNodes* for relevant issues of a mobile *System*, concerning the mobility of both physical (e.g. computing nodes) and logical entities (e.g. software components). The global view of *Nodes* and *Links* comprises a *Network*.

– The **Architectural Style** model deals with how functionality is encapsulated into business components and services. Each *Architectural Style* has different associated entities depending on the pattern applied which makes the distinction between different architectural styles. For portlets development, the MVC style is the most used. It defines: (i) a *Model* that is responsible for storing the state and the application logic; (ii) *Views* that provide the user of the application with a presentation of the current state of the application; and (iii) a *Controller* that is responsible for mapping user interface events to calls that invoke operations on the model. That is, the glue that ties the model to the view and defines the interaction pattern in the portlet.

2.3 The Data Viewpoint

Although portlets cannot be considered as data-centric applications, they require high-performance infrastructure for data storing. Information is needed not only for portlet consumers but also for its processes. This level describes the information handled by the application and provides a mechanism for managing and storing data persistently.

This viewpoint is organized around two models: the *Information Structure* and the *Location* models.

sociations in the Navigation model, whereas UML links are just instances of UML associations.

– The ***Information Structure*** model deals with the information that has to make persistent in terms of *InformationUnits*, *Attributes*, *Relationships*, *Constraints* and *AccessOperations*.

– The ***Location*** model describes the distribution and replication of the data being modelled, since information can be fragmented in *Nodes* or replicated in different *Locations*.

These models are not relevant in our case, because we are concerned on how to interact with the portlet, not how it internally stores its persistent data.

For a more detailed description of these viewpoints and the semantic of their associated models and concepts, the interested reader can refer to [11].

2.4 Dependencies between Models

Although the models of the framework are ideally independent of each other, some of them capture requirements on the same element of design (e.g., events, properties or actions). Therefore some modifications in a model can affect other models. We add precision to model elements and the relationships among them by means of unambiguous OCL constraints (preconditions, invariants and postconditions). For example, Figure 1 shows a connection between *Presentation* and *Structure Business* models. In this way, *Transitions* in the *Presentation* model consists of *Events* which model a significant occurrence located in time and space. These *Events* have to be also *Events* in the context of *Internal Processes* models, which trigger the execution of an associated behaviour. This can be formally expressed as an OCL constraint.

Interdependencies determine not only the methodological guidelines to be followed, but also something more important: they establish how the different viewpoints merge and complement each other.

3. Designing Portlet-based Web Applications

The design of a portlet-based Web application can be addressed combining existing models of individual portlets with additional models describing the extra-functionality supported by our Web application. Following this strategy, the design process is strongly governed by the (too early) implementation decision of using portlets and, consequently, if we want to reuse another type of component technology or provider in the future, we are forced to redefine all the Web application models from the beginning. On the contrary, we have opted for delaying as much as possible implementation decisions, in order to obtain a set of reusable platform-independent models.

Our proposal is aligned with the MDA framework [16, 1, 10] and particularizes general guidelines presented in previous work [11, 12]. To be precise, the development of portlets-based Web applications in the MDA context is based on the following steps:

Step 1 Create the class diagrams (PIMs) describing models for each Web application layer. In this step, we identify the global systems requirements at three levels of abstraction: *User Interface*, *Business Logic* and *Data* levels. As a result of this phase, three PIMs are generated describing high-composition architectural views of the services and components of our application.

Step 2 Mark the PIM elements with stereotypes identifying the system scope and boundaries, i.e, the data and services that will be provided by our system, and the ones that will be externally required. At this point, we will describe how portlets cooperate (*portlet aggregation* [2]) with each other to produce and achieve the global functionality that the system is required to implement—either by sending signals or invoking operations.

Step 3 Specify the target platform. We need to determine the concrete platforms and communication mechanisms between our application and the external systems identified previously.

Step 4 Generate the PSMs.

Step 5 Generate the code (e.g., the Web pages).

Integration and cooperation among portlets can be carried out at two levels of abstractions: at the user interface level and at business logic level. In following subsections we are going to focus on these two viewpoints proposing a strategy for addressing both issues through the travel agency system (TAS) example (see `http://www.lcc.uma.es/~av/mdwe2005/TheTASexample/` for a complete description of this scenario).

3.1. Addressing the User Interface Level

At first sight, it could be considered that the role of the main system for portlet-based Web applications is limited to a customizable facade providing a single sign-on service for assembled portlets. However, the unique rendering space of a portal adds complexity to the application model when we consider multiple portlets co-existing on a Web page.

As mentioned in the introduction, when a user navigates within one portlet the others usually remain unchanged, ignoring what is being rendered by it. In order to transfer data from one portlet to another, users have to manually copy and paste key data from sources to targets portlets. We will try to model inter-portlet data dependencies—at user interface level—to free users from this task.

3.1.1 The Conceptual Model

To design the PIM of this viewpoint, we need to identify—in the first place—the information that will be presented to the user during a session . These requirements are captured in the *Conceptual model*. That model for the TAS example is shown in Figure 2. However, we need to decide how to deal with this model when we want to re-use particular portlets such as Iberia, Bancohotel or PepeCar which have their own non-public *Conceptual Models*. In many cases, these portlets will place additional constraints—as well as specialized collaborations and data exchange—that can not be modelled in a typical UML class diagram.

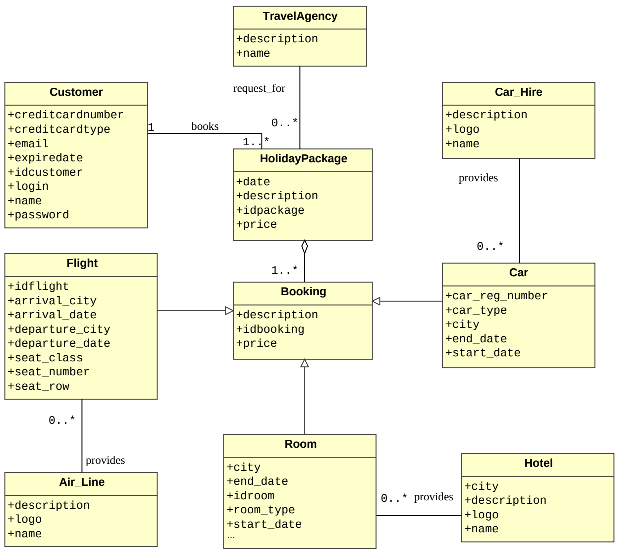


Figure 2. Conceptual model for the TAS

Therefore, we need to particularize and model the ability by which specific class instances (specific portlet implementations) will match the system requirements and will communicate/share data with each other. In this regard, UML 2.0 [15, 14] introduces a new structural diagram called *Composite Structure* diagram. A *Composite Structure* diagram "*depicts the internal structure of a classifier, including the interaction points of the classifier to other parts of the system. It shows the configuration of parts that jointly perform the behavior of the containing classifier. The architecture diagram specifies a set of instances playing parts (roles), as well as their required relationships given in a particular context*."

Consequently, starting from Figure 2 we are going to derive a *Composite Structure Conceptual Model*, as a set of

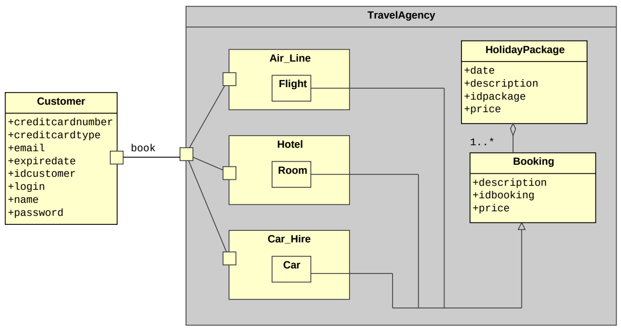


Figure 3. Composite conceptual model

parts interacting together to achieve user interface requirements. Since *Airline*, *Hotel* and *CarHire* classes are only used in the navigational context of the *TravelAgency* class, we model this fact considering them as parts/properties of the containing class *TravelAgency*. Likewise, the *Flight*, *Room* and *Car* classes form part of the *Airline*, *Hotel* and *CarHire* data structures respectively. Thus, Figure 3 represents the fact that when an instance of the *TravelAgency* is created, a set of instances corresponding to its properties (one *Airline*, *Hotel* and *CarHire* instances) are created as well—either immediately or at some later time. Each part or property is isolated from its environment by means of a port, which will drive the interactions with its environment[3]. On the other hand, connectors define channels along which messages are sent. Thus, when a costumer makes any request to the portal, the request is captured by the *TravelAgency* port, which delegates it to the appropriate portlet on the portal page.

Although this model works well at the Conceptual level, it is incomplete because it does not reflect the internal *collaborations* between the *TravelAgency*, *AirLine*, *Hotel* and *Car* classes. Rather, it models a travel agency system from a global point of view. Moreover, if we want to model how a data item is shared (or broadcasted) from one portlet to multiple target portlets in the page, we need to define a "*collaboration*". A UML *collaboration* is a selective view of that situation. It may be attached to an operation or a classifier through a *CollaborationOcurrence*. Such a *collaboration* can constrain the set of valid interactions that may occur between the instances that are connected by a link. Furthermore, a *collaboration* specifies the property instances that can participate in the *collaboration*.

For example, consider Iberia, Bancotel and PepeCar, three portlets on the same application screen. It would be interesting to retrieve and use the arrival City from the row in the summary *Iberia* portlet listing as an entry value for the

[3]As we shall see, ports will represent portlets containers at implementation level for the *Airline*, *Hotel* and *CarHire* classes.

City textbox of the Bancotel portlet. Then, the page would present the list of hotel offers to the customer. In this way, both portlets now would display related information, while the third portlet still shows their entry panel (Figure 4 illustrates this collaboration).

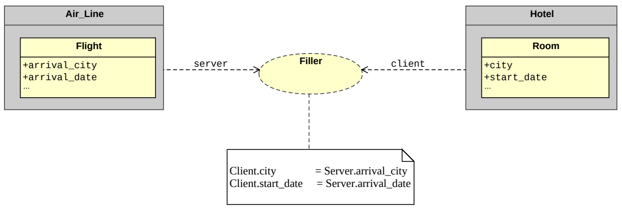


Figure 4. Collaboration diagram for the TAS

As an aside, please note that although inter-portlet communication can be modeled using any of the aforementioned methods (*Composite Structure* diagrams and *collaborations*), one must be careful about the dependencies that are introduced when using inter-portlet communication. In the Iberia/Bancotel portlets example, what would happen if the user decided to remove the Iberia portlet from his page? Would the Bancotel portlet still be functional? Should we force the user to remove the Bancotel portlet as well? These are some of the design considerations that need to be taken in account before using inter-portlet communication.

3.1.2 The Navigation Model

At this point, the *Conceptual* model is complete. Then, the *Navigation* model is built as a refinement of the *Conceptual* model we have just defined. The *Navigation* model specifies the navigational structures of the Travel Agency, i.e., how users navigate through the available information using *Indexes* (≪Index≫ HolidayPackageIndex), *Menus* (≪Menu≫ HolidayPackageMenu, ≪Menu≫ BookingMenu, ≪Menu≫ CustomerMenu) or *Guided Tours* (≪GuidedTour≫ BookingGuidedTour). We have added constraints to ≪NavigationLinks≫ describing which events will trigger the navigation through the link (e.g., when a process finishes, after clicking a ≪MenuOption≫, etc.)

3.1.3 The Presentation Model

After that, the *Presentation* model further refers to groups of pages organized around ≪PresentationUnits≫ as: (i) ≪SinglePresentationUnits≫, with their attributes marked as ≪text≫, ≪image≫, ≪button≫, etc.; and (ii) ≪GroupPresentationUnits≫ that comprise UML classes and packages stereotyped ≪page≫, ≪section≫ or ≪form≫. Basically, we have used in our example ≪ExternalSection≫ to display portlet responses and ≪page≫ to display the main portal pages. We have also marked as ≪ExternalForms≫ those UML classes that invoke external services. Note that each ≪ExternalPresentationUnits≫ has information about its own data, structure, presentation, etc.

Since adaptation is not required in this case, the final PIM of the *User Interface* viewpoint is obtained by merging these three models, and is shown in Figure 5.

Once the PIM of this layer is described, we need to provide some sort of support for its deployment, configuration and execution in a particular platform, i.e., we need to generate its corresponding PSM. This last step only concerns UML packages stereotyped ≪GroupPresentationUnits≫, because integrated portlets deal with their corresponding ≪ExternalGroupPresentationUnit≫. Applying a two-fold transformation process from model-to-model (based on, e.g., ATL mapping rules) and model-to-code (using, e.g., templates that contain predefined parts of the meta-code text), we can map the source PIM to a target PSM and to code finally. The feasibility of these transformations and how the Web pages are obtained from them is well documented in [8].

3.2. Addressing the Business Logic Level

The same as the User Interface Viewpoint, the Business Logic view of the system needs to be platform-independent and interchangeable. Portlets typically evolve over time and are largely reused as bases for new portlets. Thus, the ability to change a portlet model or to adapt it to a new provider requires that each business logic model be self-contained and extensible.

As shown in Figure 6, the PIM for this viewpoint is focused just on the system operations hiding the rest of the details (software architecture, distribution, system boundaries, communication protocols, implementation platforms, etc.). This solution is specified in terms of UML packages and their interconnections in a platform independent way, achieving reusability across different target platform environments. Thus, the PIM does not contain any information on the pieces of functionality that will be locally implemented, and the ones that will be provided by external services and applications.

Once that high-level PIM is specified, we need to identify the system scope and boundaries, and then build a model of the system with this information. That target model (still a PIM, but with that information on it) will be built by transforming the original PIM using marks (see Figure 7). To identify the elements in the TAS PIM that should be transformed in a particular way, we will use the stereotypes ≪ExternalSystem≫ and ≪ExternalAssociation≫. An ≪ExternalSystem≫ defines any other external system interacting with the system under

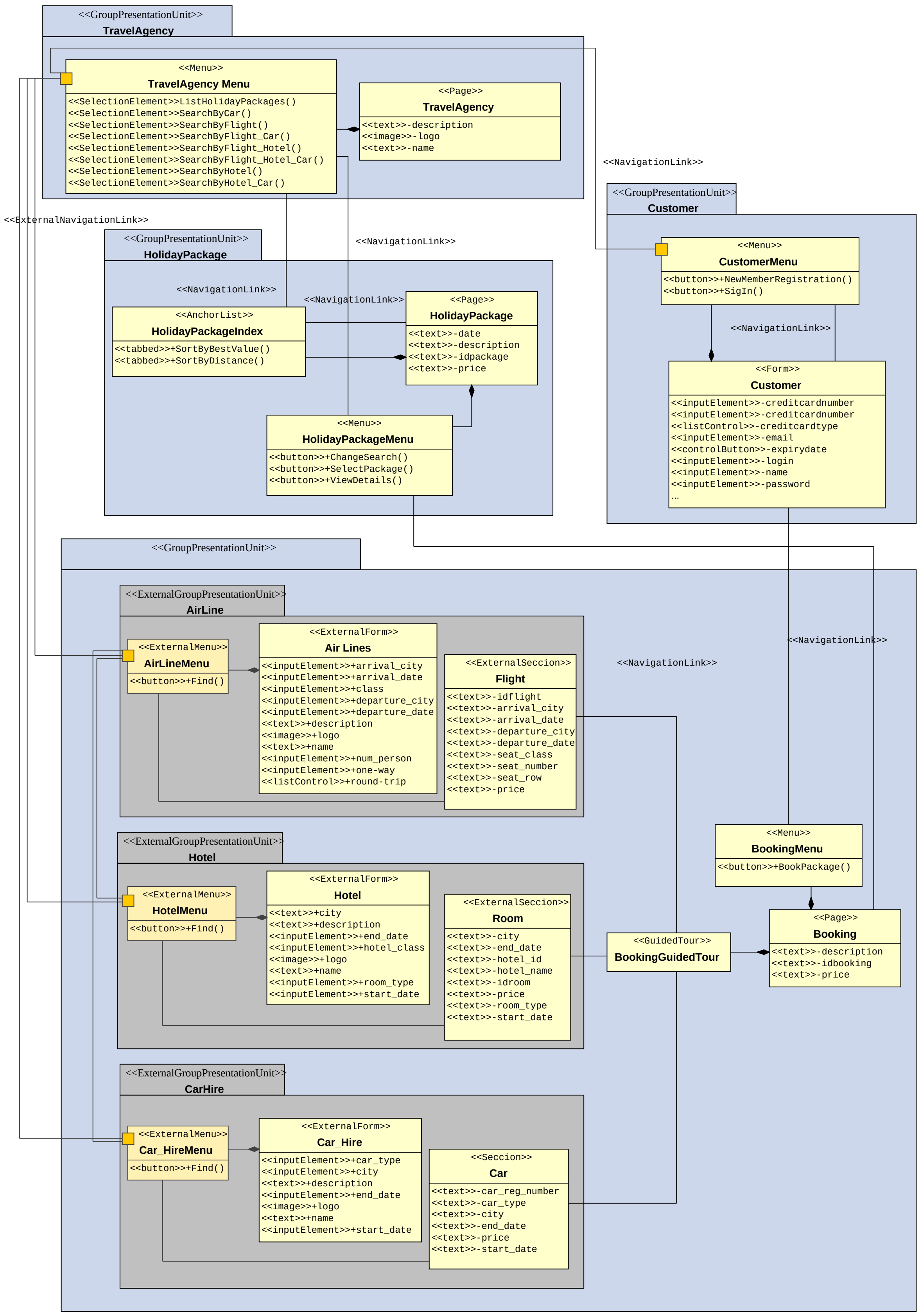


Figure 5. The PIM of the User Interface viewpoint

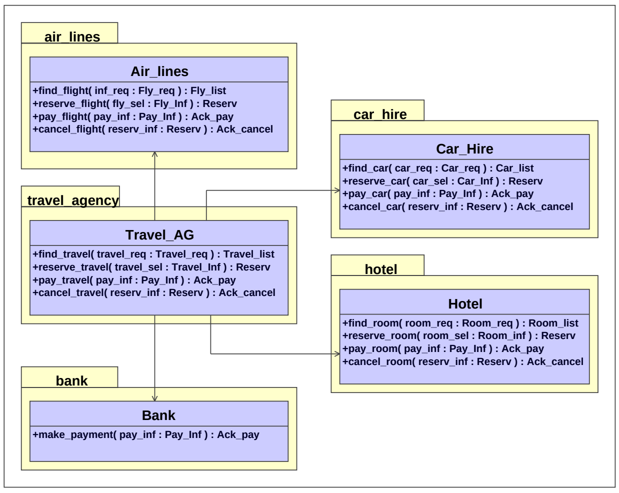

air_lines
Air_lines
+find_flight(inf_req : Fly_req) : Fly_list
+reserve_flight(fly_sel : Fly_Inf) : Reserv
+pay_flight(pay_inf : Pay_Inf) : Ack_pay
+cancel_flight(reserv_inf : Reserv) : Ack_cancel
car_hire
Car_Hire
+find_car(car_req : Car_req) : Car_list
+reserve_car(car_sel : Car_Inf) : Reserv
+pay_car(pay_inf : Pay_Inf) : Ack_pay
+cancel_car(reserv_inf : Reserv) : Ack_cancel
travel_agency
Travel_AG
+find_travel(travel_req : Travel_req) : Travel_list
+reserve_travel(travel_sel : Travel_Inf) : Reserv
+pay_travel(pay_inf : Pay_Inf) : Ack_pay
+cancel_travel(reserv_inf : Reserv) : Ack_cancel
hotel
Hotel
+find_room(room_req : Room_req) : Room_list
+reserve_room(room_sel : Room_inf) : Reserv
+pay_room(pay_inf : Pay_Inf) : Ack_pay
+cancel_room(reserv_inf : Reserv) : Ack_cancel
bank
Bank
+make_payment(pay_inf : Pay_Inf) : Ack_pay

Figure 6. The TAS Structure PIM

consideration. In the same way, an ≪ExternalAssociation≫ defines an interaction between the system under deployment and an ≪ExternalSystem≫ [11].

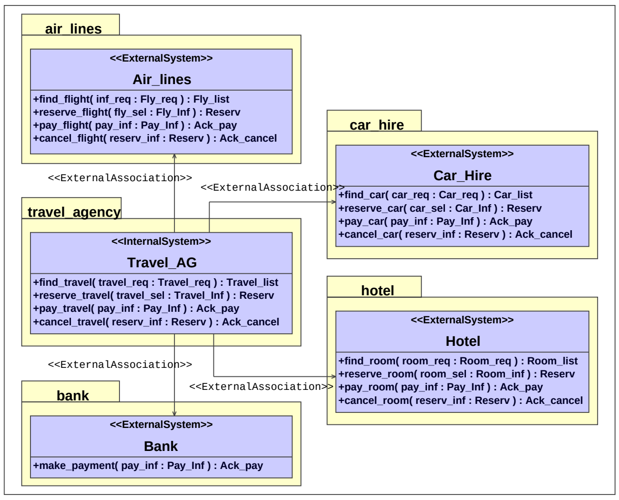

air_lines
<<ExternalSystem>>
Air_lines
+find_flight(inf_req : Fly_req) : Fly_list
+reserve_flight(fly_sel : Fly_Inf) : Reserv
+pay_flight(pay_inf : Pay_Inf) : Ack_pay
+cancel_flight(reserv_inf : Reserv) : Ack_cancel
car_hire
<<ExternalSystem>>
Car_Hire
+find_car(car_req : Car_req) : Car_list
+reserve_car(car_sel : Car_Inf) : Reserv
+pay_car(pay_inf : Pay_Inf) : Ack_pay
+cancel_car(reserv_inf : Reserv) : Ack_cancel
<<ExternalAssociation>>
<<ExternalAssociation>>
travel_agency
<<InternalSystem>>
Travel_AG
+find_travel(travel_req : Travel_req) : Travel_list
+reserve_travel(travel_sel : Travel_Inf) : Reserv
+pay_travel(pay_inf : Pay_Inf) : Ack_pay
+cancel_travel(reserv_inf : Reserv) : Ack_cancel
hotel
<<ExternalSystem>>
Hotel
+find_room(room_req : Room_req) : Room_list
+reserve_room(room_sel : Room_inf) : Reserv
+pay_room(pay_inf : Pay_Inf) : Ack_pay
+cancel_room(reserv_inf : Reserv) : Ack_cancel
<<ExternalAssociation>>
<<ExternalAssociation>>
bank
<<ExternalSystem>>
Bank
+make_payment(pay_inf : Pay_Inf) : Ack_pay

Figure 7. The marked TAS PIM

Note that the marked PIM is, by definition, technology independent. In consequence, the prefix "External" used by the stereotypes ≪ExternalSystem≫ and ≪ExternalAssociation≫ in Figure 7 does not imply any implementation decisions. Instead, it is only used to limit the system scope and boundaries.

At this point, when we have the marked PIM, we still need to transform it further into a model that contains the information about how the system services are "componentized", prior to decide the technology and the service providers. This componentization will be described using the UML 2.0 constructs and infrastructure for describing software architectures, because what we want to build in this phase is the software architectural description of the system. This transformation will be guided by the following mapping rules:

- **Packages transformation.** Each UML package is mapped to a UML Component initialized with the same as its corresponding UML package.
- **Classes transformation.** The UML class stereotyped as ≪InternalSystem≫ or ≪ExternalSystem≫ is mapped to a UML Class holding the same characteristics as its original (name, attributes and operations).
- **Associations transformation.** For each UML association stereotyped ≪ExternalAssociation≫, two component ports will be generated, each one as Association ends of that relationship. Ports will be associated to the UML Component derived in the previous step. Its behavior is defined in terms of an interface associated with that port, which specifies the nature of the interactions that may occur over that port. Thus, the port interface's name is given the value of the UML class name from which it is derived and its operations correspond to its UML class operations.
- **Association's ends transformation.** For the endpoint of an ≪ExternalAssociation≫ stereotyped ≪InternalSystem≫, a usage dependency from the port to the interface is generated, showing how the ≪InternalSystem≫ provides a set of services.

  For the endpoint of an ≪ExternalAssociation≫ stereotyped ≪ExternalSystem≫, an implementation dependency from the port to the interface is generated, showing how the ≪ExternalSystem≫ requires a set of services.
- Finally, an assembly connector is defined from every required Interface to its provided Interface.

Applying these mapping rules on the PIM in Figure 7, we obtain the model shown in Figure 8. However, this model is incomplete since it does not model how navigation events (clicks on screen links or submission of Web forms) result in portlet actions being received by the portal. Analogously, it does not specify how the portal forwards the actions it captures to the appropriate portlet containers that will handle the requests. Another important issue that we need to model is how message-forwarding across portlets can be carried out. Messages are either calls to operations (which are usually dispatched to methods on the receiving

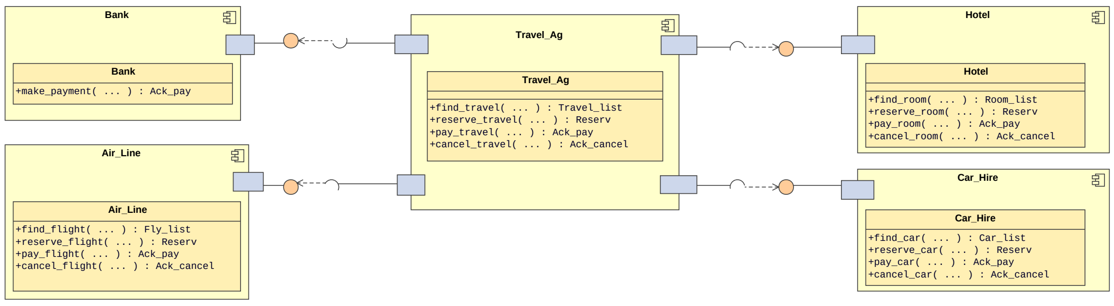

Bank
Bank
+make_payment(...) : Ack_pay
Air_Line
Air_Line
+find_flight(...) : Fly_list
+reserve_flight(...) : Reserv
+pay_flight(...) : Ack_pay
+cancel_flight(...) : Ack_cancel
Travel_Ag
Travel_Ag
+find_travel(...) : Travel_list
+reserve_travel(...) : Reserv
+pay_travel(...) : Ack_pay
+cancel_travel(...) : Ack_cancel
Hotel
Hotel
+find_room(...) : Room_list
+reserve_room(...) : Reserv
+pay_room(...) : Ack_pay
+cancel_room(...) : Ack_cancel
Car_Hire
Car_Hire
+find_car(...) : Car_list
+reserve_car(...) : Reserv
+pay_car(...) : Ack_pay
+cancel_car(...) : Ack_cancel

Figure 8. The TAS PIM after applying the MDA transformation

object), or sending of signals (which are buffered on the receiving object and handled by the corresponding behaviors in the objects). The *Internal Process* model specifies both internal and external behavioral aspects by means of activity, interaction and sequence diagrams.

Space limitations prevent us from giving a more detailed description of these specifications. We show an excerpt of the activity diagram for the *FindTravel* process in Figure 9. Notice that some portlet-data dependencies have been previously captured in the *User Interface PIM*.

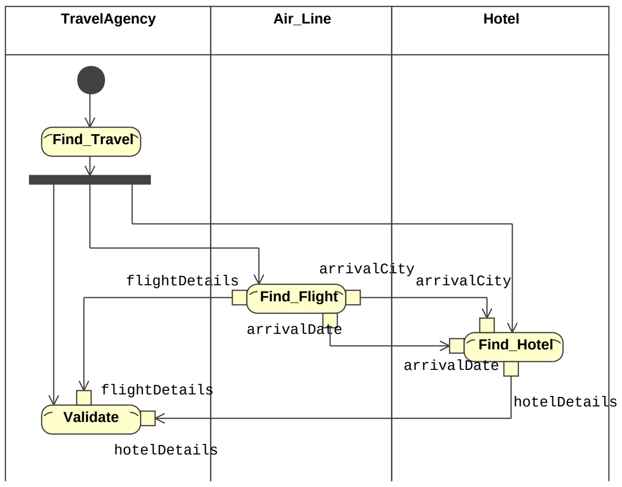


Figure 9. An excerpt of the Activity Diagram for the *FindTravel* process

At this point, we have a set of models with the information about: (a) the system functionality, from a global point of view; (b) how it is "componentized" into different services; (c) which of these services are internally implemented, and which ones are provided by external services; (d) the interactions between all the system services (both internal and external).

From there, the process of building the PSM of the system based on the implementation technologies and platforms can proceed as in our previous work [11, 12].

4. Conclusions

In this paper we have discussed the issues involved in the integration of portlets into model-based Web application development. In general, the majority of Web Engineering proposals do not yet support the integration of third party systems with Web applications (by means of suitable design concepts and models), including Web Services and portlets. Our contribution tries to shed some light on this area, by analyzing the different kinds of concerns that need to be addressed, and the models required to capture such information.

There is not much work that deals with these issues. Díaz et al. consider in [3] a subset of the concerns involved in a portal construction and define a set of platform-independent models for them, namely the service model, the orchestration model and the presentation model. Although it is a very good and expressive approach, the proposal is in the context of building Web portals using portlets, and not for developing general Web applications. Thus, the proposed design process is strongly governed/influenced by the (too early) implementation decision of using portlets. In this respect, our proposal follows an MDA approach, and hence allows the system developer to take that decision at a later stage, and then use the portlet models if required.

This paper also extends our previous work on integrating Web Services into Web applications using a model-driven approach [12]. It is important to notice the strong relationship between the User Interface models and the Business Logic models of a portlet. Such a relationship was not required in the case of Web Services, since they sit at the Business Logic level only. However, this relationship is crucial for integrating portlets into Web applications, because they contain not only functionality, but also presentation—and they are closely related.

As future work we plan to continue working on case studies and applications that help validate our proposal, and that serve as proof-of-concept of our ideas. Our goal is to develop a complete set of Web applications using our approach which can illustrate the problems that appear when integrating external systems into Web applications, and how to tackle them from a model-driven approach.

References

[1] A. W. Brown. Model driven architecture: Principles and practice. *Software System Model*, 3:314–327, 2004.

[2] O. Díaz, J. Iturrioz, and A. Irastorza. Improving Portlet aggregation through deep annotation. *Proceedings of the 14th International World Wide Web Conference (WWW 2005)*, May 2005. Japan.

[3] O. Díaz and J. Rodríguez. Portlets as Web Components: an Introduction. *Journal of Universal Computer Science*, 10(4):454–472, Apr. 2004. `http://www.jucs.org/jucs_10_4/portlets_as_web_components"`.

[4] F. Frasincar, G.-J. Houben, and R. Vdovjak. An RMM-Based Methodology for Hypermedia Presentation Design. *Proceedings of the 5th East European Conference on Advances in Data-bases and Information Systems (ADBIS 2001)*, LNCS 2151:323–337, September 25-28 2001. Vilnius, Lithuania.

[5] C. Gnabo. Web-based Information Systems Development – A User Centerd Engineering Approach. *Web Engineering: Managing Diversity and Complexity of Web Application Development*, LNCS 2016:105–118, 2001.

[6] Java Community Process. *JSR 168 Portlet Specification Version 1.0*, 2003. `http://www.jcp.org/en/jsr/detail?id=168`.

[7] G. Kappel, W. Retschitzegger, and W. Schwinger. Modelling Customizable Web Applications – A Requirement's Perspective. *Proceedings of the International Conference on Digital Libraries: Research and Practice (ICDL)*, 2000. Koyoto, Japan.

[8] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained. The Model Driven Architecture: Practice and Promise*. Addison-Wesley, Apr. 2003.

[9] N. Koch. *Software Engineering for Adaptive Hypermedia Systems - Reference Model, Modelling Techniques and Development Process*. PhD thesis, Fakultt der Mathematic und Informatik, Ludwig-Maximilians-Universitt Mnchen, Dec. 2000.

[10] J. Miller and J. Mukerji. *MDA Guide*. Object Management Group, Jan. 2003. OMG document ab/2003-06-01.

[11] N. Moreno and A. Vallecillo. A model-based approach for integrating third party systems with web applications. *Fifth International Conference on Web Engineering (ICWE2005)*, July 2005. Sydney, Australia.

[12] N. Moreno and A. Vallecillo. Modeling interactions between web applications and third party systems. *Fifth International Workshop on Web Oriented Software Technologies (IWWOST2005)*, June 2005. Porto, Portugal.

[13] OASIS. *Web Service for Remote Portlets Specification Version 1.0*, 2003. `http://www.oasis-open.org/specs/index.php#wsrpv1.0`.

[14] Object Management Group. *UML 2.0 Infrastructure Specification*, 2003. `http://www.omg.org/cgi-bin/doc?ptc/03-09-15.pdf`.

[15] Object Management Group. *UML 2.0 Superstructure Specification*, 2003. `http://www.omg.org/cgi-bin/doc?ptc/03-08-02.pdf`.

[16] OMG. *Model Driven Architecture. A Technical Perspective*. Object Management Group, Jan. 2001. OMG document ab/2001-01-01.

[17] OMG. *A UML Profile for Enterprise Distributed Object Computing*. Object Management Group, May 2002. OMG document PTC/2002-02-05.

Conceptualization of Navigational Maps for Web Applications

Antonio Navarro, José Luis Sierra, Alfredo Fernández-Valmayor, Baltasar Fernández-Manjón
Dpto. Sistemas Informáticos y Programación
Universidad Complutense de Madrid
{anavarro, jlsierra, alfredo, balta}@sip.ucm.es

Abstract

The characterization of navigational maps for Web applications helps its developers in the process of construction of these applications and it also helps users while performing their browsing. This paper presents the Pipe approach for characterizing navigational maps for Web applications at their conceptualization stage. Although Pipe was originally intended as a hypermedia-oriented notation it has been successfully used for characterizing navigational maps for Web applications. In addition, navigational maps described in terms of the Pipe notation can be easily mapped to UML-Conallen class diagrams at the design stage. This paper presents the characterization of the travel agency's navigational map using Pipe notation at the conceptualization stage. The paper also demonstrates the mapping of this Pipe navigational map to UML-Conallen class diagrams.

1. Introduction

The development of Web applications is not an easy task. In the development process of this type of applications, at least three model types must be provided: *conceptual*, *navigation* and *presentation* [15]. Conceptual/structural model describes the organization of the information managed by the application in terms of the contents that constitute its information base and semantic relationships that exist among them. Navigation model describes the facilities for accessing information and for moving across the application content. Presentational model describes the way in which application content and navigation commands are presented to the user [18].

Regarding navigation model, *navigational maps* can be a very useful tool. These maps describe a global view of a Web application for an audience [22]. At present, many web sites include navigational maps to help users during browsing. Therefore, in our opinion, the characterization of navigational maps is a key issue during development of Web applications. Using navigational maps, developers can obtain a global view of the whole application that can help them in the development process. In addition, the presence of navigational maps can help users of Web sites to find the desired information much quicker.

Taking into account the development process of Web applications and the nature of the type of software, requires one to distinguish between two distinct stages in their development [18]. One stage focuses on conceptualization and prototyping, while the other pays attention to design and development. At conceptualization stage, the application is represented by a set of abstract models that convey the main components of the envisioned solution. At the design stage, conceptual schemas are transformed into a lower-level representation, closer to the needs of implementation, but still independent of the actual content of the information base.

This paper presents the Pipe notation [3] and uses it to characterize the navigational map of the travel agency's case study at the conceptualization stage. The paper also shows the mapping of the travel agency navigational map in terms of Pipe notation to an UML-Conallen class diagram [10] at the design stage. Although Pipe was originally intended as a hypermedia notation [1], its use in Web engineering projects has demonstrated its applicability as a tool to characterize navigational maps. In addition, due to the abstraction level of Pipe notation, it's possible to map it to specific design notations.

The paper is organized as follows. Section 2 briefly describes Pipe notation. Section 3 provides the Pipe characterization of the travel agency's navigational map. Section 4 maps Pipe characterization to UML diagrams that use Conallen's extension. Section 5 compares Pipe with other approaches. Finally, in section 6 conclusions and future work plans are presented.

2. Pipe notation

Pipe is a formalized graphical notation for the characterization of hypermedia applications. A description of the notation can be found in [2] whereas the formalization can be found in [3]. Pipe is conceived to be used in domains not easily representable in terms of entities/classes and their relations. Consequently, Pipe is focused on characterizing: (i) the units of information as individuals and relations among them; (ii) the user interface of the application, and (iii) the relations between the layer of contents and the layer of user interface to provide navigational access to the elements of the application.

Pipe provides three types of diagrams to characterize the above mentioned information: (i) a *contents graph*, which characterizes the information elements of the application and their relations; (ii) a *navigational schema*, which characterizes the graphical user interface of the application; and (iii) a set of *canalization functions*, which relate the contents graph and the navigational schema. This section briefly describes these elements.

2.1 Contents graph

The Pipe contents graph models the units of information in the application and the way they are related. It only considers individual units of information, anchors, links and a relationship function that captures all the semantic relations between the units. The Pipe contents graph uses the following modeling elements:

– *Static contents*. Units of information that exist prior to any interaction by the user.

– *Static anchors*. Anchors whose destinations are static contents.

– *Static links*. Relations between static anchors.

– *Dynamic contents*. Units of information generated by user's interaction [21].

– *Dynamic anchors*. Anchors whose destinations are dynamic contents.

– *Dynamic links*. Relations between dynamic anchors. In Pipe, dynamic links are characterized using a *generation function g* [3] specific for each application, but defined considering a common signature. Therefore, *g* function acts like the implementation of object-oriented interfaces and is used in Pipe to model dynamic behavior in the hypermedia applications at a conceptual level.

Figure 1 depicts the visual elements of the contents graph.

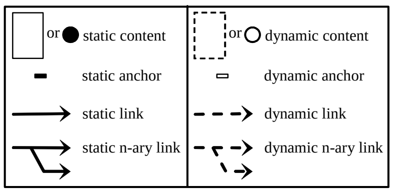


Figure 1. Visual elements of the contents graph

The Pipe graphical notation is fully formalized [3]. The formalization of the contents graph is made in terms of the *relationships function r*. This function has an extensional definition for static anchors, and an intensional definition, in terms of the generation function *g*, for dynamic anchors. In such a way, function *r* acts as a *black box* that "hides" the real nature of the links (static or dynamic).

2.2 Navigational schema

The Pipe navigational schema makes an important abstraction of the graphical elements that compose a graphical user interface (GUI) and the navigational relations that appear among the elements of the GUI in hypermedia applications. The Pipe navigational schema uses the following modeling elements:

– *Nexus nodes*, which represent the *windows* that contain the rest of elements of the GUI.

– *Container nodes*, which represent the *panes* that are inside the windows. They contain the contents (static or dynamic) of the contents graph.

– *Nexus activator nodes*, which represent the *buttons* that are inside the windows. The only event associated to this type of buttons is the activation of another window.

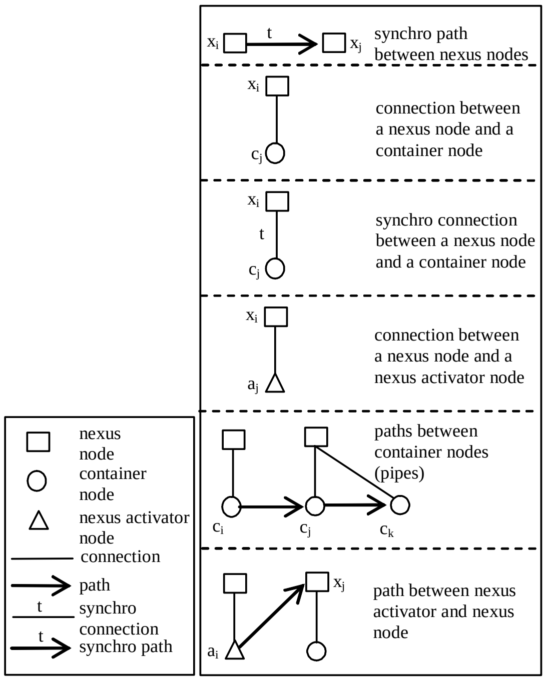
– *Connections*, which represent the way to relate a container node or a nexus activator node to its nexus node (i.e. they represent relations of aggregation between windows and their panes and buttons).

– *Paths*, which represent the navigational relations that exist between container nodes in the GUI. In this way if a path exists between a container node c_1 and a container node c_2, it is possible to display in c_2 the destination of a link with origin in a content that is being displayed in c_1. Paths are also called *pipes* because they are responsible for *canalizing* (interpreting) the relations (links) established between the units of information (contents) in the navigational schema (GUI).

– *Synchro connections*. Which represent time-activated connections.

– *Synchro paths*. Which represent time-activated paths established between nexus nodes.

Figure 2 depicts the visual elements of the navigational schema and the allowed relationships between them.



nexus node
container node
nexus activator node
connection
path
t
synchro connection
t
synchro path
x_i
t
x_j
synchro path between nexus nodes
x_i
c_j
connection between a nexus node and a container node
x_i
t
c_j
synchro connection between a nexus node and a container node
x_i
a_j
connection between a nexus node and a nexus activator node
paths between container nodes (pipes)
c_i
c_j
c_k
x_j
a_i
path between nexus activator and nexus node

Figure 2. Visual elements of the navigational schema and allowed relationships between them

2.3 Canalization functions

The Pipe canalization functions relate the elements of the contents graph to the elements of the navigational schema. In this way several user interfaces can be used with the same units of information and semantic relations. Likewise, the same user interface can be used with different units of information and/or semantic relations. There are two canalization functions in Pipe:

– *Content assignation function*. This function assigns units of information (contents) to container nodes (panes).

– *Canalization function*. This function assigns relations between units of information (links) to paths/pipes among container nodes (panes).

Colors and visual patterns are used in Pipe to characterize the canalization functions.

3. Conceptualization of the travel agency

The description provided for the travel agency in the workshop's specification is focused on the functional behavior of the use cases in the system. Therefore, using a UML-based approach [7][16], use cases, activity, classes, interaction, component and deployment diagrams can be provided.

Although this specification is not focused on the description of the navigational map of the application, its navigational map can be easily characterized. Next sections describe this navigational map using the Pipe notation.

3.1 Contents graph

Pipe notation was intended to characterize hypermedia applications with domains not easily characterizable in terms of class or entities and their relationships [3]. Consequently, Pipe has focused on the characterization of individual units of information and their semantics relations [13]. As a result, the modeling primitives of Pipe are appropriate to characterize the heterogeneous relations established among individual *pages* of Web applications [10] (e.g. HTML [27] or JSP [9] pages).

Note that in the case study of the travel agency an important number of homogeneous elements and relations can be characterized using classes and their relations. Due to the origins of our approach, this information about classes and relationships cannot be represented in Pipe. Instead, UML class diagrams can be used to complement Pipe notation at the conceptualization stage. The integration of UML class diagrams and Pipe contents graph is made using the generation function *g*. The description of this integration process is out of the scope of this paper. In this section we provide the Pipe characterization of the contents accessed by the user during the browsing of the application, omitting the UML characterization. This Pipe characterization serves as the navigational map for the application. Section 4 provides the mapping of this Pipe characterization to UML diagrams at the design stage. In these UML diagrams the structure of classes and their relations could also be provided. Figure 3 shows the contents graph of the application that is indeed the navigational map of the travel agency.

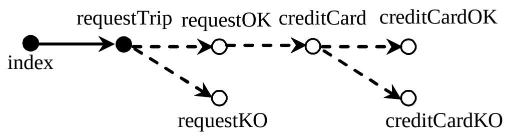


Figure 3. Contents graph/navigational map of the travel agency

The navigational map of Figure 3 describes a static index (`index`) which is linked to a static form that picks up data related to the requested trip (`requestTrip`). This form dynamically links [21] with the page that shows the result of the search for the trips in the travel agency. If the search fails (`requestKO`), the user must use the `index` to continue browsing the application. If the search obtains positive results (`requestOK`), the user can navigate to a form where his/her credit card data can be introduced (`creditCard`). Then these data are validated and independently of the result of the validation (`creditCardOK` or `creditCardKO`) the user must use the `index` to continue browsing the application. In this figure, the information about anchors [3] is omitted.

Note that in this contents graph, HTML pages are the characterized *contents*. Although *flights*, *hotels*, *clients*, etc. are the contents of the application from a data-modeling point of view, the HTML pages browsed by the user are the real contents of the application from Pipe's point of view. This is a *perverted* interpretation of Pipe, because in Pipe, the contents of the hypermedia applications are those heterogeneous units of information and their relations that are not easily representable in terms of classes/entities and their relations (e.g. a text that links with a sentence, and a sentence that links with the contextual meaning of their words [2]). In other words, because in Pipe there are no classes, individual elements are the contents of the application. The use of Pipe to characterize navigational maps of Web applications forces the interpretation of HTML pages as Pipe contents, being the real contents of the Web application represented in terms of UML class diagrams or entity-relationships [20] diagrams. These diagrams can be provided at conceptualization stage in order to complement Pipe diagrams and in order to fully characterize the contents of the travel agency.

In any case, Pipe contents are nearer to elementary items extracted from a data source and composed in different manners within pages than to pages displayed to users. Pipe notation is intended to be used at conceptualization stage. Therefore, in Pipe the content `requestOK` characterizes to the trips extracted from the database of trips according to customer's constraints. If the content `requestOK` has to include headers and footers with common information, a *frame-based approach* (see discussion in section 3.2 and 4.2) can be the best choice in Pipe. If the content `requestOK` has to include information dynamically extracted from different databases (possibly in different computers) the function *g* is the responsible of characterizing this process. For the sake of conciseness, the characterization of the function *g* is outside of the scope of this paper. In any case, such a characterization can be found in [3].

3.2 Navigational schema

Pipe navigational schema, characterizes the user interface of the application. Although the specification of the travel agency does not include the characterization of any user interface we are going to suppose a simple one. The application has a screen with two vertical regions. The left region is reserved to show the index of the application, while the right region is reserved to show the rest of contents of the application.

The terms *screen* and *region* are used instead of terms such as *frameset* and *frame* because, in our opinion, at conceptualization stage these details can be omitted. Indeed, during design this simple and abstract conception of the user interface could be refined, providing that the basic interaction behavior is preserved. For example, some designs can decide to use frameset and frames to represent screens and regions. Other designs can decide to aggregate the index to every content of the application (except the index to itself, of course), omitting the use of frameset/frames. Figure 4 depicts the Pipe characterization of this simple user interface.

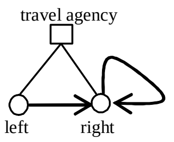


Figure 4. Navigational schema/user interface of the travel agency

According to this characterization, there is a window with two *panes* (or a screen with two regions). The arrow (*pipe*) between panes `left` and `right` means that the "destination contents" of the links with the origin in "displayed contents" of the `left` pane are displayed in the `right` pane. The arrow (pipe) between `right` pane and itself means that the "destination contents" of the links with the origin in "displayed contents" in `right` pane are displayed in the same pane.

We are aware that windows, panes and buttons are the basic elements of a graphical user interface. Regarding specific widgets (e.g. group or check boxes), Pipe does not provide explicit elements to characterize them. If wished, some visual content elements (e.g. group or check boxes elements) could be defined in order to be included in the container nodes of the user interface.

3.3 Canalization functions

Canalization functions represent the navigational interpretation of contents and relations in terms of the user interface of the application. Figure 5 depicts this information.

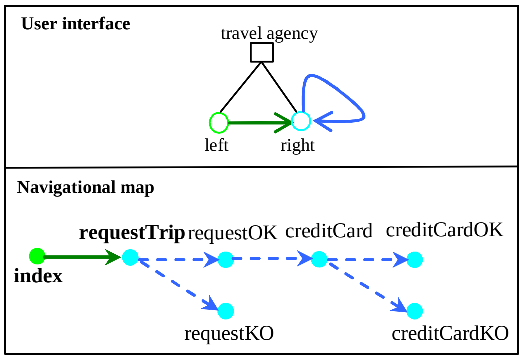


Figure 5. Relations between user interface and navigational map

Note that according to the colors of Figure 5, the content `index` is assigned to the `left` pane, while the rest of the contents are assigned to the `right` pane. In addition, the link between `index` and `requestTrip` is *canalized* by the pipe between `left` and `right`. The rest of the links are canalized by the pipe between the `right` and itself. Therefore, when the user selects the anchor in the content `index` (that is displayed in the `left` pane) that gives access to the content `requestTrip`, the content `requestTrip` appears in the `right` pane. In the same manner, if the user selects the anchor (i.e. *submit* button) in the content `requestTrip`, the content `requestOK` (or `requestKO`) appears in the `right` pane.

3.4 Browsing semantics

Browsing semantics describes the manner in which the information is to be visualized and presented [19]. Pipe includes a formalized browsing semantics [3] which describes applications' state, while the user is browsing.

Figure 6 describes travel agency's state, while the user is browsing it. Figure 6 (a) depicts the initial state where the context `index` is displayed in the `left` pane, and the content `requestTrip` is displayed in the `right` pane (the boldface in Figure 5 indicates that these contents are the default contents of these panes). Figure 6 (b) depicts the state of the application after the user requests a trip, and the application returns some options. Figure 6 (c) depicts the state of the application after the user decides to fill-in the form with the data of the credit card. Finally, Figure 6 (d) depicts the state of the application after the system validates the data of the credit card.

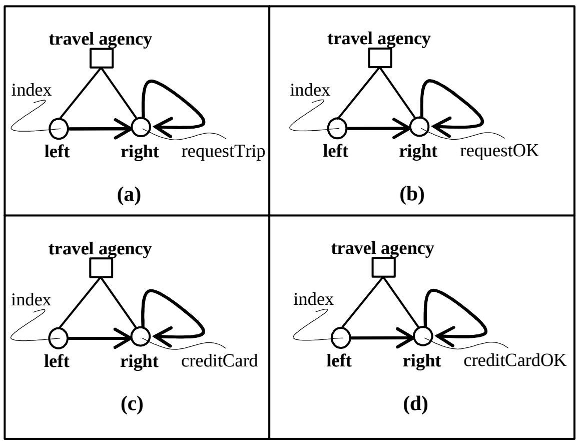


Figure 6. State of the application while being browsed

Note that the information provided in Figure 6, besides the information provided in Figure 5, can be used to indicate to the users the state of the application in terms of the navigational map and the user interface of the application.

4. Design of the travel agency

The design of the travel agency is represented in terms of the Conallen´s extension to the UML for Web applications [10]. The key point of Conallen´s extension is the principle of *separation of concerns*. According to this principle, there are in a Web application *client pages* (e.g. HTML pages) and *server pages* (e.g. JSP pages) that are represented in the UML by stereotyped classes and are related by navigated associations [10]. In addition, Conallen also provides a UML characterization of the structures of frames and framesets.

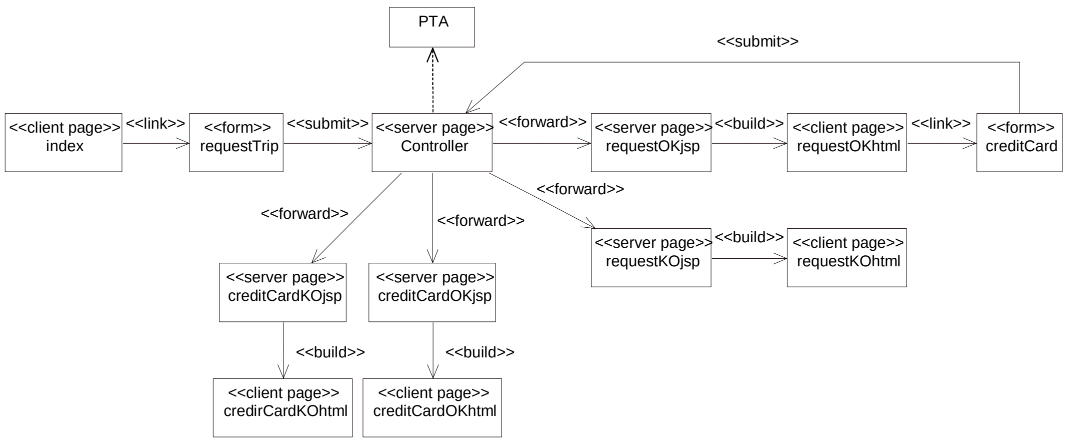

PTA
<<submit>>
<<client page>> index
<<link>>
<<form>> requestTrip
<<submit>>
<<server page>> Controller
<<forward>>
<<server page>> requestOKjsp
<<build>>
<<client page>> requestOKhtml
<<link>>
<<form>> creditCard
<<forward>>
<<forward>>
<<forward>>
<<server page>> requestKOjsp
<<build>>
<<client page>> requestKOhtml
<<server page>> creditCardKOjsp
<<server page>> creditCardOKjsp
<<build>>
<<build>>
<<client page>> credirCardKOhtml
<<client page>> creditCardOKhtml

Figure 7. UML-Conallen navigational map of the travel agency

This section provides the UML representation of the Pipe diagrams, taking into account the Conallen's extension.

4.1 Navigational map

Pipe content's graphs representing navigational maps can be smoothly mapped onto UML Conallen's extension during the design stage. Figure 7 shows the translation of the contents graph (Figure 1) for the present travel agency case study. Note that a *Model 2* (or *Model View Controller*) architecture [23] is chosen.

In this figure all the information, included in the specification about the computational behavior of the travel agency, is hidden within the class `PTA` (Personal Travel Assistant). In other words, the class `PTA` is a facade [5] of the application. For example, when the class `Controller` receives the information of the form `requestTrip`, the `Controller` delegates in the class `PTA` and, taking into account its response, the `Controller` forwards the control to the JSP pages `requestOKjsp` or `requestKOjsp` to generate the output for the user in terms of HTML pages (`requestOKhtml` or `requestKOhtml`). Note that the *Struts* API [26] could be used to implement this design. In addition, an *Enterprise Java Bean*-like [8] design could be used to implement the `PTA` facade and the rest of classes. Also, it should be noticed that the `PTA` facade gives the opportunity to model the actual data elements of the application and to incorporate the business logic associated with these elements. Because the paper is focused on navigational maps, the classes behind the facade are not provided. Although these classes constitute a substantial part of the application, their concrete structure and behavior is not relevant for characterizing the navigational part of this application.

4.2 User interface and final application

As previously mentioned, a frame-based or a non-frame based implementation of the regions could be provided at the design stage. If a frame-based implementation is chosen, Figure 8 describes its UML characterization in terms of Conallen's extension.

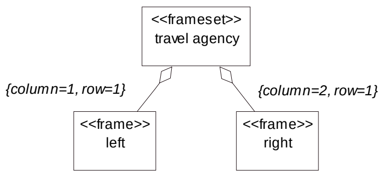


Figure 8. UML-Conallen user interface of the travel agency

In this figure, and in terms of Conallen's extension [10], the `left` frame could be omitted, since it is not a target of any link, but is included to provide a better characterization of the user interface. If a non-framed implementation is chosen, then all the classes (except the class `index`) must include an aggregation relation with the class `index`.

Finally, Figure 9 depicts the relations among the contents and the user interface, or in other words, the final application in terms of the notation defined in [10]. In this figure the Pipe canalization functions are used to assign contents to frames (e.g. content `index` is assigned to the frame `left`) and to interpret content links into the frame structure level (e.g. the link between `index` and `requestTrip` is targeted by the frame `right`).

5. Related work

At present, there exists an array of related work in the domains of hypermedia and Web engineering.

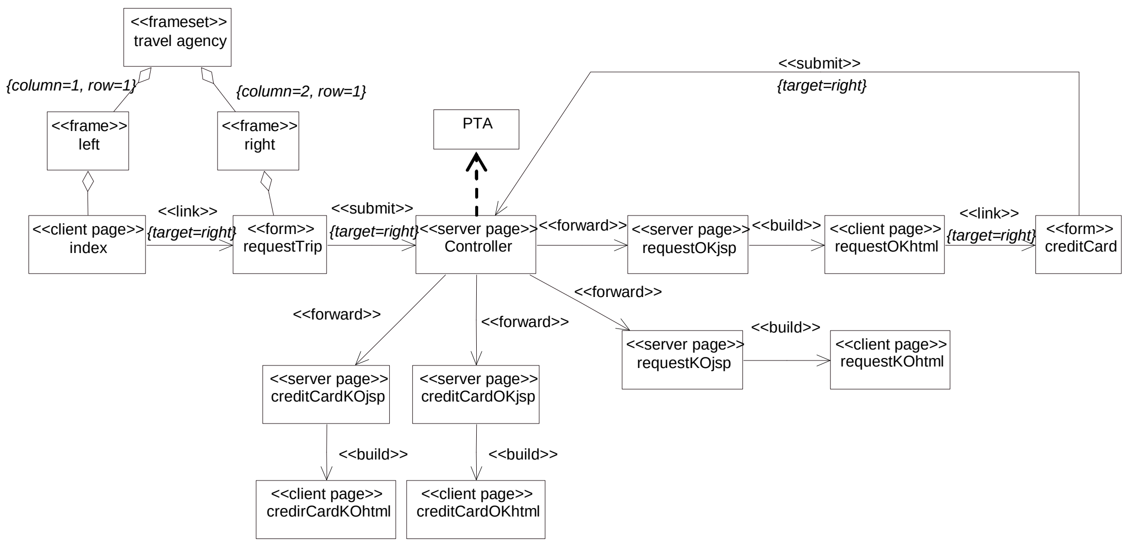


Figure 9. UML-Conallen design of the travel agency

Hypertext Design Model (HDM) [6], *Object-oriented Hypertext Design Model* (OOHDM) [4] and *Relationship Management Model* (RMM) [25], are some of the most relevant design notations in the hypermedia domain. All of them provide modeling primitives to characterize the classes or entities of the applications and their semantic relations. Because HDM and RMM characterize the applications from a data-model point of view none of them provides an explicit characterization of the navigational map of the application. OOHDM provides several modeling diagrams to characterize the navigational aspects of the application in terms of its classes and their relationships (i.e. navigation classes and navigational schema). In our opinion, the characterization of navigational maps in terms of classes and their relationships makes it harder for the users to visualize the navigational structure of the application since it requires the knowledge of object-oriented notations.

Object-oriented hypermedia (OO-H) [11], *UML-based Web Engineering* (UWE) [16], *Web Modeling Language* (WebML) [24], and *Web Site Design Method* (WSDM) [17] are some of the most relevant design notations in the Web engineering domain today. All these notations use some or other sort of diagram to characterize the navigational interpretation of the structural diagrams: OO-H uses *navigation access diagrams*, UWE uses *navigation diagrams*, WebML uses *navigation specifications* and WSDM uses *navigation tracks*. Because the modeling philosophy is similar in these approaches and in OOHDM (assimilating classes and relations), the conclusions derived for OOHDM are also applicable to these approaches.

Regarding Conallen's extension to UML, the use of stereotyped classes permits the presence of individual elements (e.g. the client page `requestTrip` of Figure 8) that can be used to characterize navigational maps for Web applications. In our opinion, Conallen's extension is well – suited for design, because it permits definition of the object-oriented behavior of dynamic applications, but is not especially suited for the conceptualization stage. Indeed, it is more complicated to use it for conceptualization, since it ties-in more this stage with the design stage than the Pipe does. In addition, in our opinion, the Pipe characterization of Figure 3 of the travel agency's navigational map is clearer than the navigational map that could be derived from the UML characterization of Figure 7 because: (i) it is not necessary the knowledge of object-oriented notations to understand Figure 3; (ii) Figure 3 hides architectural details included in Figure 7 (e.g. the presence of a Model 2 architecture); and (iii) Figure 3 hides technical details regarding the presence of client and server pages. Therefore, and as our experience has demonstrated, Pipe notation is simpler to understand by customers at the conceptualization stage [1][2][3].

Regarding Pipe notation scalability, in our opinion, it is similar to the scalability of visual notations (e.g. Conallen's extension). If a great amount of contents appear in the diagram, the separation in several subdiagrams can be the best choice.

Dialog Flow Notation (DFN) [12] and *State WebCharts* (SWC) [14] are focused on the characterization of the navigation in Web applications. DFN represents the dialog flow within an application as a directed graph of states connected by transitions, and SWC uses statecharts to describe the navigation between documents. To some extent, both approaches characterize Conallen's separation of concerns. DFN uses *masks* and *actions* [12] while SWC uses *static*, *transients* and *dynamic* states [14]. In Pipe the separation among client and server pages is not considered because the dynamic processing is hidden behind the generation function *g*. In addition DFN and SWC use the same diagram to characterize the structural and navigational models, while in Pipe these models are represented using contents graph and navigational schema diagrams.

6. Conclusions and future work

The characterization of navigational maps is a key issue during development of Web applications. Using navigational maps, developers can obtain a global view of the whole application that can help them during the development process. In addition, the presence of navigational maps can help users of Web sites to find the desired information quicker.

This paper presents a use of the Pipe notation to characterize navigational maps for Web applications (hypermedia or non-hypermedia). Pipe contents graph can be used to characterize the navigational map for Web applications at conceptualization stage, thus omitting computational design details included in Conallen's extension. In addition, Pipe contents graph permits the explicit definition of Web application's navigational map instead of deriving it from modeling primitives defined in terms of classes/entities and their relations.

Using Pipe navigational schema and canalization functions it is possible to provide a presentational characterization of the navigational map in terms of screens and regions. This characterization, besides the Pipe browsing semantics, could be used to characterize the application's state during its browsing.

Moreover, as demonstrated in this paper, the generation of UML-Conallen class diagrams using Pipe contents graph is a straightforward task. Although the translation from Pipe to UML diagrams presented in this paper does not include any complex detail, we are aware that a systematic and well-defined conversion procedure between Pipe and UML (in both directions) has to be defined. Indeed, at present, our main research effort is made in this direction.

Being focused on the representation of heterogeneous units of information and their heterogeneous relations, the Pipe notation lacks the modeling primitives for characterizing domains that can be easily represented in terms of classes/entities and their relations. For this purpose, UML class diagrams or entity-class diagrams can be used in combination with Pipe diagrams. In this paper we have omitted such a characterization because we have focused on the provision of a navigational map for the travel agency instead of characterizing its data and their relations.

Currently, we are developing a CASE tool to support the Pipe notation. Future work includes the automatic translation from Pipe notation to Conallen's extension (in both directions) to alleviate the transition from conceptualization to the design stage. In addition, we are working on the automatic generation of HTML pages and object-oriented codes from Pipe diagrams. UML-Conallen diagrams derived from Pipe diagrams could be used to generate such pages and codes.

7. Acknowledgements

The Spanish Committee of Science and Technology (TIC2001-1462, TIC2002-04067-C03-02 and TIN2004-08367-C02-02) has supported this work.

8. References

[1] A. Navarro, B. Fernández-Manjón, A. Fernández-Valmayor, and J.L Sierra, J.L. "Formal-Driven Conceptualization and Prototyping of Hypermedia Applications". In proceedings of *FASE 2002-ETAPS 2002* Grenoble, April, 2002, pp. 308-322.

[2] A. Navarro, B. Fernandez-Manjón, A. Fernández-Valmayor, and J.L. Sierra, "The PlumbingXJ Approach for Fast Prototyping of Web Applications". *Journal of Digital Information 5, 2, special issue on Information Design Models and Processes*, 2004.

[3] A. Navarro, A. Fernández-Valmayor, B. Fernandez, and J.L. Sierra, "Conceptualization, Prototyping and Process of Hypermedia Applications". *International Journal of Software Engineering and Knowledge Engineering 14, 6, special issue on Modeling and Development of Multimedia Systems*, 2004, pp. 565-602.

[4] D. Schwabe, G. Rossi, and S.D.J. Barbosa, "Systematic Hypermedia Application Design with OOHDM". In *Proceedings of the Hypertext 96,* Washington DC, USA, March, 1996, pp. 116-128.

[5] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[6] F. Garzotto, P. Paolini, D. and Schwabe, HDM. A Model-Based Approach to Hypertext Application Design. *ACM Transactions on Information Systems, 11, 1,* 1993, pp. 1-26.

[7] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

[8] Java Technology. Enterprise Java Beans Technology http://java.sun.com/products/ejb/index.jsp

[9] Java Technology. Java Server Pages Technology. http://java.sun.com/products/jsp/

[10] J. Conallen, "Modeling Web Application Architectures with UML". *Communications of the ACM 42, 10*, 1999, pp. 63-70.

[11] J. Gómez, C. Cachero, and O. Pastor, "Conceptual Modeling of Device-Independent Web Applications". *IEEE MultiMedia 8*, 2, 26-39.

[12] M. Book, and V. Gruhn. "Modeling Web-Based Dialog Flows for Automatic Dialog Control". In *Proceedings of the. 19th IEEE International Conference on Automated Software Engineering (ASE 2004)*, Linz, Austria, 2004, pp. 100-109.

[13] M. Thüring, J.M. Haake, and J. Hannemann, "What's Eliza Doing in the Chinese Room? Incoherent Hyperdocuments - and How to Avoid Them". In *Proceedings of the Hypertext 91*, Texas, 1991, pp. 161-177.

[14] M. Winckler, M., and P. Palanque. "StateWebCharts: a Formal Description Technique Dedicated to Navigation Modelling of Web Applications". In *Proceedings of the International Workshop on Design, Specification and Verification of Interactive Systems (DSVIS'2003)*, Funchal, 2003.

[15] N. Koch, and A. Kraus, "Towards a Common Metamodel for the Development of Web Applications". *In proceedings of International Conference on Web Engineering 2003, ICWE 2003*, 2003, pp. 497-506.

[16] N. Koch, H. Baumeister, R. Hennicker, and L. Mandel, "Extending UML to Model Navigation and Presentation in Web Applications". *In Modeling Web Applications in the UML Workshop, UML2000*, York, England, October 2000.

[17] O.M.F De Troyer, and C.J. Leune, "WSDM: A User Centered Design Method for Web Sites". *Computer Networks* 30, 1-7, pp. 85-94.

[18] P. Fraternali, "Tools and Approaches for Developing Data-Intensive Web Applications: A Survey". *ACM Computing Surveys 31, 3*, 1999, pp. 227-263.

[19] P.D. Stotts and R. Furuta, "Petri-Net-Based Hypertext: Document Structure with Browsing Semantics". *ACM Transactions on Office Information Systems, 7, 1, 1989*, pp. 3-29.

[20] P.P. Chen, "The Entity-Relationship Model - Toward a Unified View of Data". *ACM Transactions on Database Systems 1, 1*, 1976, pp. 9-36.

[21] R.C. Bodner, M.H. and Chignell, "Dynamic hypertext: querying and linking". *ACM Computing Surveys 31, 4es*, 1999.

[22] S. Abrahão and O. Pastor. "Measuring the Functional Size of Web applications" *International Journal of Web Engineering and Technology 1, 1, 2003*, pp. 5-16.

[23] S. Brown et al. *Professional JSP. 2nd Edition*. Wrox Press, 2001.

[24] S. Ceri, P. Fraternali, and A. Bongio, "Web Modeling Language (WebML): a modeling language for designing Web sites". *Computer Networks 33, 1-6*, 2000, pp. 137-157.

[25] T. Isakowitz, E.A. Stohr, and P. Balasubramanian, "RMM: A Methodology for Structured Hypermedia Design". *Communications of the ACM 38, 8*, 1994, pp. 34-44.

[26] The Apache Software Foundation. Struts. http://struts.apache.org/

[27] W3C. HTML 4.01 Specification, 1999. http://www.w3.org/TR/html4/

Atomic Use Case as a Concept to Support the MDE Approach to Web Application Development

Kinh Nguyen
Computer Science & Computer Engineering Department
La Trobe University,Australia
kinh.nguyen@latrobe.edu.au
Tharam Dillon
Faculty of Information Technology
University of Technology Sydney,Australia
tharam@it.uts.edu.au

Abstract

While use case is a popular technique for capturing requirements, the process of proceeding from use cases to subsequent modeling activities is to a large extent still unclear. In this paper, we propose the concept of atomic use case to formally model the functional requirements in support of the MDE approach to web application development. In particular we demonstrate how to construct a precise model for the business logic layer and to establish clear relationships between business logic model to other models such as the domain model, user interface model, navigation model, and the business process model. We also explore how the atomic use case concept can be incorporated into UWE (UML Web Engineering) Methodology.

1. Introduction

In this paper, we propose the concept of atomic use case as a fundamental concept to support the MDE (Model-Driven Engineering) approach to web-based application development. A web application, in particular a web information system, can be seen as having the following major components: (a) web-based components, (b) business logic and data source components, (c) other components such as web-services, portlets, agents, metadata, data mining, etc. We will be largely concerned with the first two kinds of components.

The concept of atomic use [7] is proposed as a solution to what we regard as a long-standing problem of information system, be it relational or object-oriented – namely the problem of analyzing and modeling the behavior of information systems.

One advantage of relational database technology, it is often claimed, is the separation of the static aspect and the dynamic aspect. The modeling of the static aspect (the structure of the database) has been handled in practice reasonably well. In contrast, the dynamic aspect (the operations of the database system,) is handled quite poorly. In the 70s, IFIP (The International Federation on Information Processing) recognized it as a key problem and organized a number conferences, inviting researchers to propose solutions to this problem. A number of proposals were made but none of them were adequate to the task, especially as a practical solution. Consequently, the problem gradually slipped into the phase of being a forgotten problem. Interest in the problem was briefly revived in the late 80s with the emergence of Z notation. Before long, it was realized that Z notation could not deliver the expected results, and the the problem slipped back into the state of a forgotten problem. In the mean time, the industry handles the problem in an informal (hence ambiguous) and ad hoc manner, which incurs huge hidden costs (poor understanding of functional requirements, poor specification for programmers, etc.).

As for object-oriented information systems, first of all, we can observe that UML without OCL would be too imprecise to describe behavior of information systems. An activity diagram can represent at a number of tasks and the flows between them; but it lacks the facilities to describe the tasks in detailed and precise terms. The same is true of sequence diagrams, which can show the sequence of messages in a collaboration, but are not good at describing the detailed effects of such messages. Though there are situations in which the sequence of messages alone can be very helpful (e.g. to explain how enterprise Java beans work, that is, how a client, through a number of stubs, can communicate with the beans on the server), it is not the case for information systems. For information systems, it is the effects of the messages that really tell us what really is going on. In practice, it is more often than not that after drawing pages and pages of sequence diagrams, when it comes to implementation, we have to ask a whole host of questions all over again. The gap between sequence diagrams and the implementation code is far too big in most cases. In the language of MDE, the model does not have sufficient information for the subsequent transformation act. To strengthen the UML models, OCL has been introduced. Since its inception, many important features have been added, and OCL now appears to have adequate expressive power for specifying complex applications. Given that is the case for OCL, an issue needs to be be seriously considered: how are we to use it? For example, what would be the units of behavior that we are going to use it for? And how are we going to identify these units in practice?

We propose atomic use case as an answer to the problem of precise behavior modeling. In this paper, we will introduce the concept and demonstrate how we can use it to construct a precise model for the business logic layer, and from there, to establish the clear relationships between business logic model and the user interface (of any kind). Finally, we explore how the atomic use case concept can be incorporated into UWE (UML Web Engineering) Methodology and briefly describe how we may apply this concept to the Travel Agency case study.

2. Introductory Example

We will take an example to show what atomic use cases are and the role they play through out the life-cycle. Because we wish to show the role this concept can play in the whole life-cycle, we will need to choose a very simple example.

Problem Statement Consider an application in which we are required to maintain information about a set of employees. Each employee has a unique ID, a name, and a phone number. The ID and name of an employee cannot be changed. System operations, or use cases, include: (1) Add an employee, (2) Delete an employee, (3) Change the phone number of an employee, (4) Retrieve employees by ID, by name, or by phone number.

2.1. Identify and Specify Atomic Use Cases

We can go through the use cases one by one, identify the atomic use cases, and formally specify them. Through this process, as will be shown, we end up with a complete model that captures the full functionality of the application.

Consider the operation (use case) to “Add an Employee.” A use case description for this operation can be as follows:

“To add a new employee, first, the user enters the id of the new employee. The system checks to determine if the id is new. If it isn’t, an error message will be displayed and the operation is terminated. Otherwise, the user enters next the

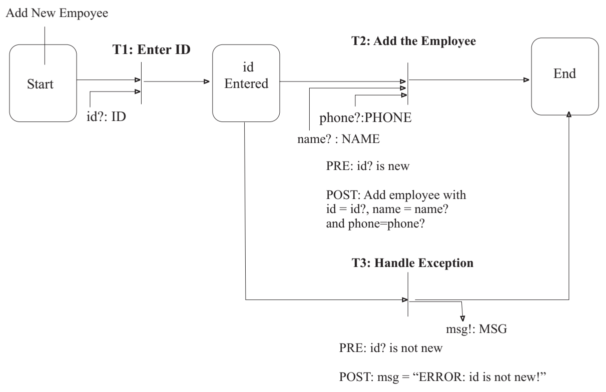


Figure 1. A Petri Net Representation for “Add an Employee” Use Case.

name and the phone number.The system then creates a new employee with the input data and saves it.”

The use case and the various path it can take can be represented by a high-level Petri net shown in Figure 1 (What we present here is a simple version of what we call the “obligation net” [8], a high-level Petri in which each transition has pre- and postconditions to precisely express the obligations the system has to fulfill.)

We now make a shift in the way we view the system’s behavior. Instead of thinking in terms of “do this then do that” or “if this, then do that” (the procedural view), we consider the operation as one whole unit and ask “For this use case, in how many different ways can the system makes its response?” (the declarative view). For the current use case, the system can respond in two ways, corresponding to two paths in the system obligation net.

The first path consists of transitions T1 and T2. Following this path, the system makes a *positive* response: it adds a new employee to the information base. The second path consists of transitions T1 and T3. In this case, the system makes a *do-nothing* response: it simply leaves the system in its prior state due to the non-fulfillment of the precondition for adding an employee. We refer to the response described by the first path as an *atomic use case*, and the one described by the second path as an *exception use case*. A general definition can be given as follows:

Definition: *An atomic use case is conceived as an* indivisible *response by the system that either (1) effects a change of the system’s state, which takes the system from a consistent state to a consistent state, to reflect an event taking place in the application domain, or (2) performs a query that is of interest to the user in its own right.*

Having identified the atomic use case, and in keeping with the “atomic” viewpoint, we can specify it informally in terms of the input, output, pre- and postconditions as follows:

– Input: id?, name?, phone?

– Output: None

– Pre: id? is new

– Post: Create a new employee with id?, name?, phone? and add the employee set

We can now seek to specify the atomic use case formally. As expected, the specification always follows a consistent format which consists of input, output, pre- and postconditions, though some of these elements may be absent in a particular case. The "Add Employee" atomic use case can be specified as follows:

$\boxed{\mathit{AddNewEmployee}}$

input

$id? : ID$

$name? : NAME$

$phone? : PHONE$

pre

$id? \notin \{e : allEmployees \bullet p.id\}$

post

$$\begin{aligned}
&\exists\, newEmployee : allEmployees' \bullet \\
&\quad newEmployee \notin allEmployees \\
&\quad newEmployee.id = id? \\
&\quad newEmployee.name = name? \\
&\quad newEmployee.phone = phone? \\
&\quad allEmployees' = allEmployees \\
&\qquad\qquad \cup \{newEmployee\}
\end{aligned}$$

Two points should be made from this example. First, in order to precisely specify the use case, we need to use some formal notation. In the above example, we have used Object-Z [3]. We could use OCL instead. Both use the same mathematical concepts and we can easily translate from one into the other (except for some advanced operators of Object-Z). Object-Z is more concise and we use it here to save space.

Second, we have assumed the existence of two classes (only the static features are required at this stage). One is the *Employee* class, which has three attributes as shown below:

Employee

$id : ID$

$name : NAME$

$phone : PHONE$

...

The other is a system class. It represents the system from the functional view point. It maintains a set of employees and provides method to manage that set of employees. We will call the class *EmployeeApp*:

EmployeeApp

$allEmployees : \mathbb{P}\, Employee$

$\forall\, e1, e2 \bullet e1 \neq e2 \Leftrightarrow e1.id \neq e2.id$

...

In this example, the *Employee* class is actually the whole of the domain model. In general, to formally specify the atomic use case, we need (a) the domain model or part of the domain model relevant to the use case, and (b) the system class. One crucial aspect of the system class is that through the attributes of the system object we can get to all the domain objects of the system.

By going through the use cases one by one, and identifying and formally specifying the atomic use case for each of them, we would obtain a collection of atomic use case specifications, which constitutes *a complete functional requirements model* of the application.

2.2. Derive Methods for Domain and System Classes to Obtain a Complete Business Logic Model

Once an atomic use case is formally specified, we can deduce the methods required of the domain classes and the system class to support that use case.

Each atomic use case will become a method of the system class. For the "Add a New Employee" use case, we require the method shown below in the system class:

EmployeeApp

AddEmployee

$id? : ID$

$name? : NAME$

$phone? : PHONE$

$$\begin{aligned}
&id? \notin \{e : allEmployees \bullet e.id\} \\
&\exists\, e : Employee \bullet \\
&\quad e \notin allEmployees \\
&\quad e.INIT(id?/id?, name?/name?, phone?/phone?) \\
&\quad allEmployees' = allEmployee \cup \{e\}
\end{aligned}$$

...

From the postcondition of that method of the system class, it is clear that we need to be able to create new *Employee* instances. Thus, we have identified a method (constructor) required of the domain class to support the use case. The method is expressed in Object-Z as an *INIT* schema as shown below:

```
┌─ Employee ─────────────────────────
│ ┌─ INIT ───────────────────────
│ │ id? : ID
│ │ name? : NAME
│ │ phone? : PHONE
│ ├──────────
│ │ id = id?
│ │ name = name?
│ │ phone = phone?
│ └──────────────────────────────
│ ...
└────────────────────────────────────
```

By repeating this activity for all atomic use cases, we would get a *complete business logic layer model* as far as the functional requirements are concerned. In general, this model (which strictly speaking is not a design model) consists of a system class and all the domain classes relevant to the particular application.

2.3. Implement and Test the Business Logic Model

Having derived the methods required of the system and domain classes, we can implement or prototype them. An implementation for the "Add New Employee" use case is shown below. We need one method for the system class:

```
public void addNewEmployee( String id,
String name, String phone) throws Exception
{
  // compute the precondition.
  // Assume that collectIds() returns the set of
  // ids of the set of employees
  boolean pre =
      allEmployees.collectIds().contains(id);

  // if precondition is not satisfied,
  // abort the operation
  if (! pre)
  {   throw new Exception
          ( "The ID already exists!" );
  }

  // otherwise, create the new employee and add
  // it to the information base
  Employee newEmployee =
       new Employee(id, name, phone);
  allEmployees.add( newEmployee );
}
```

and one method for the domain class *Employee*:

```
public Employee(String id, String name,
String phone)
{
    this.id = id;
    this.name = name;
    this.phone = phone;
}
```

Note that we have all the details from the business logic model we need to do that. No further discovery activities are needed.

Testing Once the two methods above are available, we can test them with a testing script such as the one shown below. The variable name *theSystem* is to emphasize the fact that an instance of *EmployeeApp* represents the complete system in this example.

```
// create a system object
EmployeeApp theSystem = new EmployeeApp();

// add first employee and display the system's
// state
theSystem.addNewEmployee("E10","Smith","1234");
System.out.println( theSystem.toString() );

// add second employee and display the
// system's state
theSystem.addNewEmployee("E20","Adams","2345");
System.out.println( theSystem.toString() );

// try to add employee with an existing id
// and observe that the system's state remains
// the same
theSystem.addNewEmployee("E10","Clarke","3456");
System.out.println( theSystem.toString() );
```

The testing script contains several test cases (scenarios of the current use case), which are based on the precondition specified in the atomic use case. For each test, the state of the system object is displayed to verify that the implementation satisfies the postcondition specified in the atomic use case.

The tests reveal two important consequences of the implementation of the atomic use case.

- First, we can perform the required operation, i.e. adding employees.
- Second, and just as important, the system can protect itself from invalid requests and preserves the integrity of its state.

The ability to make appropriate responses to both valid and invalid requests is, of course, exactly what we should be looking for. Notice that we can achieve all of these by considering *only* the atomic use cases.

The Functional Core Once we proceed to implement the rest of the operations specified in the business logic model, the two classes that we obtain constitute an executable component that allows us to perform all the required operations (add and delete employees, change phone numbers, etc.). We call this the *functional core*. It is a "basic core of the system" that is fully functional in the sense that it can store the relevant information, update the information, and respond to queries in support of the business activities. For this reason, we take the functional core to be the *business logic layer* of the system. That is, in our approach, we take the business logic layer to be precisely the implementation of the identified *atomic* use cases.

Moreover, once the functional core has been fully tested, we can build the desired graphical user interface as a separate layer on top of it. As will be shown, it is possible to build a separate GUI layer that interacts with the functional core only at a small number of well-defined points.

3. Further Clarification on the Concept of Atomic Use Case

In the definition of atomic use case, given earlier, the criterion that an update atomic use case must *"take the system from a consistent state to a consistent state"* is significant and is useful to identify atomic use cases. The following simple example illustrates this point.

Example - Enroll Student Consider the case of enrolling students in subjects. Suppose subjects are classified as *core* or *optional*, and each student must take at least 3 core subjects. Without the condition that a student must take at least 3 core subjects, the act of enrolling a student in a subject is an atomic use case. With that condition in place, that act is no longer an atomic use case: it may cause the information base to be in an inconsistent state. The atomic use case in this case must be "To enroll a student in a set of subjects in one go". More precisely,
– The inputs are a student id and a set of subject codes
– The outputs: NONE (it is clearer not to regard error messages as output; they are implied by the preconditions)
– The preconditions are: (1) the id must exist, (2) each unit must exists, (3) the set contains at least three core unit
– Postcondition: Enroll the student in those subjects.

Similarly, the phrase *"to reflect an event taking place in the application domain"* provides a useful criterion for identifying atomic use case. The following example illustrates this point.

Example - Add Student or Staff Consider an application which deals with students and staff in an academic institution (suppose we maintain some different information about them). "Add a Student" and "Add a Staff" are atomic use cases. In one of our presentations, it has been asked: Should we take "Add a Person" as an atomic use case? The answer is "No". In the application domain, we may have the event of "Having a new student" or "Having a new staff member", but not the event "Having a new person". "Person' is an abstraction with some information left out of student or staff, and the so-called event "Having a New Person" cannot fully describe the situation. Furthermore, it we consider the states of the information base, we can see this clearer: When the information base change from state S to state S', then S' may be S plus information about a new student or a new staff, but *not* simply about a new person.

Finally, when the use case is a query use case it needs to be *"of interest to the user in its own right."* The following example serves to illustrate this point.

Example - Redistribution Parts between Warehouses In [4], Jacobson presents a rather sophisticated screen to show how the user may interact with a system to redistributes parts among various warehouses (we move items from the 'From' warehouse to the 'To' warehouses). There is a drop-down list to select the 'From' warehouse. When a 'From' warehouse is selected by the user, the system respond by listing the rest of the warehouses as potential 'To' warehouses. In making this response, the system would need to perform a query against the information base. Now, it is unlikely that such a query is of interest to a user by itself (in this example, it serves as a small step in determining the potential 'To' warehouse and where in that warehouse we should move items to). If that is the case, the query does not amount to an atomic use case. It is simply a query that supports the user interface.

In our approach, we would extend the functional core to provide the user interface-support queries. The functional core and the extended part together are called the *extended function core*

4. Relationships to Use case Descriptions Graphical User Interfaces

Use cases can be given at different grains of granuality. Three main levels are usually distinguished, and using the terminology of [2],they are: summary goal level (business use cases), user goal level (system use cases), and subfuctions (subfunction use cases). Use case descriptions are usually given in three general formats: the simple unstructured format, the user-system dialog format, the flows of events format. Given a description of a use case, regardless of its format, we can identify the atomic use case associated with it. As shown earlier, one way to do this is to sketch a net like one in Figure 1 and observe how the system responds to various paths (some lead to atomic use cases, some do not). Very often, we can even recognize the atomic use cases directly: they normally correspond to the main flows of the system use cases.

Identifying atomic use cases through use case descriptions is not the only option. In fact, it is more practical to do so through the graphical user interface. In the industry, people are less likely to talk about use cases; they often talk about user interface and how the user interacts with the user interface. We will be talking the 'language of the industry' when we identify the atomic use cases through the user interface sketches (or designs) and the descriptions of how they work.

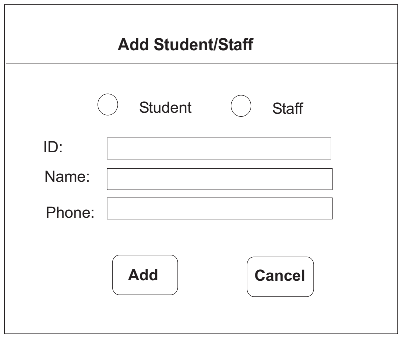


Figure 2. Screen to Add Student or Staff.

Example - Add Staff or Student Consider the screen in Figure 2.

– When the user checks either the 'Student' or the 'Staff' checkbox, no query is made to the information base. The user interface simply 'remembers' that the user has made that choice.

– When we enter id, the system would respond by checking whether the id is new or not, and warn us if it's not new. This action requires the system to make a query to the underlying information base. The purpose of this query is to support the user interface.

– After the user has entered relevant information and press button "ADD", the system will respond by calling the business layer to add a student or a staff member. Thus, out of this screen we have two atomic use cases: one to add a new staff and one to add a new student.

In general, given a screen or a series of screens, web-based or otherwise, many events can be generated by the user's actions. By examining how the system responds to each of these events, in particular how it interacts with the information base, we can identify atomic use cases and all the queries needed to support the operation of the user interface.

5. Transforming Functional Core Model to Business Logic Layer

The implementation of the functional core model, in essence, requires the translation of formal expressions for pre- and postconditions into programming code. To do that, in most cases we only need to handle a small number of common expressions. These expressions are are list in Figure 3, which show the equivalents in Smalltalk. Equivalent Java code segments would be less concise, but definitely standard patterns can easily be established.

In the language of MDE, (a) we have a formal platform independent model of the business logic layer, and (b) we can readily formulate transformation rules to transform the model into a platform-specific component.

$\forall x : X \bullet p(x)$	`X size = (X select: [:p| p(x)]) size`
$\exists x : X \bullet p(x)$	`(X select: [:P| p(x)]) isEmpty not`
$x \in X$; $x \notin X$	`x in: X (or X includes: x )`
$x \notin X$	`x notIn: X (or (X includes: x) not)`
$X \cup \{a\}$	`X add: a`
$X \setminus \{a\}$	`X remove: a`
$X \cup Y$	`X union: Y`
$X \cap Y$	`X intersect: Y`
$X \setminus Y$	`X removeAll: Y`
$\{x : X \mid p(x)\}$	`X select: [ :x | p(x) ]`
$\{x : X \bullet e(x)\}$	`X collect: [ :x | e(x) ]`
$\exists x : X \bullet p(x) \blacktriangleright x.oper$	`x := X detect: [ :x | p(x) ].` `x oper`

(Note that `in:aCollection` can be defined as a method in the class `Object` as `aCollection includes: self`)

Figure 3. Equivalent Expressions between Formal Specification and Smalltalk

EJB Implementation We can implement each atomic use case as a session bean, which may access data sources in a distributed manner. In this case, the system class is implemented implicitly as a collection of session beans.

However, in most cases, it is better to put all the atomic use cases in one session bean, one method for each atomic use case. In this case, the system class is implemented explicitly. Conceptually, such an implementation would be much clearer and easier to understand. Practically, it would make it easier for the client programs to locate and use the beans on the server-side: there is only one bean that the client should be aware of.

Note It is not unusual to break up an application into a small number of major components. In such a case, the "system class" is implicitly made up of these components and each atomic use case is to be described in the context of a particular component. Also, we must specify the interactions among the components in terms of the messages they can send to each other.

Relational Implementation It is not uncommon to implement the information base as a relational database and to have the business logic code written in a non-object-oriented fashion (e.g. using JDBC instead of Hibernate, say). Then how would we proceed from the specification to the implementation?

In this case, we need to convert the object-oriented model into a relational model. That is, we convert the class diagram into a relational schema, and each atomic use case must be expressed as a method that acts on the relational schema. The first task (to obtain the relational schema) is rather straight forward and many mapping rules have been suggested. The second task (to capture the behavior), at the intuitive level is very straight forward. Moreover, we only need to be concerned with a small number of expressions (that are used to express the pre- and postconditions) presented earlier.

Alternatively, we could choose to write the behavioral specification based on the relational model from the outset. In this case, standard Z notation is not quite suitable for practical use (as witnessed by history). We have experimented with an extension of Z, which we call 'RZ', 'R' for 'relational'. It does not have (yet) a formal semantics, though intuitively the meanings of the additional constructs are clear. As a kind of 'formal pseudo code', RZ works really well (quite unambiguously) in expressing the intentions of the modeler. In addition, for the consumption of those who do not want to use mathematical notations, we have added a few features to SQL to form a pseudo code language to express the atomic use cases (i.e. their pre- and postconditions) in a highly (but not formally) precise manner. We are currently investigating the use of an object-oriented specification language (such as Object-Z, OCL) for specifying pre- and postconditions against a relational schema by representing tuples of a relation as simple objects where all the attributes are publicly accessible for manipulation. The use of Maude [1] could be quite suitable for this task as well.

6. Atomic Use Cases and Events in Web Applications

The relationships between the atomic use cases and the events generated by graphical user interfaces (considered earlier) indicate how atomic use cases can be applied to the analysis, modeling and specification of web applications. An event generated by the user's actions on a web-based interface can be conceived as a contract (to be fulfilled in most cases by the method that handles the event on the server side). This contract consists of

– The input data (they normally come from the HTML forms or cookies or the session object);

– The output (in the contract for a web event, we define the output to be data that we need to pass to the returned web page – see example below);

– The precondition (to specify the conditions that the input data must satisfy);

– The postcondition (to specify actions such as updating of the cookies or session object, performing query or update action against the business logic layer, i.e. invoking atomic use cases);

– The returned page (the page is divided into one or more "blocks", and for each block, we specify the input data that it receives and displays and the events it may generate; for each event, we specify its parameters).

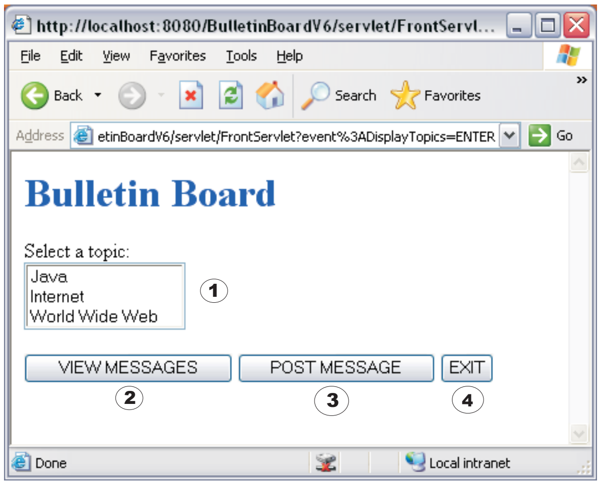
The content of such a contract for a web event can be formally specified. As an example, consider the screen for an online bulletin board shown in Figure 4. Several events can be generated by this screen and the specification for the "View Messages" event, for example, is shown below (the boolean expressions are in OCL):

```
Event ViewMessages
Input:
   // topic? is the name of the selected topic
   Request topic?: String
Output:
   // set of messages to be displayed by the
   // returned web page
   messages!: Set(String)
   // the topic needs to be passed to the
   // returned web page
   topic!
Pre:
   // name of selected topic must exist in the
   // information base
   bulletinBoardFnCore.
       getTopics.name ->includes(topic?)
Post:
   // retrieve the messages and pass the topic
   // along to the returned web page
   messages! =
       bulletinBoardFnCore.
           getMessages(topic?).text
   topic! = topic?
Returned Page:
   Block 1:
       Input:
           message! : Set(String)
           topic!: String
       Next Events:
           DisplayTopics(Session topic: String)
               where topic = topic!
```

Notice that (a) In the Post section, a call is made to the business logic layer; and (b) The returned page contains the Display Topics event whose argument is the topic received by the View Messages event by virtue of two constraints: `topic! = topic?` in the Post section, and `topic = topic!` in the Next Events subsection of the Returned Page section.

7. Exploring the Application of Atomic Use Cases to UWE

UWE (UML-based Web Engineering) [6] is a methodology for web application development with ArgoUWE [5] as


http://localhost:8080/BulletinBoardV6/servlet/FrontServl...
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
etinBoardV6/servlet/FrontServlet?event%3ADisplayTopics=ENTER
Go
Bulletin Board
Select a topic:
Java
Internet
World Wide Web
1
VIEW MESSAGES
POST MESSAGE
EXIT
2
3
4
Done
Local intranet

1 selection list of topics obtained from topics!
2 event: View Messages
3 event: Post Message
4 event: Exit

Figure 4. Layout for Display Topics Event

a supporting CASE tool. One of the motivations for UWE is to provide support for complex business processes. Let us explore how we can apply the concept of atomic use case to UWE.

Example The example of an e-shop given in [5]. This example illustrates the basic steps and the basic models of UWE. The example shows four basic UWE models: the conceptual model, the navigation model, the (business) process structure model, and the process flow model. Part of the navigation model for the e-shop example is reproduced in Figure 5. It shows the navigation nodes, process nodes, navigation links (between navigation nodes), and process links (between navigation nodes and process nodes). One of the process nodes is the Checkout node, whose details are shown in the process flow model in Figure 6.

Analysis of the Checkout Process From the atomic use case point of view, we would perceive this Checkout node as consisting of two "aspects" which are associated with two different kinds of concerns: the first is about an atomic use case and the second is about the user interface design to support this atomic use case.

The atomic use case is the one that takes details about an order submitted by the user (such as customer name, address, credit card number, whether they want the items wrapped, etc.) and accordingly updates the underlying information base of the e-shop. We can call this the "Take Order" atomic use case (or "Place Order" from the customer's viewpoint).

With the Take Order atomic use case in mind, we can see that the essential role of the Checkout process model in Figure 6 (possibly apart from the "send invoice" action) is to obtain the details needed as input for the Take Order atomic use case. This is the second aspect we mentioned above. Viewing it this way, a question arises: How should we model this second aspect?

Instead of the process model in Figure 6, we could model the second aspect by a graphical user interface. For example, we can have a screen with two screen "blocks" that the user can interact with simultaneously:

– One block to confirm the items in the shopping cart and to select the wrapping options and

– One block to set the payment method.

The screen also has buttons to cancel or to place order. Such a screen would allow us to get the information we need, but it does not constraint us to the flow pattern described in the Checkout process flow in Figure 6. The process flow model is an over-specification: It is one way to get information to support the atomic use case but it is not the only one.

Possible Modifications to the UWE Models For the above Checkout process, it could be argued that the crucial point is the content of the information, rather how we obtain it. That is, the actual process in this case seems to play a secondary role.

For that reason, we could introduce to the UWE navigation model a new type of process node (through the stereotype mechanism) to represent processes with characteristics similar to those of the Checkout process above. We may call them "Use Case Process Nodes". A use case process node can be modeled by

– An activity diagram with a branching fork which has a number of alternatives: one of which leads to a "cancel", and each of the rest leads to an atomic use case (which represents the point where we update the information base)

– The specifications of the atomic use cases (we normally would have them already)

– A screen (which can be taken to be a part of presentation model) to indicate how the user can interact with the system to effect the atomic use case.

The suggested specification appears to be much simpler than what we currently have in UWE, and it avoids the problem of over-specification (which in most cases contains an element of arbitrariness on the part of the modeler).

By introducing the Use Case Process Node and by modeling it the way suggested above, we may bring about several advantages. First, there may be many instances of such processing nodes in an application, and it is clearer conceptually to single them out (to distinguish them from other kinds of processing nodes). Second, having singled them out, we can model them in a standard way as suggested above, which would simplify the specifying process.

Incidentally, another advantage is brought about by the act of modeling the atomic use cases itself. With current

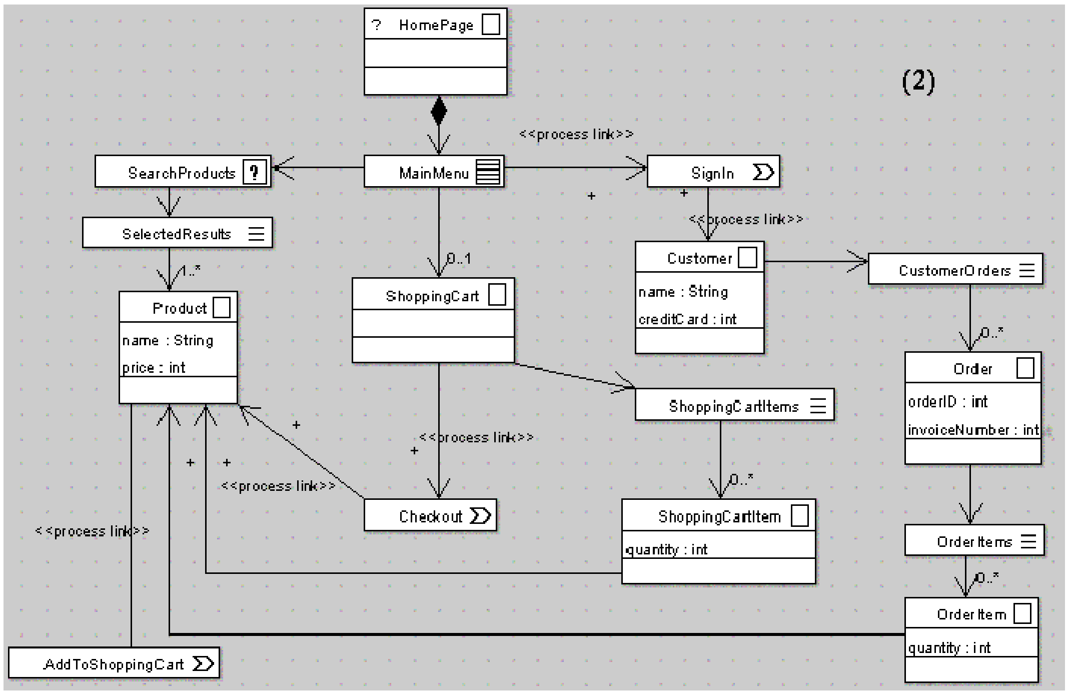


Figure 5. Navigation Model for the e-shop. Source: Knapp et al. [5]

UML modeling tools, the atomic use cases can be specified (more or less completely) with the help of OCL, and code (e.g. Java) can be generated out of the model. This generated code can quickly be enhanced to be a working prototype. This feature could be very attractive as a means to induce the average software engineer to apply OCL for rigorous modeling, which is an essential part of the MDE approach.

The admission of atomic use cases into the UWE model may open up a way to classify the typical business processes into various kinds of different grains of granuality. Some would be the use case process nodes described above. Some may represent external processes, which may be modeled by specifying the inputs and outputs. Still some may involve interactions with the user or external sources, which would require many of the UWE features for modeling.

8. Atomic Use Cases and the Travel Agency Case Study

Consider the Travel Agency Case Study described in [9]. We will give a sketch as to how one may approach this case study using atomic use cases. We will view the application as made up of the following types of components: the Personal Agent Assistant, the Broker Agent, the Transportation Company and the Finance Company.

The Personal Agent Assistant (or Assistant for short) is responsible for processing requests from the customers. To process a request, the Assistant maintains a list of Broker Agents, a list of Financial Companies, and data about the customer bookings and payment details. It would also need to store the current request and the list of current offers. We can think of the Assistant as a finite-state machine whose actions, which take place when transitions occur (some of which may be triggered by time constraints), would be the following atomic use cases:

– Enter Request (to store a user's request). Input: A request; Output: None; Pre: The request is valid; Post: Store the request.

– Get Offers (to send messages to Broker Agents to request offers) Input: The request data; Output: A list of offers; Pre: None; Post: Store the list of offers. (Note that we cannot impose the condition that the offers match the requirements of the request, though we do expect that to be the case.)

– Present Offers (to take the offers from the previous use case and to make the valid offers available to the user). Input: The set of offers (denoted by "InOffers"); Output: A set of offers (denoted by "OutOffers"); Pre: The request id exists; Post: OutOffers = the set of offers in InOffers that

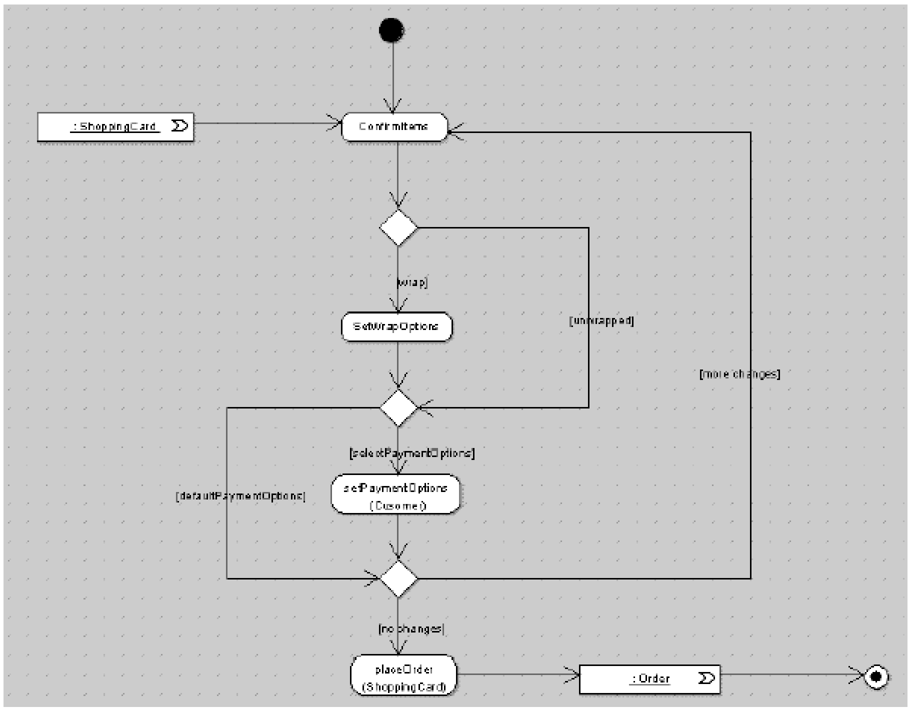


Figure 6. The Checkout Process Model. Source: Knapp et al. [5]

match the request.

– Make Booking (to take a booking (or bookings) on behalf of the user). Input: The offer identification, payment details; Output: Booking confirmation messages; Pre: The selected offer is one of the valid offers; Post: Validate payment details with Financial Company AND send messages to confirm the bookings for the selected offer and cancel the rest.

– Cancel Request (to cancel the request). Input: None; Output: None; Pre: None; Post: Delete current request and and cancel all offers

The Broker Agent would maintain a list of Transportation Companies. The main atomic use case it has are:

– Provide Offers (to take a request from the Assistant and to respond with a list of offer). Input: A request; Output: A list of offers; Pre: None; Post: The offers match the request AND store the offers.

– Confirm Booking (to confirm the booking of an offer which were previously temporarily booked). Input: The selected offer identification; Output: confirmation messages; Pre: The selected offer is in the current list of offers; Post: Send message to confirm bookings for selected offer and cancel the rest.

For the Transportation Company and the Finance Company, essentially we need to define the interface and the assumptions about the contracts related to the messages.

Having identified those atomic use cases, we can proceed to specify them formally, and construct the conceptual model (perhaps in the process of formalizing the atomic use cases), the navigation model, and the business process model.

9. Conclusion

In this paper, we have introduced the concept of atomic use case. It is a natural concept and therefore easy to grasp. In fact, it has appeared in various guises in the systems development literature. However, to really benefit from it, we need to have a clear understanding of the concept. Toward this end, we provided a definition (one that aims to assist us in identifying the atomic use cases). We showed how we can identify the atomic use cases and specify them, and how we can build a complete business layer with them. Finally, we explored how we can incorporate the concept into UWE for web application development, which could lead to a more definite choice of granularity for the process nodes and clearer relationships between the various models.

References

[1] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Comput. Sci.* , 285:187-243, Sept. 1995.

[2] Alistaire Cockburn *Writing Effective Use Cases*, Addison-Wesley, 2001.

[3] R. Duke and G. Rose (2000) *Formal Object-Oriented Specification Using Object-Z*, MacMillan.

[4] Jacobson J., Ericsson M. and Jacobson P. (1992) *Object-Oriented Software Engineering: A Use Case Driven Approach*, Wokingham:Addison-Wesley.

[5] Alexander Knapp, Nora Koch, Gefei Zhang, and Hanns-Martin Hassler. Modeling Business Processes in Web Applications with ArgoUWE, In *7th International Conference on the Unified Modeling Language (UML2004),* LNCS 3273, 69-83, Springer Verlag, October 2004.

[6] Nora Koch and Andreas Kraus. The expressive Power of UML-based Web Engineering. In *Second International Workshop on Web-oriented Software Technology (IWWOST02),* D. Schwabe, O. Pastor, G. Rossi, and L. Olsina, editors, CYTED, 105-119, June 2002.

[7] Kinh Nguyen, Tharam Dillon, Atomic Use Case: A Concept for Precise Modelling of Object-Oriented Information Systems, in *OOIS'03, The Ninth International Conference on Object-Oriented Information Systems,* Geneva, Switzerland, 2003

[8] Kinh Nguyen. *A Semi-Formal Object-Oriented Method for Analysis and Modelling of the Functional Requirements of Information Systems*, PhD Thesis in Computer Science, La Trobe University, 2003.

[9] A Travel Agency System, Case Study for Workshop on model-driven Web Engineering (MDWE 2005), http://www.lcc.uma.es/ av/mdwe2005/TheTAEexample