A Model-Based Approach for Integrating Third Party Systems with Web Applications

Nathalie Moreno and Antonio Vallecillo

Dpto. de Lenguajes y Ciencias de la Computación
Universidad de Málaga, Spain
{vergara,av}@lcc.uma.es

Abstract. New Web applications are rapidly moving from stand-alone systems to distributed applications that need to interoperate with third party systems, such as external Web services or legacy applications. In most cases, this integration is not properly addressed by current Web Engineering proposals, that either achieve it only at one single level (user interface, process, code or data), or assume the existence of a central conceptual model, something which is not always true in a service-oriented scenario. This paper presents a model-based framework that provides concepts and mechanisms for facilitating the high-level integration of Web applications with third party systems, allowing the manipulation of the external entities of such systems as native elements of our models.

1 Introduction

We are currently witnessing an evolution of Web applications, which are rapidly moving from stand-alone systems to distributed applications that need to interoperate with third party systems, such as external portlets, Web services or legacy applications.

In most cases, this integration is not properly addressed by existing Web Engineering proposals (such as WebML [1], OOHDM [2], UWE [3], OO-H [4], W2000 [5] or WSDM [6]). They provide excellent methodologies and tools for the design and development of Web applications. However, they have also shown some limitations when external and legacy systems need to be integrated into the applications they build. For instance, they either achieve this integration only at one single level (user interface, process, code, or data – see e.g., [7]) or assume the existence of a central model (usually called the conceptual or structural model), around which the rest of their models are built. However, the existence of such a central model is not always true, as it happens, for instance, in service-oriented scenarios where each party can have its own data schema. Finally, most of these Web proposals do not supply mechanisms for making explicit their provided and required interfaces, their processes, choreographies, data models, and business logic. Such a sort of "componentization" of Web application development is required in order to properly exchange information and services with other systems.

Integration problems also depend on the kind of external systems being considered. Web services have become the Internet standard for exchanging functionality between loosely coupled clients and servers. Portlets [8] allow the exchange of presentation, on top of basic functionality, for Web portals construction. Finally, legacy software have been defined as "software that works". Although usually poorly documented, difficult to maintain, and complex to adapt and integrate with other systems, this kind of applications cannot be ignored because they support core business functions, embed business rules not documented anywhere else, and preserve the strategy and finances of most large organizations.

A proper integration approach requires an structured an efficient way to assist software architects and developers achieve it not only at implementation level, but also during all phases of the development process.

This paper presents a model-based framework that provides the concepts and mechanisms required for facilitating the high-level integration of Web applications with third party systems, allowing the manipulation of the external entities of such systems as native elements of our models. It is based on the MDA principles [9, 10], that aim at providing portability, interoperability and reusability through architectural separation of concerns. MDA prescribes certain kinds of models to be used (i.e, CIM, PIM and PSM), how those models may be prepared, and the relationships between the different kinds of models.

The basis for prescribing these models is the concept of viewpoint, where a viewpoint on a system is a technique for abstraction using a selected set of concepts and structuring rules, in order to focus on particular concerns within that system. In this regard, our framework identifies a set of models related to the development of a Web application, each one addressing a key concern.

The rest of the paper is structured as follows. Section 2 briefly introduces the framework, identifies its main layers and models, and defines guidelines for using the viewpoints prescribed by the framework for structuring Web applications development. Section 3 details how to use the framework for facilitating integration of Web applications with third party systems, using a classical example to illustrate the approach. Finally, Section 4 draws some conclusions and outlines some future research work.

2 A Framework for Building Web Applications Within the MDA Context

Our framework tries to help organize complex systems by separating the different concerns that matter for model-driven Web application development. As depicted in Figure 1, it is organized in three main independent layers, each one corresponding to a *viewpoint*. Consequently, a Web application in our approach distinguishes three main PIMs, one for each layer (User interface, Business logic, and Data). These PIMs comprise a set of models defined in terms of the entities that are relevant to the corresponding concerns, and the relationships between the models.

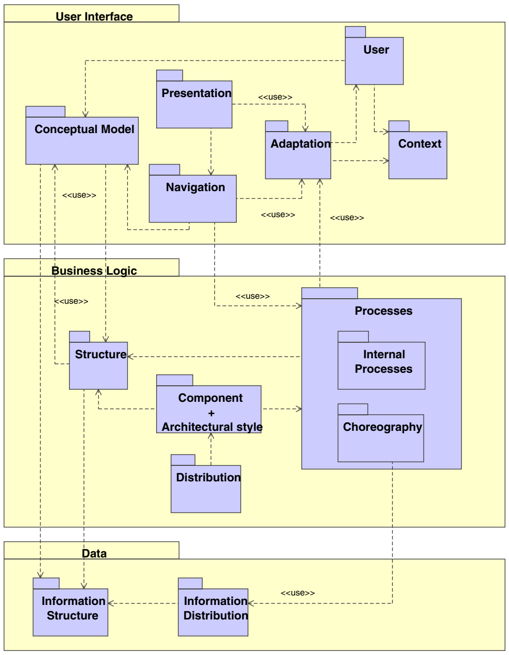

Fig. 1. Models representing the different concerns involved in the development of a Web application

2.1 The Data Viewpoint

The **Data structure** level describes the organization of the persistent information managed by the application (that could be finally stored in, e.g., a relational database). Information is represented in terms of the data elements that constitute its information base and the semantic relationships between them. This level is organized in two models:

- The ***Information Structure*** model deals with the information that has to be made persistent before it gets stored in a database.
- The ***Information Distribution*** model describes the distribution and replication of the data being modeled, since information can be fragmented in *Nodes* or replicated in different *Locations*.

2.2 The User Interface Viewpoint

The **User interface** level is responsible for accepting persistent, processed or structured data from the *Process* and *Data* viewpoints, in order to interact with the end user and deliver the application contents in a suitable format. Originally, Web applications were specifically conceived to deal mainly with navigation and presentation concerns, but currently they also need to address other relevant issues:

- The ***Conceptual*** model encapsulates the information handled by the rest of the models at this level.
- The ***Navigation*** model represents the application navigational requirements in terms of *Access Structures* that can be accessed via *Navigational Links*.
- Navigational objects are not directly perceived by the user, rather they are accessed via the ***Presentation*** model. This model captures the presentational requirements in terms of a set of *PresentationUnits*.
- The ***User*** model describes and manages the user characteristics with the purpose of adapting the content and the presentation to the users' needs and preferences.
- The ***Context*** model deals with *Device*, *Network*, *Location* and *Time* aspects, and describes the environment of the application. These are needed to determine how to achieve the required customization.
- The ***Adaptation*** model captures context features and user preferences to obtain the appropriate Web content characteristics (e.g., the number of embedded objects in the Web page, the dimension of the base-Web page without components, or the total dimension of the embedded components). Adaptation policies are usually specified in terms of ECA rules.

2.3 The Business Logic Viewpoint

The **Business Logic** level encapsulates the application's business logic, i.e., how the information is processed, and how the application interacts with other computerized systems.

- 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 system, 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. For a complete description of a business process, apart from the *Structure* model, we need information related to the *Activities* carried out by the *BusinessProcessInformation*, expressing their behaviour and the *Flows* that pass around objects or data.
- The ***Choreography*** model defines the valid sequences of messages and interactions that the different objects of the system may exchange [11]. The choreography may be individually oriented, specifying the contract a component exhibits to other components (*PartialChoreography*) or, it may be globally oriented, specifying the flow of messages within a global composition (*GlobalChoreography*).
- The ***Distribution*** model describes how its basic entities, the *Nodes*, are connected by means of point to point connections or *Links*. While the *Information Distribution* model of the *Data* viewpoint specifies the distribution of

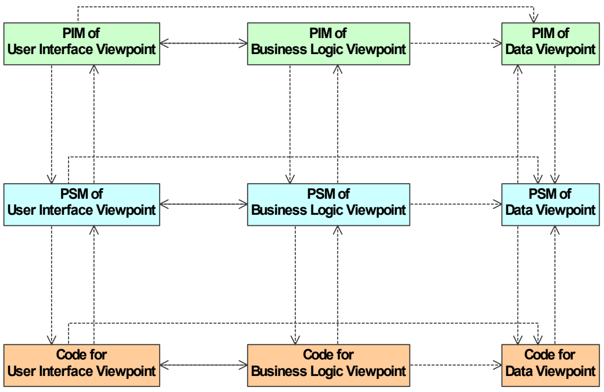
PIM of User Interface Viewpoint
PIM of Business Logic Viewpoint
PIM of Data Viewpoint
PSM of User Interface Viewpoint
PSM of Business Logic Viewpoint
PSM of Data Viewpoint
Code for User Interface Viewpoint
Code for Business Logic Viewpoint
Code for Data Viewpoint

Fig. 2. PIMs required in the development of a Web applications, and their transformation to code (adapted from [13])

the data, this model describes the distribution of the processes that achieve the business logic of the system.

- The ***Architectural Style*** model defines the fundamental organization of a system in terms of its components, their relationships, and the principles guiding its design and evolution [12], i.e., how functionality is encapsulated into business components and services.

2.4 How the Framework Is Used

The development of a typical Web application with data, business process and hypertext requirements involves the definition of at least three PIMs, each one corresponding to a *viewpoint* (see top of Figure 2). Looking at the different viewpoints, we realized that they can be naturally integrated using the MDA architecture and its facilities for relating them.

The process begins by determining the scope of the system, by means of identifying the framework metamodels that need to be instantiated. For data-intensive Web applications, the *Data viewpoint* modeling is the cornerstone for all the other viewpoints. In consequence, our starting point is an *Information Structure* model of the application, which is represented in our proposal by a UML class diagram marked with the stereotypes defined in Table 1. The *Information Structure* model is then refined to represent how information is distributed in ≪Nodes≫, with attribute Location specifying their location. As shown in Table 1, external data sources will be referenced in the *Information Structure* model as ≪ExternalInformationUnit≫ UML classes, related to ≪InformationUnit≫ UML classes by means of ≪ExternalRelationship≫ associations.

Please notice that the fact that stereotypes of UML 2.0 Profiles may have associated attributes is very useful [14]. They add information to the stereotypes when marking the models – allowing us, for instance, to include information about the Location of a ≪Node≫, the kind and characteristics of an external association marked with ≪ExternalRelationship≫, etc.

Once the PIM of the *Data* viewpoint has been defined, the behavioral details are given in the *Structure* model of the *Business Logic* layer. This model captures the Web application functionality. It can be modeled in many different ways, including, e.g., preconditions, postconditions or invariants, or using any other notation that allows the semi-automatic generation of implementations (e.g., an Action Semantic Language). When external systems are used, they will be included in the model as ≪ExternalBusinessProcessInformation≫ UML classes. Their relationships with our ≪BusinessProcessInformation≫ UML classes will be represented by ≪ExternalAssociation≫ UML associations. Again, the use of attributes in stereotypes will allow us to represent (at a later stage) the kind of external services and the way to invoke them, e.g., using SOAP in case of Web services, JSR-168 or WSRP in case of portlets, etc.

Integration at this level is the most interesting but also the most difficult, because it may require interoperability between processes that may not have common a behavioral model (e.g., action semantics or granularity).

At this point, the system functionality can be refined into two additional models: the *Internal Process* model and the *Choreography* specification. Both of them are UML activity diagrams marked with the stereotypes defined in Table 1. Notice that the ≪ExternalActivity≫ and ≪ExternalFlow≫ marks allow to handle external business process and workflows as elements of our models.

The issues related to the architectural style (i.e., how processes are encapsulated into ≪Components≫) and process distribution (e.g., how and where components are deployed) complete the description of the *Business Logic* layer in marked UML component diagrams. Finally, the PIM of the *Business Logic viewpoint* is obtained by merging the previous models into a single one that contains all the information. In this regard, not only class, activity or component diagrams can be marked, but also UML deployment diagrams. The stereotypes ≪StaticNode≫, ≪ComputingNode≫, ≪MobileNode≫ and ≪ExternalNode≫ have been defined to represent system artifacts as nodes, which are connected through communication paths (≪Link≫ and ≪ExternalLink≫) to create network systems. Nodes can be either hardware devices (≪Device≫), places (≪Place≫), actors (≪Actor≫) or software execution environments (≪ComputingNode≫).

Finally, the description of how the information is displayed to the user starts with the definition of the *Conceptual* model of the *User Interface* viewpoint. Two main stereotypes distinguish between internal information objects that require to be manipulated at presentation level (≪UserKnowledgeUnit≫), and information objects that are created by external entities and that need to be displayed by our application (≪ExternalUserKnowledgeUnit≫) – as it happens, for instance, with portlets. Then, the *Conceptual* model is enriched with *Access Structures*, that determine how to reach the information objects using ≪Indexes≫, ≪Menus≫ or ≪GuidedTours≫. Finally, in order to obtain the PIM of the *User Interface* viewpoint, we need to provide for each ≪UserKnowledgeUnit≫ UML class and for each *Access Structure*, a graphical representation using the stereotypes defined in Table 1.

Table 1. Framework Models Entities

Level	**Model**	**Concepts**
Data	Information Structure	InformationUnit, ExternalInformationUnit, Attribute, Relationship, ExternalRelationship, AccessOperation, Parameter, Constraint, Transaction
	Information Location	Node, ExternalNode, Link, ExternalLink, Location
User Interface	Conceptual	UserKnowledgeUnit, ExternalUserKnowledgeUnit, Attribute, Association, Generalization, Specialization, Aggregation, Perspective, Dependency Relationship
	Navigation	NavigationUnit, NavigationLink, ExternalNavigationUnit, ExternalNavigationLink, LandmarkNavigationUnit, ContextNavigation, Event, Access Structure (Indexes, Menus, Guided Tour), OptionMenu, IndexOption
	Presentation	PresentationUnit, SinglePresentationUnit (Text, Image, Input element, Text area, Selection element, etc.), GroupPresentationUnit (Section, Page, Form,ExternalPage, etc.), Tabbed
	Context	Context, Device, Network, Location, Time
	User	History, Session, User, Role, UserFeature, Preference, Previous Knowledge
	Adaptation	Event, Rule, Condition, Action, Entity
Business Logic	Structure	BusinessProcessInformation, Attributes, Signature, Association, ExternalBusinessProcessInformation, ExternalAssociation
	Internal Processes	Activity, ExternalActivity, ActivityType, StartActivity, EndActivity, Flow, ExternalFlow, ControlStructure, ConditionalStructure
	Choreography	PartialChoreography, GlobalChoreography, Activity, Transition, AtomicActivity, ComplexActivity, ExecutionMode, Exception, ControlStructure, ConditionalStructure, Connection
	Architecture	Component, Module, Layer, Sub-module, Client, Server, Master, Slave, Pipe, Filter, Broker, Peer, Event bus, Sources, Channel, Bus, Listener, Model, View, Controller, Adapter, Interpreter
	Distribution	Node, StaticNode (Device, Place, Actor), ComputingNode, MobileNode, System, ExternalNode, Network, Link, ExternalLink

Depending on whether adaptation is required or not, a *User* model, a *Context* model and the adaptation rules of the *Adaptation* model need to be specified. These three models are merged to complete the *User Interface* PIM specification.

Once the three PIMs are appropriately marked, we just have to follow the MDA transformation process from PIMs to PSMs, applying a set of mapping rules (one for each mark and for each marked element). The result of the application of such mapping rules are a set of class diagrams marked according to the target technologies (e.g. Java, JSP, Oracle, etc.). Finally, the PSMs are translated to code applying a transformation process again (see Figure 2). As mentioned in [13], special care should be taken with the bridges between the three PIMs and their corresponding PSMs, for which transformations are also required.

3 Proof of Concept: The Travel Agency System

In order to illustrate our proposal, this section describes how to design a Travel Agency Web application that interacts with both customers through a Web interface, and with external service providers using Web services. The Travel Agency sells vacation packages to its customers. The packages include flights, hotel rooms, car rentals, and combinations of these. External service providers include transportation companies (airlines, hotels and car rentals) and financial organizations (credit companies and banks).

To book a vacation package, the customer will provide details about his preferred dates, destinations, and accommodation options to the Travel Agency System (TAS). Based on this information, the TAS will request its service providers for offers that fulfill the user's requirements, and then will present the list of offers to the customer. At this point, the customer may either select one of the offered packages, reject them all and quit, or refine his requirements and start the process again. If the customer selects one of the packages, the TAS will book the individual services to the corresponding transportation companies, and charge the customer. Based on these requirements, there is no need for persistent storage or for adaptation in this case. Thus, only two PIMs are required: the *User Interface* PIM and the *Business Logic* PIM.

The PIM of the *User Interface* viewpoint is developed starting from its *Conceptual* model. For service-oriented scenarios, it has to be defined from scratch taking into account the information that needs to be presented to the user during a session. Therefore, our *Conceptual* model will consist of seven ≪UserInformationUnits≫ UML classes, namely TravelAgency, HolidayPackage, Booking, Customer, Flight, Room and Car (see Fig. 3). Please note how the three latter classes have been added to the model to store temporarily returned values of invoked external services, and to handle the transactions. Besides, these entities will act as bridges between the Travel Agency Interface and the external Web services interfaces (marked as ≪ExternalUserInformationUnits≫ in the *Conceptual* model).

Then, the *Navigation* model is built as a refinement of the *Conceptual* model. 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). With this purpose, each ≪UserInformationUnit≫ in the *Conceptual* model has been mapped to one or more ≪NavigationUnits≫ in the *Navigation* model, in the same way as *Associations* of the *Conceptual* model have been mapped to ≪NavigationLinks≫.

To reference the external points of navigation outside the scope of the application, ≪ExternalUserInformationUnit≫ UML classes (Airline, CarHire or Hotel) have been marked as ≪ExternalNavigationUnit≫ in the *Navigation* model. Likewise, the relationships between internal and external ≪NavigationUnits≫ have been marked as ≪ExternalNavigationLinks≫ in this model. 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.)

The *Presentation* model further refers to groups of pages organized around ≪PresentationUnits≫ as: (*i*) ≪SinglePresentationUnits≫, with their attributes

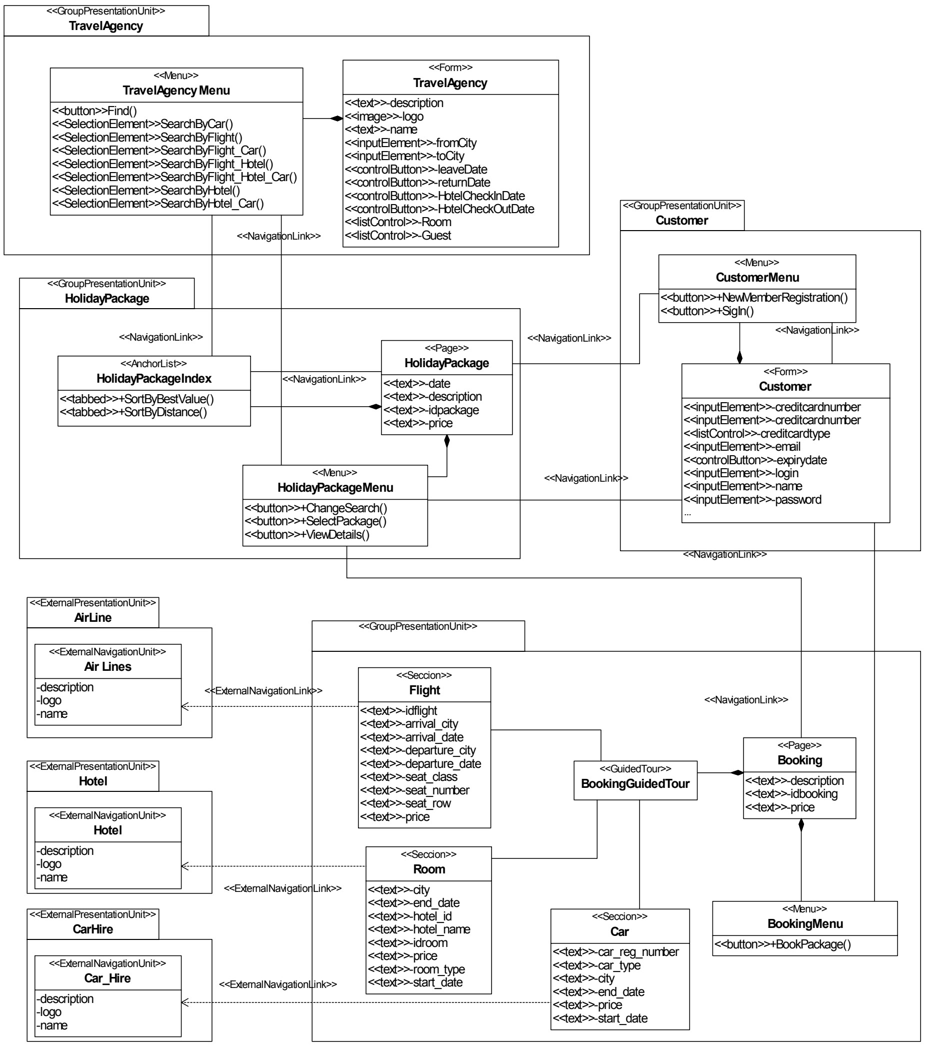

Fig. 3. The PIM for the User Interface viewpoint

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 ≪section≫ to display service responses and ≪page≫ to display the main portal pages. We have also marked as ≪forms≫ those UML classes that invoke external services.

Since adaptation is not required in this case, the final PIM of the *User Interface* viewpoint (shown in Figure 3) is obtained by merging these three models.

The PIM for the *Business Logic* layer can be developed in parallel starting from the *Structure* model. As shown in Figure 4, it involves component types representing internal system services (≪BusinessProcessInformation≫), their

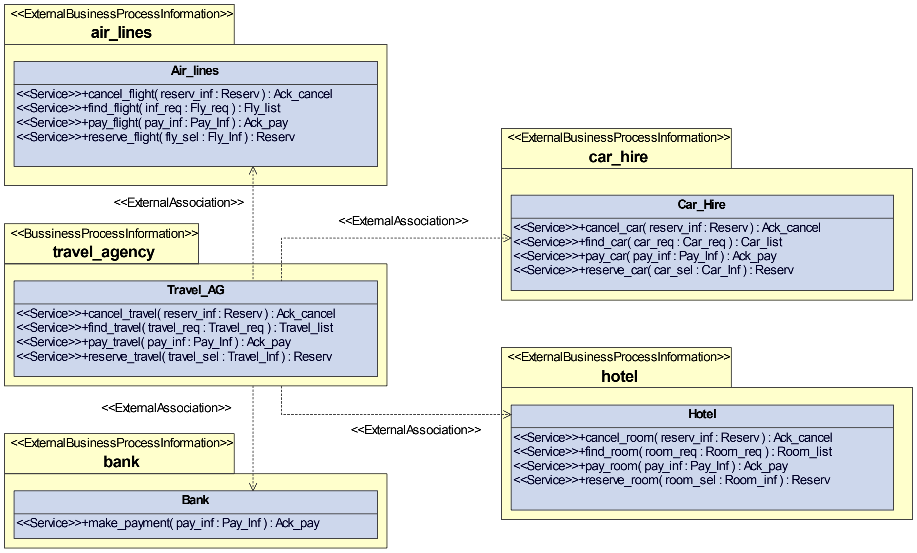
<<ExternalBusinessProcessInformation>>
air_lines
Air_lines
<<Service>>+cancel_flight(reserv_inf : Reserv) : Ack_cancel
<<Service>>+find_flight(inf_req : Fly_req) : Fly_list
<<Service>>+pay_flight(pay_inf : Pay_Inf) : Ack_pay
<<Service>>+reserve_flight(fly_sel : Fly_Inf) : Reserv
<<ExternalAssociation>>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

Fig. 4. Business Logic Structure Model

attributes (≪Attributes≫), the signature of their operations (≪Services≫), and the relationships between them (≪Associations≫). External system interfaces have been modeled as ≪ExternalBusinessProcessInformation≫ and their relationships with other ≪BusinessProcessInformation≫ processes as ≪ExternalAssociations≫.

Please notice that we can easily model these external services by making use of their actual specifications (i.e., interfaces, choreography, behavior) if we know them. Alternatively, we can always model their abstract (i.e., required) specifications, and then use adaptors at the PSM level when the actual services are decided [15].

The next step consists of providing a specific description of how internal processes (marked as ≪Services≫) find_travel, reserve_travel, pay_travel and cancel_travel work. This is described in the *Internal Process* model (not shown here). Then, the interactions between internal and external processes are detailed in the *Choreography* model (see Figure 5). Notice that both the *Internal Process* and the *Choreography* models are UML activity diagrams marked with the stereotypes defined in Table 1.

Finally, the *Component* and *Distribution* models (in this order) will reveal how the internal processes are grouped according to the software architecture style, and encapsulated for distribution in the final platform.

In addition to the *User Interface* and the *Business Logic* PIMs, the bridges between them (represented by dependency relationships in our framework) need to be specified. OCL restrictions and dependency relationships between elements from different models are used for that. For example, *events* in *Navigation* model establish when *activities* in the *Internal Process* model can start or finish.

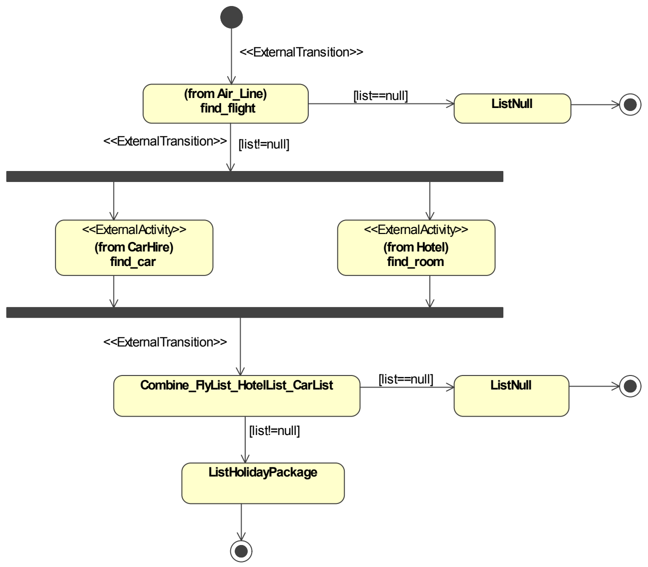

Fig. 5. An excerpt of the choreography for find_travel

Once the PIMs of our application are described, we need to generate their respective PSMs. Using a set of model transformations (as detailed in [13]), we can obtain a Java PSM for the *Business Logic* PIM and a Web presentation (e.g. a WAE PSM) for the *User Interface* PIM, for example. Then, another set of transformations is required (from PSMs to code) to produce the final code and Web pages of the application – but these transformations are already well-known and documented (see, e.g., [16]).

4 Concluding Remarks

Web applications are no longer isolated system, they need to interoperate with other external systems. Model-driven Web engineering proposals should take this fact into account, being able to incorporate these external applications into their models. In this paper we have presented a model-based framework that allows the high-level integration of Web applications with third party systems, enabling the manipulation of the external entities of such systems as other elements of our models.

Once the framework has been defined, there are some issues that need to be addressed. First, the specification of the bridges between the different PIMs have to be improved so precise correspondences between their elements can be established. Second, we plan to make use of QVT languages for defining the model transformations, so they can be easily re-used and integrated into an MDA tool. Finally, we plan to work on the interoperability issues that arise when (not necessarily compatible) external applications need to be smoothly integrated.

Acknowledgements

The authors would like to thank anonymous referees for their helpful comments and remarks. This work has been supported by Spanish Research Project TIC2002- 04309-C02-02.

References

1. Ceri, S., Fraternali, P., Bongio, A.: Web modelling language (WebML): a modelling language for designing web sites. Proc. of WWW9 (2000) Amsterdam, Netherlands.
2. Rossi, G., Schwabe, D., Guimaraes, R.: Designing personalized web applications. Proc. of WWW10 (2001) 275–284 Hong Kong, China.
3. Koch, N., Kraus., A.: Towards a common metamodel for the development of Web applications. Proc. of ICWE'03 **LNCS 2722** (2003) 495–506 Berlin.
4. Garrigós, I., Gómez, J., Cachero., C.: Modelling Dynamic Personalization in Web Applications. Proc. of ICWE'03 **LNCS 2722** (2003) 472–475 Oviedo, Spain.
5. Baresi, L., Garzotto, F., Paolini, P.: Extending UML for Modelling Web Applications. Proc. of HICSS-34 (2001)
6. Troyer, O.D., Leune, C.: WSDM: A User-Centered Design Method for Web Sites. Proc. of WWW07 (1998) 85–94
7. Brambilla, M., Ceri, S., Comai, S., Fraternali, P., Manolescu, I.: Model-driven specification of web services composition and integration with data-intensive web applications. IEEE Data Eng. Bull. **25** (2002) 53–59
8. Bellas, F., Fernández, D., Muiño, A.: A flexible framework for engineering "my" portals. Proc. of WWW13 (2004) 234–243
9. Miller, J., Mukerji, J.: MDA Guide. Object Management Group. (2003) OMG document `ab/2003-06-01`.
10. OMG: Model Driven Architecture. A Technical Perspective. Object Management Group. (2001) OMG document `ab/2001-01-01`.
11. OMG: A UML Profile for Enterprise Distributed Object Computing. Object Management Group. (2002) OMG document `PTC/2002-02-05`.
12. IEEE: Recommened Practice for Architectural Description of Software-Intensive Systems. IEEE Std. 1471-2000. (2000)
13. Kleppe, A., Warmer, J., Bast, W.: MDA Explained. The Model Driven Architecture: Practice and Promise. Addison-Wesley (2003)
14. Fuentes, L., Vallecillo, A.: An introduction to UML profiles. UPGRADE, The European Journal for the Informatics Professional **5** (2004) 5–13
15. Moreno, N., Vallecillo, A.: What to we do with re-use in MDA? Second European Workshop on Model Driven Architecture (EWMDA-2) (2004) Canterbury, Kent.
16. Bézivin, J., Hammoudi, S., Lopes, D., Jouault, F.: Applying MDA approach for Web service platform. In: Proc. of EDOC 2004, Monterey, California, IEEE CS Press (2004) 58–70