

# Coping with Requirements Changes: SWS-challenge phase II

Marco Brambilla<sup>1</sup>, Stefano Ceri<sup>1</sup>, Dario Cerizza<sup>2</sup>, Emanuele Della Valle<sup>2</sup>,  
Federico Facca<sup>1</sup>, Christina Tziviskou<sup>1</sup>

<sup>1</sup> Dipartimento di Elettronica e Informazione, Politecnico, 20133 Milano, Italy  
{mbrambil, ceri, facca, tzivisko}@elet.polimi.it

<sup>2</sup> CEFRIEL, 20133 Milano, Italy  
{cerizza, dellava}@cefriel.it

**Abstract.** In this paper we describe our approach to the second phase of SWS Challenge 2006. We present the WebML design of the mediator and its implementation by the means of the CASE tool WebRatio; furthermore, we discuss the integration of Glue WSMO discovery engine in our approach. The integration is achieved by invoking the Web Services exposed by Glue both for publishing and for discovering semantic services; in particular, we show the publishing of a new shipment/purchase service and the discovery of the most convenient shipment/purchase service. Finally, we present in detail the changes we applied to the system in order to model the additional requirements introduced in the second phase of the Challenge.

## 1 Introduction

One of the most engaging promises of Semantic Web Services is to enable the construction of flexible business applications, spanning over several enterprises and capable of dynamic composition, keeping the applications up to date with respect to continuously changing requirements. The second phase of the SWS Challenge addresses these problems, by asking its participants to effectively and quickly react to changes of the application requirements of the first phase. For coping with the challenge, we propose a model-driven methodology to design and develop semantic Web service applications and their components, described according to the emerging WSMO initiative. In particular, we show the advantages of a top-down approach that leverages software engineering methods and tools, such as: formalized development processes, component-based and visual software design techniques, and computer-aided software design. This approach leads from an abstract view of the business needs to the application concrete implementation.

Our method is based on existing models for the specification of business processes (such as BPMN) combined with Web engineering models for designing semantically rich Web applications (such as WebML) implementing Service Oriented Architectures with the support of Semantic Web Service tools (such as WSMX and WSMT). For the needs of service discovery and reasoning, we integrate our design methodol-

ogy with a discovery engine called Glue [5]. By combining the Semantic Web with Software Engineering methods, the efforts for building a solution migrate from the programming level up to the design level; thus, the whole design process becomes much more efficient, and the major advantages occur as a consequence of fast requirements evolution. In this paper, we demonstrate that the new requirements (both for the mediation and discovery aspects) can be mastered at the model level, and that software can be generated from the modified models without requiring low-level programming of the solution.

The paper is structured as follows: Section 2 presents the employed technologies; Section 3 describes the initial modeling of the two scenarios of mediation and discovery proposed by the Challenge; in Section 4 we summarize the changes that are needed for facing the new requirements in the two scenarios; and finally Section 5 draws some conclusions and presents the ongoing and future work.

## 2 Technologies employed

**WebML/WebRatio WS edition.** WebML language [3, 6] is a high-level notation for data- and process- centric Web applications. It allows specifying the conceptual modeling of Web applications built on top of a data schema used to describe the application data, and composed of one or more hypertexts used to publish the underlying data.

The WebML data model is the standard Entity-Relationship (E-R) model. Upon the same data model, it is possible to define different hypertexts (e.g., for different types of users or for different publishing devices), called *site views*. A site view is a graph of *pages*, allowing users from the corresponding group to perform their specific activities. Pages consist of connected *units*, representing at a conceptual level atomic pieces of homogeneous information to be published: the content that a unit displays is extracted from an entity, and selected by means of a *selector*, testing complex logical conditions over the unit's entity. Units within a Web site are often related to each other thru *links* carrying data from a unit to another, to allow the computation of the hypertext. WebML allows specifying also update *operations* on the underlying data (e.g., the creation, modification and deletion of instances of an entity, or the creation and deletion of instances of a relationship) or operations performing other actions (e.g. send an e-mail). In [1] the language has been extended with operations supporting process specifications.

To describe Web services interactions, WebML has been extended with Web service units [2, 4]. These units are synthetically represented in Figure 1. Web services operation symbols correspond to the WSDL classes of Web service operations. In the definition of the icons, we adopt two simple graphical conventions: (i) two-messages operations are represented as round-trip arrows; (ii) arrows from left to right correspond to input messages from the perspective of the service (i.e., messages sent by the Web application). *Request-response* and *response* operations are triggered when the user navigates one of their input links; from the context transferred by these links, a message is composed, and then sent to a remote service as a request. In the case of a synchronous request-response, the user waits until the response message is received,

then continues navigation as indicated by the operation's output link. Otherwise, navigation resumes immediately after the message is sent. *Solicit* and *one-way* are instead triggered by the reception of a message. Indeed, these units represent the publishing of a Web service, which is exposed and can be invoked by third party applications. In the case of one-way, the WebML specification may dictate the way in which the response is built and sent to the invoker. Moreover, Web services publishing units cannot have output links leading to pages, because there is no user interaction involved in the response to the caller.



**Fig. 1.** Web services operations

The language is *extensible*, allowing for the definition of customized operations and units. It has been implemented in a prototype that extends the CASE tool WebRatio [7], a development environment for the visual specification of Web applications and the automatic generation of code for the J2EE and Microsoft .NET platforms. The design environment is equipped with a code generator that deploys the specified application and Web services in the J2EE platform, by automatically generating all the necessary pieces of code, including data extraction queries, Web service calls, data mapping logics, page templates, and WSDL service descriptors.

**Glue WSMO discovery Engine.** Glue [5] is a WSMO compliant discovery engine that provides the basis for introducing discovery in a variety of applications that are easy to use for requesters, and that provides efficient pre-filtering of relevant services and accurate discovery of services that fulfill a given requester goal.

In conceiving Glue the model for WSMO Web Service discovery was refined explicating the central role of mediation:

- by making the notion of class of goals and class of Web Service descriptions explicit;
- by using ggMediators for automatically generating a set of goals semantically equivalent to the one expressed by the requester but expressed with a different form or using different ontologies;
- by making wgMediators the conceptual element responsible for evaluating the matching;
- by using ooMediators for solving any terminological mismatch that can appear with different polarized ontologies for the domains; and
- by redefining the discovery mechanism as a composite procedure where the discovery of the appropriate mediators and the discovery of the appropriate services is combined.

Moreover in designing Glue the authors refined WSMX Discovery Engine architecture according to their refined WSMO discovery conceptual model both in terms of components and execution semantics.

The component responsible for exposing Glue functionalities to other services is the Communication Manager. It offers plain Web Services for publishing semantic Web Service Descriptions (WSD), submitting the goal for searching the repository for WSDs previously published and, finally, getting the results of the matching. The components responsible for translating the SOAP messages managed by the Communication Manager to semantic description of Web Services and goals are named Constructor. The Goal Translator is the component responsible for using ggMediator to translate the user goals in goals more close to the provider perspective. Finally, the central component of Glue is the Proof Generator which is responsible for gathering all the necessary information from the other components, invoking the internal reasoner and providing the discovered results to the client.

The execution semantics of Glue implements a composite discovery procedure that

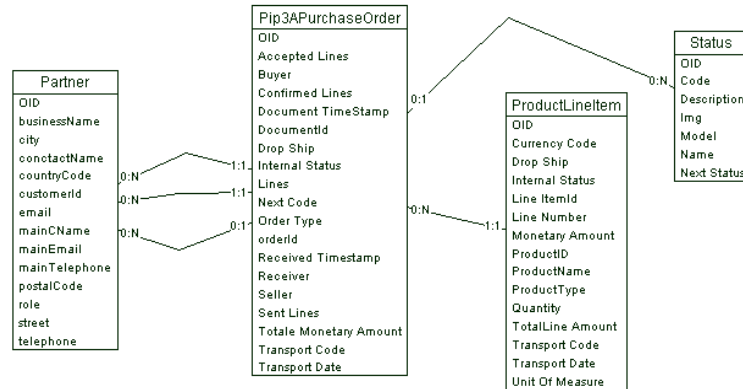
1. given an identifier of a class of goals and an XML message constructs a goal;
2. looks up the ggMediators that can be used to translate the instance of goal in another instance;
3. for each instance of goal looks up the wgMediators that has the class of the goal as target;
4. filter the Web Services limiting the scope to those that are sources of the identified wgMediators; and
5. evaluates the rules in the wgMediator returning only those Web Services that semantically match the goal.

Glue implementation uses internally F-logic and it is built around an open source F-logic inference engine called Flora-2 that runs over XSB, an open source implementation of tabled-prolog and deductive database system.

### 3 Initial Modeling

**Mediator scenario.** The modeling of the mediator started from the design of the data model. The RosettaNet message was analyzed and a corresponding WebML E-R diagram was obtained from it. We identified three main entities: the Pip3APurchaseOrder, the Partner and the ProductLineItem, as showed in Figure 2.

As showed by relationships in Figure, each Pip3APurchaseOrder instance has one of more ProductLineItem instances, one Partner representing the Buyer, one Partner representing the Seller and one Partner representing the Receiver. We created also a entity Status to track the status of each Pip3APurchaseOrder. We modeled only the essential data for the scenario.

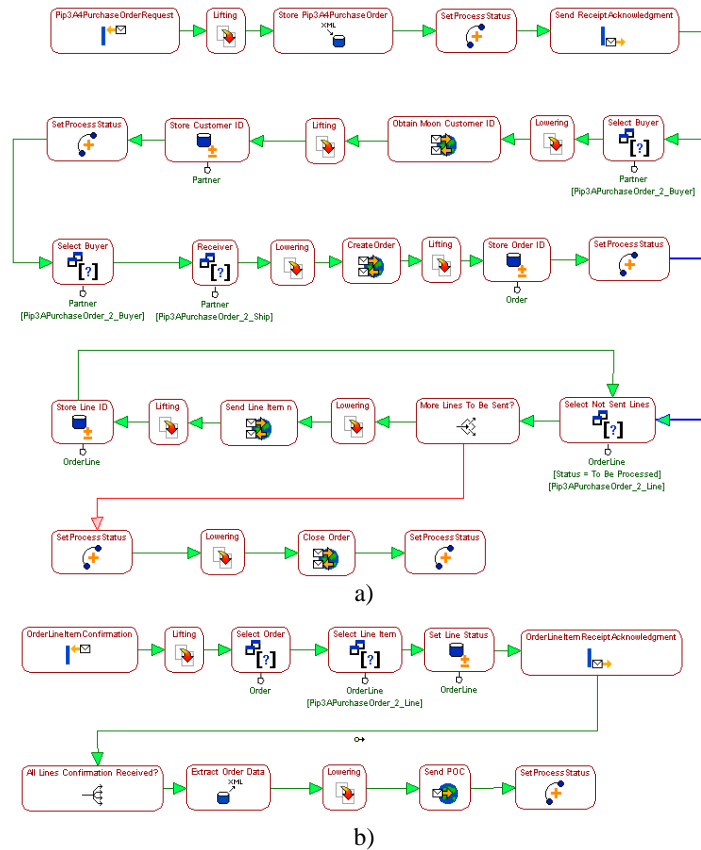


**Fig. 2.** The E-R diagram for the data model used of the Mediator.

Once the WebML data model was completed, we started modeling the WebService providing the mediation feature. The obtained model is showed in Figure 3a.

- First we modeled the operation receiving the RosettaNet message and forwarding the order to Moon. In the first line, as soon as the order is received (Solicit Unit), the Pip3APurchaseOrder is converted to the Canonic XML (Adapter Unit) and stored in the database (XML-In Unit), the status of the current Pip3APurchaseOrder is set to “To Be Processed” (Connect Unit) and the Acknowledged message is returned to the service invoker (Response Unit).
- Next, the Buyer Partner is selected (Selector Unit) and a message to query the CRM service is created (Adapter Unit) and sent to the Moon Legacy System (Request-Response Unit). Once a reply has been received, the CustomerId is extracted from the reply message (Adapter Unit) and stored in the data model (Modify Unit). The status of the order is set to “CustomerId received” (Connect Unit).
- Then Buyer Partner and the Receiver Partner are selected (Selector Units) and a message for the createNewOrder operation is created (Adapter Unit) and sent to the Moon Legacy System (Request-Response Unit). Once a reply has been received, the OrderId is extracted from the reply message (Adapter Unit) and stored in the data model (Modify Unit). The status of the order is set to “OrderId received” (Connect Unit).
- Next, the ProductLineItem instaces related to current Pip3APurchaseOrder are processed by a cycle: at every interaction a message for a single line is created and sent to the Moon Legacy System (Request-Response Unit), and the received LineId is stored (Modify Unit).
- Finally when all the lines have been processed the message for the closeOrder operation is created (Adapter Unit) and sent to the Moon Legacy System (Request-Response Unit) and the status of the order is set to “Order closed” (Connect Unit).

Then we modeled the operation to receive lines confirmation by the Moon Legacy



**Fig. 3.** The WebML model of the Mediator.

System (Figure 3b).

- For each line confirmation received (Solicit Unit), the status is extracted (Adapter Unit), the relative order and line stored in mediator database are selected (Selector Units), the status of the stored line is modified according to the received confirmation (Modify Unit) and the Acknowledge message is returned to the service invoker (Response Unit).
- Finally, if all the lines have been received (Switch Unit), the XML serialization of the data for the current Pip3APurchaseOrder is extracted (XML-Out Unit) and a RosettaNet Purchase Order Confirmation message is created (Adapter Unit) and sent to the RosettaNet client (Request-Response Unit) and the status of the order is set to “Rosetta PO Confirmation sent” (Connect Unit).

**Discovery scenario.** In modelling ontologies, goals, Web Services and Mediator for the first phase of the challenge, we followed the mediator centric methodology described in [2] and we used F-logic.

First of all, we modelled four ontologies including date-time, location, products and shipments. The development was kept to the minimum necessary for the two scenarios. In particular our date-time ontology is not expressive enough to model the generic notion of “business day”.

Secondly, we defined two classes of goals one for the shipment and one for purchasing. In both cases we modelled the capabilities limiting ourselves to post-condition. The following table shows the class of goal for the shipment and an instance of it.

|   |  |
|---|--|
| <pre>goalClass_Shipment::goalClass[   capability=&gt;capabilityGoal_Shipment::capabilityGoal[     postcondition=&gt;requestsShipmentService   ] ].  requestsShipmentService[   requestedPickupLocation=&gt;location,   requestedDeliveryLocation=&gt;location,   currentDateTime=&gt;dateTime,   requestedPickupDateTimeInterval=&gt;dateTimeInterval,   requestedDeliveryDateTime=&gt;dateTime,   requestedDeliveryModality=&gt;deliveryModality,   requestedGuarantee=&gt;guarantee,   goodWeight=&gt;float,   goodDimension=&gt;dimension,   requestedShipmentPriceInterval=&gt;priceInterval ].</pre> | <pre>goalInstance:goalClass_Shipment[   capability-&gt;_#:capabilityGoal_Shipment[     postcondition-&gt;_#:requestsShipmentService[       requestedPickupLocation-&gt;stanford,       requestedDeliveryLocation-&gt;stanford,       currentDateTime-&gt;_#:dateTime[         date-&gt;_#:date[dayOfMonth-&gt;28,monthOfYear-&gt;4,year-&gt;2006],         time-&gt;_#:time[hourOfDay-&gt;23,           minuteOfHour-&gt;0,secondOfMinute-&gt;0]       ],       requestedPickupDateTimeInterval-&gt;_#:dateTimeInterval[         start-&gt;_#:dateTime[...], end-&gt;_#:dateTime[...],         requestedDeliveryDateTime-&gt;_#:dateTime[...],         requestedDeliveryModality-&gt;letter,         requestedGuarantee-&gt;guaranteeYes,         goodWeight-&gt;10,         goodDimension-&gt;_#:dimension[l-&gt;100,w-&gt;100,h-&gt;100],         requestedShipmentPriceInterval-&gt;_#:priceInterval[start-&gt;0,end-&gt;1000]       ]     ]   ] ].</pre> |
|---|--|

Third, we model the classes of Web Services for shipment and purchasing. In both cases we model all the restrictions that must hold in order to invoke the service as assumptions and the result provided by the service as post conditions. In the following table we show the class of Shipment Web Service and an instance of it.

|  |   |
|--|---|
| <pre>wsdClass_Shipment::wsdClass[   capability=&gt;capabilityWSD_Shipment::capabilityWSD[     assumption=&gt;restrictionsOnShipmentService,     postcondition=&gt;providesShipmentService   ] ].  restrictionsOnShipmentService[   minNumOfHoursBetweenOrderAndPickup=&gt;integer,   maxNumOfDaysBetweenOrderAndPickup=&gt;integer,   maxNumOfDaysBetweenOrderAndPickup=&gt;integer,   maxNumOfDaysBetweenOrderAndDelivery=&gt;integer,   minPickupDTInterval=&gt;integer,   maxPickupDTInterval=&gt;integer,   maxGoodWeight=&gt;float,   weightToDimensionalWeightThreshold=&gt;float ].  providesShipmentService[   pickupLocations=&gt;&gt;location,   deliveryLocations=&gt;&gt;location,   pickupTimeInterval=&gt;timeInterval,   price=&gt;&gt;shipmentPricing ].</pre> | <pre>wsdInstance_Shipment13::wsdClass_Shipment[   nonFunctionalProperties-&gt;_#:dc_publisher-&gt;'Muller',   capability-&gt;_#:capabilityWSD_Shipment[     assumption-&gt;_#:restrictionsOnShipmentService[       minNumOfHoursBetweenOrderAndPickup=&gt;0,       maxNumOfDaysBetweenOrderAndPickup=&gt;2,       maxNumOfDaysBetweenOrderAndPickup=&gt;5,       minPickupDTInterval=&gt;7200,       maxPickupDTInterval=&gt;259200,       maxGoodWeight=&gt;50,     ],     postcondition-&gt;_#:providesShipmentService[       pickupLocations-&gt;&gt;{northAmerica,southAmerica,         africa,asia,europe},       deliveryLocations-&gt;&gt;{northAmerica,southAmerica,         africa,asia,europe},       pickupTimeInterval-&gt;_#:timeInterval[...],     ],     price-&gt;&gt;{ _#:shipmentPricing[       location-&gt;worldwide,       deliveryModality-&gt;deliveryModality,       guarantee=&gt;guaranteeNo,       basePrice-&gt;0,       pricePerWeight-&gt;0]     }   ] ].</pre> |
|--|---|

Glue approach was sufficient to model most of the details, but we were not able to model some complex features that require invocation of external Web Services to gather more information regarding a Web Service.

Last, but not least we model the matching rules within the wgMediators. The rules, written in F-logic, can be divided in three groups: those that calculate intermediate results (such as the price), those that evaluate the restrictions in the assumption part of the description and those that describe the transactions in the post-condition part of the description. The table below shows an example of each type of rule.

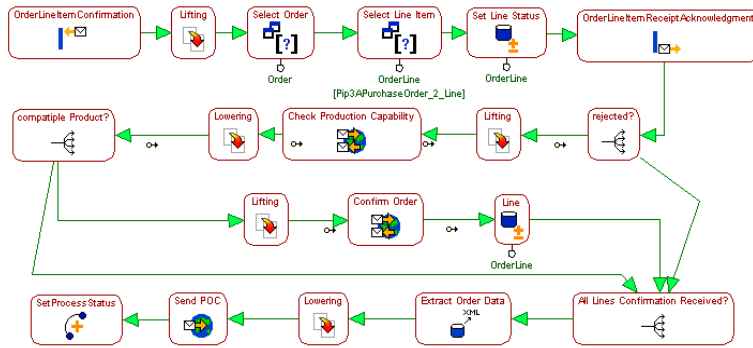
|  |
|--|
| <p><b>Rule that calculate intermediate results</b></p> <pre> calculateShipmentPrice(ShipmentPricing,Location,DeliveryModality,Guarantee,GoodWeight,PriceCalculated) :-   (Location::ShipmentPricing.location;Location=ShipmentPricing.location),   (DeliveryModality::ShipmentPricing.deliveryModality;DeliveryModality=ShipmentPricing.deliveryModality),   (Guarantee=ShipmentPricing.guarantee;Guarantee=guaranteeNo),   PriceCalculated is (ShipmentPricing.basePrice + (GoodWeight-1)*ShipmentPricing.pricePerWeight) . </pre>  |
| <p><b>Rule that evaluate assumptions</b></p> <pre> checkRestrictionOnMaxNumOfDaysBetweenOrderAndPickupInterval(RequestsShipmentService,RestrictionsOnShipmentService) :-   RequestsShipmentService[     currentDateTime-&gt;OrderDateTime.requestedPickupDateInterval-&gt;_[start-&gt;PickupDateTimeStart, end-&gt;PickupDateTimeEnd]   ],   RestrictionsOnShipmentService[     maxNumOfDaysBetweenOrderAndPickupStart-&gt;MaxDaysForStart,          maxNumOfDaysBetweenOrderAndPickupEnd-&gt;     &gt;MaxDaysForEnd],     daysBetween(PickupDateTimeStart,OrderDateTime,X),(X&lt;MaxDaysForStart;X=MaxDaysForStart),     daysBetween(PickupDateTimeEnd,OrderDateTime,Y),(Y&lt;MaxDaysForEnd;Y=MaxDaysForEnd) . </pre> |
| <p><b>Rule that encode a necessary condition</b></p> <pre> checkContainmentOfPickupAndDeliveryLocation(RequestsShipmentService,ProvidesShipmentService) :-   RequestsShipmentService[requestedPickupLocation-&gt;X],ProvidesShipmentService[pickupLocations-&gt;&gt;Y],   (X=Y;X::Y),   RequestsShipmentService[requestedDeliveryLocation-&gt;H],ProvidesShipmentService[deliveryLocations-&gt;&gt;K],   (H=K;H::K). </pre>  |

#### 4 Addressing changes request

**Mediator scenario.** In the second phase of the challenge, first the RosettaNet message was changed introducing also a `core:shipTo` element for each `ProductLineItem`. This was achieved by simply updating our data model by introducing a new relationship between the entity `ProductLineItem` and the entity `Partner`.

Then, a change in the mediation process was introduced: when the Stock Management system is incapable to fulfil request from the customer and it replies that the particular line item cannot be accepted, the Mediator must communicate with the legacy Production Management system to obtain relevant information on date and price to manufacture a new product. If this information meets initial expectations of the customer as specified in the RosettaNet message, the product should be ordered. To fulfill new requirements, we changed the mediator by introducing the set of operation required to query the new the Production Management Web Service (Figure 4).

**Discovery scenario.** Most of the proposed changes indeed were made to the WSDL; this could have been difficult to handle if we had proceed to model choreography for invoking the Web Services after the discovery. Concerning the changes to capabilities, most of them require minor modifications such as: adding some instances to the ontologies (e.g. Oceania as possible location), changing some instances of Web Services (e.g. updating latest pick up time, removing the distinction between letter and word-



**Fig. 4.** The WebML model of the modified portion of the Mediator (cfr Fig. 3b).

widePriorityExpress). Fewer of the changes also require re-modelling the classes of Web Services (e.g., by changing the accept multiplicity for a given property or adding the information about the additional price per collection) and, consequently, of the corresponding wgMediator. One proposed change that heavily impacted on the wgMediator was the new rule for computing the price. Such rule, shown in the following table, discover the correct Shipment Pricing in function of the location and the guarantee and then calculates the price by taking care of the Dimensional Weight (calculated by the *calculateDimensionalWeight* rule) and the additional price per collection, if set. The calculated price is used by other rules to filter the results according to the maximal price expressed in the goal.

|   |
|---|
| <pre> calculateShipment- Price(ShipmentPricing.Location.DeliveryModality.Guarantee.GoodDimensionalWeight,NumberOfPackages,PriceCalculated) :- (Location::ShipmentPricing.location;Location=ShipmentPricing.location), (DeliveryModality::ShipmentPricing.deliveryModality;DeliveryModality=ShipmentPricing.deliveryModality), (Guarantee=ShipmentPricing.guarantee;Guarantee=guaranteeNo), PriceCalculatedForOnePackage is (ShipmentPricing.basePrice + (GoodWeight)*ShipmentPricing.pricePerWeight), ( (ShipmentPricing.additionalPricePerCollection=(-1),PriceCalculated is (PriceCalculatedForOnePackage*NumberOfPackages)); (ShipmentPricing.additionalPricePerCollection&gt;(-1),PriceCalculated is (PriceCalculatedForOnePack- age+ShipmentPricing.additionalPricePerCollection)) ). </pre> |
|---|

Two are the main lessons learned from this work. On the one hand, in order to satisfy the requirements of the Discovery scenario, an intermediate invocation of external Web Services is required; Glue does not support such functionalities and we agree that Glue should be extended in this direction by specifying both the reference to the external service and its choreography. On the other hand, we recognize a limit of Glue methodology, which is well suited for scenarios in which a large number of service providers agree in describing their services according to a shared pattern. In this case, classes can be seen as semantically enhanced UDDI tModels. However, when few services provide such a different set of capabilities, changes that involve only one service (in case they involve re-modelling of the class) either induce changes other each Web Service description, or requires the introduction of an intermediate derived class and of the corresponding derived wgMediator.

## 5 Conclusions

This paper briefly summarized our solution to the SWS Challenge 2006 phase II. The challenge has proved a valid test bench for validating our approach, consisting in leveraging software engineering tools and methods for the design and development of Semantic Web Services and Applications.

While addressing the Challenge problems, we have in parallel developed a method for the semi-automatic extraction of the components of the WSMO architecture by using existing software engineering abstractions, addressing the extraction of the semantic description of services and design of mediators, goals, choreography, and orchestration, as well as specifying in details most of the transformations that are needed. We have also produced the definition of new hypertext primitives for importing ontologies, services and mediators, as well as for querying ontological information.

In the future, we plan to consolidate our approach and to develop new software tools centered on WSMO components; we expect to use classical data, process, and Web abstractions whenever these are naturally applicable to describe the behavior of WSMO components, and otherwise to use new WSMO-specific abstractions.

## References

- 1 Brambilla, M., Ceri, S., Comai, S., Fraternali, P., Manolescu, I., Specification and design of workflow-driven hypertexts, *Journal of Web Engineering*, 1(2) April, 2003.
- 2 Brambilla, M., Ceri, S., Fraternali, P., Acerbis, R., Bongio, A.: Model-driven Design of Service-enabled Web Applications, *SIGMOD 2005, Industrial Track*.
- 3 Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: *Designing Data-Intensive Web Applications*, Morgan-Kaufmann, December 2002.
- 4 Manolescu, I., Brambilla, M., Ceri, S., Comai, S., Fraternali, P.: Model-Driven Design and Deployment of Service-Enabled Web Applications, *TOIT*, Volume 5, number 3 (August 2005).
- 5 Emanuele Della Valle and Dario Cerizza. The mediators centric approach to automatic webservice discovery of Glue. In Martin Hepp, Axel Polleres, Frank van Harmelen, and Michael R. Genesereth, editors, *MEDIATE2005*, volume 168 of *CEURWorkshop Proceedings*, pages 35–50. CEUR-WS.org, 2005.
- 6 WebML.org.: <http://www.webml.org>, 2006.
- 7 Webratio: <http://www.webratio.com>, 2006.