Flexible Specification of Semantic Services using Web Engineering Methods and Tools

Marco Brambilla[1], Stefano Ceri[1], Irene Celino[2], Dario Cerizza[2], Emanuele Della Valle[2], Federico M. Facca[1], Christina Tziviskou[1]

[1] Dipartimento di Elettronica e Informazione, Politecnico, 20133 Milano, Italy
{mbrambil, ceri, facca, tzivisko}@elet.polimi.it
[2] CEFRIEL, 20133 Milano, Italy
{celino, cerizza, dellavalle}@cefriel.it

Abstract. Today's IT architectures are largely considered the biggest roadblocks that most companies face when making strategic business moves. The good news is that the Software Engineering community and the Semantic Web community are bridging their efforts to foster a new generation of design and development tools. In this paper we describe a top-down approach to the design and development of applications employing Semantic Web Services, that combines semantic methods and tools (i.e., ontology engineering, semantic service annotation and discovery) with Software Engineering ones (i.e., formal business process development, component-based software design techniques, and computer-aided software design). The described approach was the most complete among those that took part in the Semantic Web Service Challenge 2006.

1 Introduction

The vision of a *flexible IT* architecture, able to accommodate at no cost the changing business requirements, has been fascinating almost two generations of CIOs. Since 1970, the next generation of middlewares and application frameworks has been promising scalability, reuse, distribution, etc. However, the situation is rapidly changing; software development practitioners discovered the need of modeling their application and the respective data, and therefore knowledge technologies slightly got embraced in techniques and methodologies of Software Engineering. In the meantime, knowledge technologies got momentum from the Semantic Web, and a variety of semantic-empowered tools were developed, whose ultimate goal is the management of advanced applications in real environments. In this paper we report one of these cases in which two groups, one from the Web Engineering community and one from the Semantic Web community, got engaged together in the Semantic Web Service challenge 2006 [1] and ended up in proposing the most complete solution [2].

The Semantic Web Service challenge aims at employing semantics-based technologies on a set of problems. Participants are not measured on computational speed but rather on the way they operationally explore the semantic technology space. The challenge presents two scenarios: in the *Mediation Scenario*, a legacy system is made compliant with a RosettaNet purchase order [3] (a complex XML specification as well as a simple protocol for exchanging messages about its processing); in the *Dis-*

covery Scenario, the order is fulfilled by automatically discovering a new supplier together with a shipper. The challenge requires participant to present their solutions, to freeze them and to cope with a set of previously unknown changes to proof the flexibility of the initial solutions. The evaluation of the solutions is based on how much participants have to alter their systems to respond to the released changes.

The problem is clearly neither a pure Semantic Web one, nor a pure Software Engineering one. Semantics is clearly needed to address in a flexible way the Discovery scenario, but Software Engineering tools and methods are the right ones to address in a flexible way the Mediation scenario.

For this reason (see Section 2) we adopt an original mix of Semantic Web and Software Engineering techniques: WSMO [4] as Semantic Web Service approach, Glue [5, 6] as Semantic Web Service Discovery engine, WebML [7, 8] as Web engineering model for designing and developing semantically rich Web applications implementing Service Oriented Architecture, and WebRatio [9] as WebML CASE tool.

The combined approach introduces Semantic Web technologies in the development process of Web application (see Section 3) and employs tools (i.e., WebRatio) developed for Web engineering using Semantic Web languages and tools (see Section 4). As a result it offers an application that combining Web Engineering techniques and Semantic Web tools and languages (see Section 5). Our experience is just one of the many that are occurring in these days but we believe it is worth to report, since we introduce a significant contribution in the application of Software Engineering techniques to Semantic web application design. In Section 6 we sum up our experience and provide some forecasts for model-driven Semantic Web Service application design and development.

2 Technologies employed

In the following we provide a background on the languages and the tools we adopted.

2.1 WSMO, WSML and WSMX

The WSMO initiative [4] aims at providing a comprehensive framework for handling Semantic Web Services which includes the WSMO conceptual model, the WSML language [10] and the WSMX execution environment [11].

The Web Service Modeling Ontology (WSMO) is an ontology for describing various aspects related to Semantic Web services. WSMO defines four modeling elements (ontologies, goals, Web services and mediators) to describe several aspects of Semantic Web services, based on the conceptual grounding of the Web Service Modeling Framework (WSMF) [12]. *Ontologies* provide the formal semantics to the information used by all other components. They serve in defining the formal semantics of the information, and in linking machine and human terminologies. *Web services* represent the functional and behavioral aspects, which must be semantically described in order to allow semi-automated use. Each Web service represents an atomic piece of functionality that can be reused to build more complex ones. Web services are described in WSMO from three different points of view: besides non-functional proper-

ties, the core description includes functionality and behavior. The functionalities are described as *capabilities*, while the behavior is described in Web service *interface*. A Web service can be described by multiple interfaces, but has one and only one capability. *Goals* specify objectives that a client might have when consulting a Web service. In WSMO, a goal is characterized in a dual way with respect to Web services: goal's descriptions include the *requested capability* and the *requested interface*. Finally, *mediators* provide interoperability facilities among the other elements. They describe elements that aim at overcoming structural, semantic or conceptual mismatches that appear between the different components that build up a WSMO description. Web Service Modeling Language (WSML) offers a set of language variants for describing WSMO elements that enable modellers to balance between expressivity and tractability according to different knowledge representation paradigms. The most basic, and least expressive, variant is WSML-Core. WSML Core is separately extended in the directions of these two paradigms by the variants WSML-DL and WSML-Flight, respectively. WSML-Flight is based on a logic programming variant of F-Logic [13]. Web service Execution Environment (WSMX) is a framework for the automation of discovery, selection, mediation, and invocation of Semantic Web services. WSMX is based on WSMO and, at the same time, it is a reference implementation of it.

2.2 Glue WSMO discovery Engine

Glue [5, 6] 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 making wgMediators the conceptual element responsible for evaluating the matching; and
- by redefining the discovery mechanism as a composite procedure where the discovery of the appropriate mediators and the discovery of the appropriate services are 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. In particular the execution semantics of Glue implements a composite discovery procedure that

1. given a class of goals and an XML message containing the necessary information, constructs an instance goal;
2. looks up the wgMediators that has the class of the goal as target;
3. filters the Web Services limiting the scope to those that are sources of the identified wgMediators; and
4. evaluates the rules in the wgMediator returning only those Web Services that semantically match the goal.

Glue implementation uses internally F-logic (semantically close to WSML Flight) 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.

2.3 WebML/WebRatio WS edition

The WebML language [7, 8] 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 (Figure 1). 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 through *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. sending an e-mail). In [14] the language has been extended with operations supporting process specifications.

To describe Web services interactions, WebML includes some Web service units [15, 16]. 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 ser-

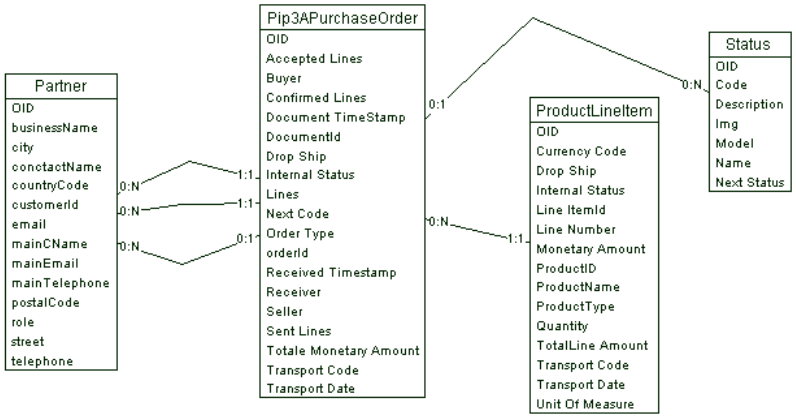


Figure 1. The E-R diagram for the data model used of the Mediator.

vice (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. Another operation typically involved in Web Service interactions is the *Adapter Unit*, which is able to apply any kind of XSLT transformation to a XML document. This unit is often used in conjunction with the *XML-In unit* or the *XML-Out unit*: the first is used to import canonic XML data (formatted according a particular XSD) into the database, the latter to extract database instances and convert them to the a canonic XML format. Usage examples of these units can be found in Figure 4.
The WebML language is *extensible*, allowing for the definition of customized operations and units. It has been implemented in the CASE tool WebRatio [9], 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.

3 Coping with Semantics in the development process

The phases of the development process of a semantic Web application are shown in Figure 2. In line with the classic Boehm's Spiral model and with modern methods for Web and software engineering, the development phases must be applied in an iterative and incremental manner, in which the various tasks are repeated and refined until results meet the business requirements.

- *Requirements specification* is the activity in which the application analyst collects and formalizes the essential information about the application domain and expected functions.
- *Process design* focuses on the high-level schematization of the (possibly distributed) processes underlying the application. Process design and distribution influence the next steps of the development, which should take into account process requirements.
- *Data design* is the phase in which the data expert organizes the main information objects identified during requirements specification into a comprehensive and coherent domain model, that may comprise the importing of existing ontologies and the definition of new ones.

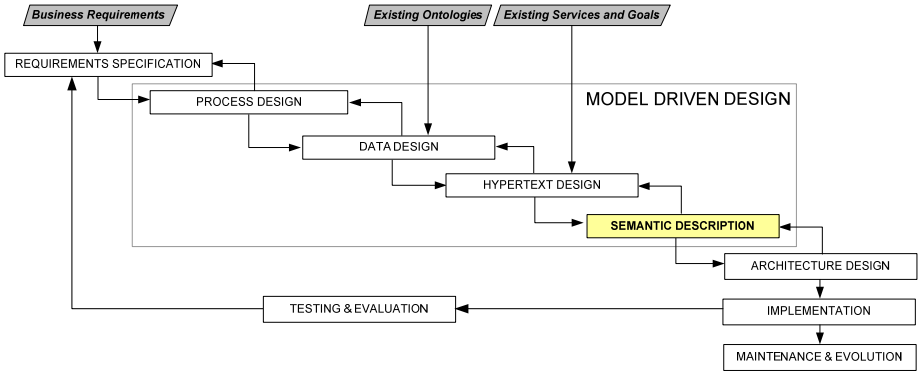


Figure 2. Phases in the development process of data- and process-intensive Web applications.

- *Hypertext design* is the activity that transforms the functional requirements identified during requirements specification into one or more Web services and Web site views embodying the needed retrieval and manipulation methods. Hypertext design may comprise importing or referencing existing services and goals. It exploits high level models, which let the architect specify how content elements are published within pages, how services provide information to requestors, and how hypertext elements are connected by links to form a navigable structure.

This paper is focused on *Semantic description* of the application, a new design phase which is required to provide WSMO compatibility; it consists in a set of tasks, partially automated, that aim at providing a set of semantic specifications of the application to be implemented. The other phases of Figure 2 are outside the scope of this paper. A comprehensive description of them can be found in [7].

4 Using Semantic Web tools together with Web engineering ones

Figure 3 summarizes the extraction of semantic descriptions of services from the application design. The design flow represents the main steps of the development process presented in Section 3. The various steps produce some intermediate artifacts (BPMN models, WebML skeletons, data models, hypertext models), possibly enriched by imported ontological descriptions (on top of Figure 3) and are exploited for deriving the set of WSMO specifications (at the bottom of Figure 3): the description of the mediator can be extracted from the hypertext describing the mediator (see Section 5.1); the Web services capability description is derived from hypertext model; the choreography information is derived from BP model; and the user goals (Section 5.1) are derived from the BP model and the hypertext model. This approach seamlessly fits into traditional Software Engineering methods and techniques based on system modeling (i.e. Model Driven Design and Model Driven Architectures); therefore, existing CASE tool for MDD can be easily extended for supporting the process.

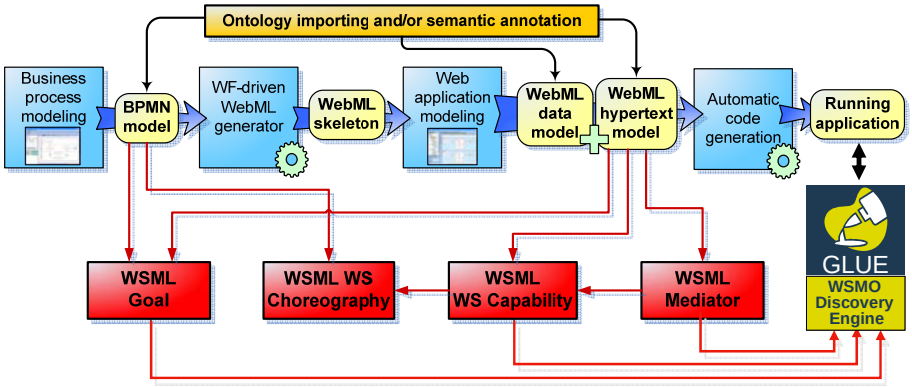


Figure 3. Overall picture of the extraction of semantic description of Web services.

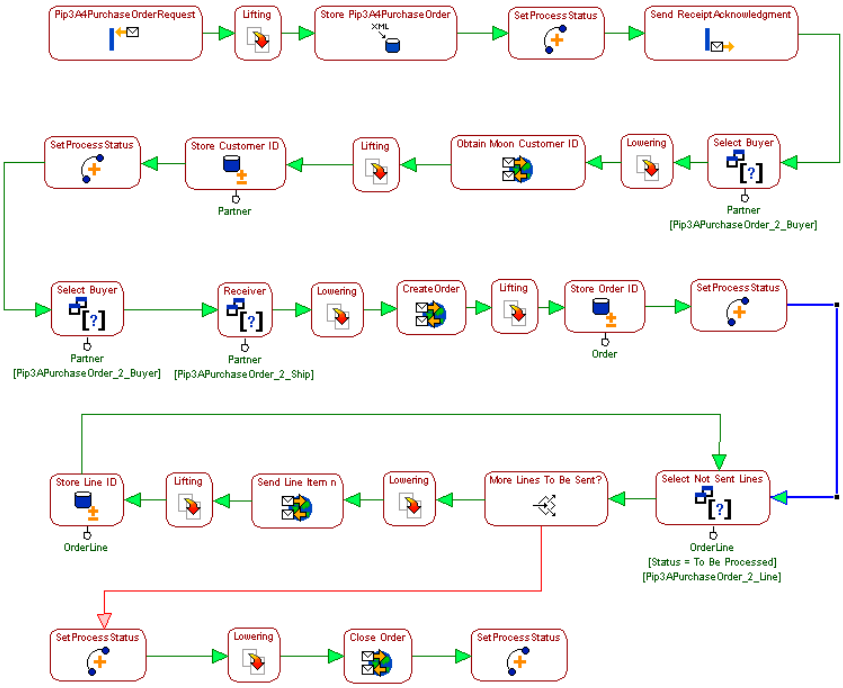
Our implementation experience proved this to be valid. Indeed, we extended the web-specific CASE tool WebRatio for supporting the approach.
Appropriate units have been devised for modeling operations on ontologies. We introduce a first set of units representing the basic ontological queries and inferences for Classes, Instances, and Properties. The units are aggregate primitives that execute properly depending on the type of their parameters. Other units are available for *Set Composition*, to perform classic set operations over 2 input sets of *uri*s (i.e., union, intersection, difference); *Importing Ontological Sources*, i.e., adding a remote or local data source that must be consistent with ontological model of the Web application; *Describing* a piece of knowledge, i.e., returning the RDF description of an *uri*, thus enabling automatic Web page annotation and data exporting.

5 Our solution to the Semantic Web Service challenge

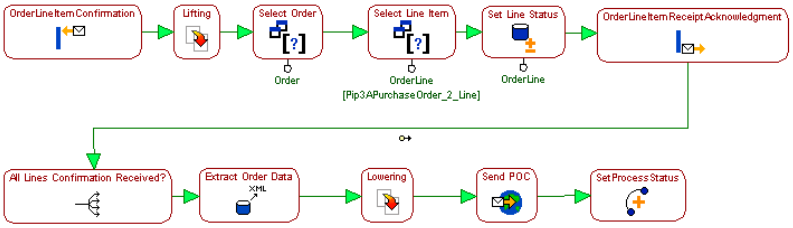
In this section we describe how we faced the two phases of the Challenge. In the first place we provide an overview of the initial solution. Then we give some insight about how we coped with the changing requirements and about the effort required to adapt our solution.

5.1 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 1.
As showed by relationships in Figure 1, 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 an entity Status to track the status of each Pip3APurchaseOrder. We modeled only the essential data for the scenario.



a)



b)

Figure 4. The WebML model of the Mediator.

Once the WebML data model was completed, we started modeling the WebService providing the mediation feature. The obtained model is shown in Figure 4a. Each line of the model depicted in the Figure corresponds to a specific step that the mediator must perform. Each of these steps comprises a set of specific operations on the received messages or on the local data. We exemplify in details the first two steps of the mediator, namely *(i)* the reception of the RosettaNet message and forwarding to the legacy system; (ii) the selection of the Buyer Partner:

- First, we modeled the operation receiving the RosettaNet message and forwarding the order to the legacy system. In the first line, as soon as the order is received (Solicit Unit), the Pip3APurchaseOrder is converted (Adapter Unit) and stored in the database (XML-In Unit), the status of the current

Pip3APurchaseOrder is set to “To Be Processed” (Connect Unit, a unit that creates new relationship instances between objects) and the Acknowledge message is returned to the service invoker (Response Unit).

- Next, the Buyer Partner is selected (Selector Unit, a unit that retrieves data instances according to a specified selector condition) and a message to query the CRM service is created (Adapter Unit) and sent to the 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).

Analogous operations are performed for the following steps.

We modeled separately the operation to receive lines confirmation by the Legacy System (Figure 4b).

- 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-conditions. The following table shows the class of goals for the shipment and an instance of it.

```
goalClass_Shipment::goalClass[
   capability=>capabilityGoal_Shipment::capabilityGoal[
        postcondition=>requestsShipmentService
   ]
].

requestsShipmentService[
   requestedPickupLocation=>location,
   requestedDeliveryLocation=>location,
   currentDateTime=>dateTime,
   requestedPickupDateTimeInterval=>dateTimeInterval,
   requestedDeliveryDateTime=>dateTime,
   requestedDeliveryModality=>deliveryModality,
   requestedGuarantee=>guarantee,
   goodWeight=>float,
   goodDimension=>dimension,
   requestedShipmentPriceInterval=>priceInterval
].
```

```
goalInstance:goalClass_Shipment[
      capability->_#:capabilityGoal_Shipment[
          postcondition->_#:requestsShipmentService[
              requestedPickupLocation->stanford,
              requestedDeliveryLocation->stanford,
              currentDateTime->_#:dateTime[
                      date->_#:date[dayOfMonth->28,monthOfYear->4,year->2006],
                      time->_#:time[hourOfDay->23,
                                    minuteOfHour->0,secondOfMinute->0]
              ],
              requestedPickupDateTimeInterval->_#:dateTimeInterval[
                  start->_#:dateTime[…], end->_#:dateTime[…] ],
              requestedDeliveryDateTime->_#:dateTime[…],
              requestedDeliveryModality->letter,
              requestedGuarantee->guaranteeYes,
              goodWeight->10,
              goodDimension->_#:dimension[l->100,w->100,h->100],
              requestedShipmentPriceInterval->_#:priceInterval[start->0,end->1000]
]]].
```

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

```
wsdClass_Shipment::wsdClass[
    capability=>capabilityWSD_Shipment::capabilityWSD[
        assumption=>restrictionsOnShipmentService,
        postcondition=>providesShipmentService
    ]
].

restrictionsOnShipmentService[
    minNumOfHoursBetweenOrderAndPickup=>integer,
    maxNumOfDaysBetweenOrderAndPickup=>integer,
    maxNumOfDaysBetweenOrderAndPickup=>integer,
    maxNumOfDaysBetweenOrderAndDelivery=>integer,
    minPickupDTInterval=>integer,
    maxPickupDTInterval=>integer,
    maxGoodWeight=>float,
    weightToDimensionalWeightThreshold=>float
].

providesShipmentService[
    pickupLocations=>>location,
    deliveryLocations=>>location,
    pickupTimeInterval=>timeInterval,
    price=>>shipmentPricing
    ].
```

```
wsdInstance_Shipment13:wsdClass_Shipment[
    nonFunctionalProperties->_#[ dc_publisher->'Muller'],
    capability->_#:capabilityWSD_Shipment[
        assumption->_#:restrictionsOnShipmentService[
            minNumOfHoursBetweenOrderAndPickup=>0,
            maxNumOfDaysBetweenOrderAndPickup=>2,
            maxNumOfDaysBetweenOrderAndPickup=>5,
            minPickupDTInterval=>7200,
            maxPickupDTInterval=>259200,
        maxGoodWeight=>50,
        ],
        postcondition->_#:providesShipmentService[
            pickupLocations->>{northAmerica,southAmerica,
                               africa,asia,europe},
            deliveryLocations->>{northAmerica,southAmerica,
                                 africa,asia,europe},
            pickupTimeInterval->_#:timeInterval[...],
            ],
            price->>{ _#:shipmentPricing[
                location->worldwide,
                deliveryModality->deliveryModality,
                guarantee->guaranteeNo,
                basePrice->0,
                pricePerWeight->0]}
]]].
```

Finally we modelled 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.

Rule that calculates intermediate results

```
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) .
```

Rule that evaluates assumptions

```
checkRestrinctionOnMaxNumOfDaysBetweenOrderAndPickupInterval(RequestsShipmentService,RestrictionsOnShipmentService) :-
    RequestsShipmentService[
        currentDateTime->OrderDateTime,requestedPickupDateTimeInterval->_[start->PickupDateTimeStart, end->PickupDateTimeEnd]
    ],
    RestrictionsOnShipmentService[
        maxNumOfDaysBetweenOrderAndPickupStart->MaxDaysForStart,                maxNumOfDaysBetweenOrderAndPickupEnd-
>MaxDaysForEnd],
    daysBetween(PickupDateTimeStart,OrderDateTime,X),(X<MaxDaysForStart;X=MaxDaysForStart),
    daysBetween(PickupDateTimeEnd,OrderDateTime,Y),(Y<MaxDaysForEnd;Y=MaxDaysForEnd) .
```

Rule that encodes a necessary condition

```
checkContainmentOfPickupAndDeliveryLocation(RequestsShipmentService,ProvidesShipmentService) :-
    RequestsShipmentService[requestedPickupLocation->X],ProvidesShipmentService[pickupLocations->>Y],
    (X=Y;X::Y),
    RequestsShipmentService[requestedDeliveryLocation->H],ProvidesShipmentService[deliveryLocations->>K],
    (H=K;H::K).
```

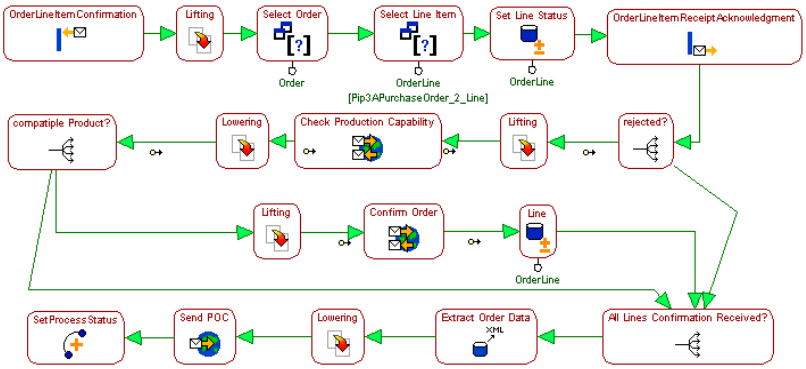


Figure 5. The WebML model of the modified portion of the Mediator (cfr Fig. 4b).

The WebML model, not shown here due to space limitations, shares the goal model with the discovery engine and allows the user to interact with it by providing a browsable front-end for defining and refining the requests.

5.2 Addressing changes request

The organizers of the Semantic Web Service Challenge, after the freeze of the solutions to the initial set of problems, introduced some changes in the two scenarios to test the flexibility of the solutions proposed by participants to the challenge. The separated approaches we initially used to solve the two scenarios permitted us to address the new changes in a easy way. This also proved that the initial choice of adopting an approach mainly based on the Software Engineering methods and techniques for the mediation scenario and an approach mainly based on the Sematic Web technologies for the discovery scenario was a good and flexible enough choice.

Mediator scenario. In the second phase of the challenge, the new requirements imposed a change in the format of the exchanged messages and a change in the mediation process. The change in the format comprised the request of a new `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, the change in the mediation process consisted in the following requirement: when the Stock Management system is incapable to fulfill a 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 this new requirement, we changed the mediator by introducing a set of operations needed to query the new the Production Management Web Service (Figure 5).

Discovery scenario. Most of the changes required in the discovery scenario were addressed through minor modifications such as: adding some instances to the ontologies (e.g. Oceania as possible location), and changing some instances of Web Services (e.g. updating latest pick up time, removing the distinction between letter and wordwidePriorityExpress). Fewer of the changes also required to re-model the classes of Web Services (e.g., by changing the accepted multiplicity for a given property or adding the information about the additional price per collection) and, consequently, the corresponding wgMediator. One change in the requirements that heavily impacted on the wgMediator was the new rule for computing the price. Such rule, shown in the following table, discovers 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.

```
calculateShipment-
Price(ShipmentPricing,Location,DeliveryModality,Guarantee,GoodDimensionalWeight,NumberOfPackages,PriceCalculated) :-      (Loca-
tion::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>(-1),PriceCalculated is (PriceCalculatedForOnePack-
age+ShipmentPricing.additionalPricePerCollection))
    ).
```

Generally speaking, thanks to the application of the software engineering methods and development processes, applying changes to the application revealed relatively inexpensive, especially if compared with pure Semantic Web-based approaches. Indeed, most of the changes could be applied at a high level of abstraction, to the design specification of the application, instead of burdening with manual low level coding of the modified requirements.

6 Discussion

Our approach was evaluated the most complete (see http://sws-challenge.org/wiki/index.php/Workshop_Budva) among those presented in the Phase II of the SWS Challenge 2006. We accomplished the coverage of all the requirement of the challenge by teaming up approaches best suited for each part of the challenge:

- we addressed the "*mediation scenario*" of the challenge (see section 3) with the WebML design and implementation of the wwMediator and the usage of the CASE tool WebRatio; and
- we addressed the "*discovery scenario*" by means of Glue WSMO discovery engine that was integrated in the deployed Semantic Web Service application.

The major merits of our solution were twofold: the solid rooting in the tradition of software engineering and the use of sophisticated, up-to-date Web technology. The

first factor convinced us to give the maximum attention to "the method", consisting of gathering requirements, producing formal specifications, and then adapting them to the new context of the Semantic Web. The second factor was provided by our use of WebRatio, the tool which was used in the challenge, which automatically generates software integrating JSP scripts with embedded data management commands and with Web service calls. Augmenting them so as to generate annotations and WSMO-compliant components has been rather easy, once the semantics of these components have been well understood in terms of classical ER and BP models. Finally, the integration of WebRatio with Glue, the WSMO discovery engine, shows the advantages coming from the joint application of Software Engineering techniques and Semantic Web tools. Even if our solution was the most complete, we did not achieve a complete coverage of the set of problems proposed by the challenge. The not complete coverage in case of the Mediation scenario was due to a misunderstanding of the initial requirements. As regards to the Discovery scenario, to fully cover the requirements we need to improve the Glue Discovery engine and its integration with WebML. We plan to provide an improved solution, solving most of the issues left open, for the Third Phase of the Challenge, that will be held in conjunction with the ISWC conference.

Besides the ones discussed at the Challenge, very few proposals exist that formalize the development cycle and the high level models of Semantic Web applications and services. [17] presents an engineered approach to extraction of semantic annotations from XML schemas and documents to be published in dynamic Web applications. Our research effort is more similar to the recent efforts of the Object Management Group (http://www.omg.org). The OMG proposed the Ontology Definition Metamodel (ODM) [19] to define a suitable language for modeling Semantic Web ontology languages and hence Semantic Web applications in the context of the Model Driven Architecture (http://www.omg.org/mda). In 2006, Acuna and Marcos presented a first systematic approach to the design of Semantic Web applications, based on the MDA/MDD methodologies [18], introducing MIDAS-S, an extension of the existing MIDAS framework for designing Web applications. These proposals are can be regarded as first contributions to the field, but they still do not provide a clear roadmap to the design of Semantic Web applications.

We believe that development methods and tools for the Semantic Web should not be different from the classic paradigms which are now dominating software design and deployment: habits of the software developer community should not change. Therefore, Semantic Web developers should adapt classic (UML-based) methods and tools hosted by classic tool frameworks (such as Eclipse). We fully agree with Charles Petrie's words: "*If semantic technology has a future — and I'm sure that it does — it's in software engineering*". Our participation to the challenge is a first attempt in this direction, where two groups of Web engineering experts and Semantic Web experts joined their culture, habits, and tool experience. The ability of our solution to adapt to changes is mostly the merit of our use of enhanced software engineering methods and platforms.

References

1 Semantic Web Service Challenge: http://www.sws-challenge.org .
2 Semantic Web Service Challenge Evaluation: http://sws-challenge.org/wiki/index.php/Workshop_Budva
3 RosettaNet Purchace Order (PIP 3A4): http://www.rosettanet.org/PIP3A4
4 Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web Service Modeling Ontology. Applied Ontology, 1(1): 77–106, 2005.
5 Della Valle, E., Cerizza, D.: Cocoon glue: a prototype of WSMO discovery engine for the healthcare field. In Proceedings of 2nd WSMO Implementation Workshop WIW'2005, 2005.
6 Della Valle, E., Cerizza, D.: The mediators centric approach to automatic web service discovery of Glue. In Hepp, M., Polleres, A., van Harmelen, F., Genesereth, M.R., editors, In Proceedings of the First International Workshop on Mediation in Semantic Web Services (MEDIATE 2005), Amsterdam, The Netherlands, December 12, 2005, CEURWorkshop Proceedings, 8, 35–50.
7 Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan-Kaufmann, December 2002
8 WebML.org:. http://www.webml.org, 2006.
9 Webratio: http://www.webratio.com, 2006.
10 de Bruijn, J., Lausen, H., Krummenacher, R., Polleres, A., Predoiu, L., Kifer, M., Fensel, D.: The Web Service Modeling Language WSML, March 2005. Available at: http://www.wsmo.org/TR/d16/d16.1/v0.2/
11 Haller, A. , Cimpian, E., Mocan, A., Oren, E., Bussler, C.: WSMX - A Semantic Service-Oriented Architecture. In Proceedings of the 2005 IEEE International Conference on Web Services (ICWS 2005), 11-15 July 2005, Orlando, FL, USA, 321–328, 2005.
12 Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF. In Electronic Commerce Research and Applications, 1(2), 113–137, 2002.
13 Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. In Journal of ACM, 42, 741–843, 1995.
14 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.
15 Brambilla, M., Ceri, S., Fraternali, P., Acerbis, R., Bongio, A.: Model-driven Design of Service-enabled Web Applications, SIGMOD 2005, Industrial Track.
16 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.
17 Reif, G., Gall, H., Jazayeri, M.: WEESA: Web engineering for semantic Web applications, 14th International World Wide Web Conference (WWW'05), Chiba, Japan, May 2005, pp. 722-729.
18 Acuña, C. J., Marcos, E.: Modeling semantic web services: a case study. Proceedings of the 6th international conference on Web engineering (ICWE '06), Palo Alto (USA), ACM Press, July 2006, pp. 32-39.
19 OMG: Ontology Definition Metamodel (ODM). http://www.omg.org/cgi-bin/doc?ad/06-05-01.pdf.