Model-Driven Design and Development of Semantic Web Service Applications

Marco Brambilla¹, Irene Celino², Stefano Ceri¹, Dario Cerizza³, Emanuele Della Valle², Federico Michele Facca¹

¹ Politecnico di Milano, Dipartimento di Elettronica e Informazione, 20133 Milano, Italy
{Marco.Brambilla, Stefano.Ceri, Federico.Facca}@polimi.it
² CEFRIEL, 20133 Milano, Italy
{celino,cerizza,dellavalle}@cefriel.it

Abstract. This paper proposes 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 uses software engineering methods and tools, such as: formalized development processes, component-based software design techniques, and computer-aided software design. This approach leads from an abstract view of the business needs to application concrete implementation, by means of powerful code generators. Our method is based on existing models for the specification of business processes (such as BPMN) combined with Web engineering models for designing and developing semantically rich Web applications implementing Service Oriented Architectures (such as WebML) with the support of Semantic Web Service tools (such as WSMX and WSMT). The proposed framework increases the efficiency of the whole design process, yielding to the construction of semantic Web service applications spanning over several enterprises. In our paper, we show that business process and Web engineering models have sufficient expressive power to support the semi-automatic extraction of semantic descriptions (i.e., WSMO ontologies, goals, Web services, and mediators), thus hiding the complexity of dealing with semantics in a structured designed process.

Keywords: Business Process Modeling, Semantic Web Services, Software Engineering, Web Engineering, Model Driven Design.

1 Introduction

One of the most challenging promises of the Semantic Web Services (i.e., OWL-S [33], WSMO [35], WSDL-S [38] and SWSF [1]) is to enable the construction of flexible business applications, spanning over several enterprises and capable of dynamic composition for presenting to their clients the “best” business options. These emerging initiatives propose to augment Web Services with semantic annotations and to operate reasoners on top of such descriptions for performing business choices, thus matching user’s goals to system’s resources. In this paper we use WSMO, one of the emerging Semantic Web
Service initiatives, because it provides a clear separation between well identified components, such as ontologies, goals, Web services and mediators, and because it is founded on the two clear principles of strong decoupling and strong mediation. While dwelling into the deep aspects of the Semantic Web Services (e.g. for building a semantic execution environment such as WSMX [23]) is outside of the scope of this paper, we focus on the design and re-engineering of software components so as to meet the Semantic Web Services requirements; we show that advanced software engineering methods enable the development of software interfaces which are compliant with WSMO. Our method overcomes one of the most critical obstacles to the spreading of WSMO based solutions, i.e. the apparent difficulty of adapting current Web applications to WSMO requirements. Our work shows that there is no contrast – but actually good continuity – between modern model-driven software design methods and the Semantic Web Service concept; in particular, classical semantic descriptions available in the most advanced model-driven software design methods are quite compatible and powerful to deal with WSMO abstractions.

We exploit modern Web engineering methods, including visual declarative modeling (based on the WebML conceptual model), automatic code generation (locally executable using standard Web technologies and globally executable across enterprise boundaries by delegating the execution to a
Semantic Execution Environment such as WSMX), and automatic elicitation of semantic descriptions for the WSMO components (i.e., Ontologies, Goals, Web Services, and Mediators) from the design of the application. Global choreography\(^1\), front-end, and services implementations are derived from Business Process models and WebML models, whereas goals, descriptions of Web services (both in terms of capabilities and of their choreography interface), and descriptions of mediators are automatically generated. In particular, we propose to cover the different aspects of the design by means of the following techniques and notations:

- **High-level design of the global choreography of the interaction between services**: we adopt BPMN (Business Process Management Notation) to build process models, involving several actors possibly from different enterprises.
- **Design of the underlying data model of the cross-enterprise application**: we use extended E-R (Entity Relationship) diagrams enriched with OQL derivation rules (whose expressive power is equivalent to WSML Flight [12]) to model the local ontology of the application and to import existing ontologies

---

\(^1\) The term choreography assumes several meanings in different communities. We refer to W3C definition of choreography with the term “global choreography” (i.e., the choreography of an application made of WS), whereas we refer to WSMO choreography definition with the term “local choreography” (i.e., the choreography interface of a Web Service).
(possibly modeled in ontology engineering tools such as WSMT [29]), whenever they exist; we expose the resulting set of ontologies to the underling WSMX;

- **Design of Web services interfaces, of integration platform, and of application front end**: we use visual diagrams representing Web sites and services according to the WebML models [10], including specific hypertext primitives for Web service invocation and publishing [7] and explicit representation of workflow concepts [8].

In this way, instead of coping with textual semantic descriptions of Semantic Web Services, application developers will obtain them from the use of abstractions that are supported by software engineering tools. The use of description generators, sometimes helped by designer’s annotations, guarantees the benefits of Semantic Web Services at nearly zero extra-cost, thus positioning their cross-enterprise applications within an infrastructure that allows for flexible and dynamic reconfiguration.

The paper is structured as follows: Section 2 presents the running example that will be discussed throughout the paper; Section 3 offers a view of the related work and on the background of the research; Section 4 describes the overall methodology, presents the development process and addresses the main development steps, including the definition, the importing, and the query primitives for ontologies; Section 5 presents the proposed approach to the elicitation of semantic description of the application; Section 6 briefly outlines our implementation experience; and finally Section 7 presents our conclusions.

## 2 Background

Our approach relies on methodologies, tools and techniques from the fields of Software Engineering, Web Engineering, and Business Process Management.

### 2.1 Modeling Business Processes using BPMN

All the B2B Web applications implement a business process, which is represented by using a workflow model. Several notations have been proposed for workflow design. For the visual representation of workflows, we adopt Business Process Management Notation [9], which is associated to the BPML...
standard, issued by the Business Process Management Initiative. Various proposals are currently issued for serialization of workflow models in XML format. Among the most promising ones, we can cite XPDL [42].

The BPMN notation allows one to represent all the basic process concepts defined by the WfMC [41] model. BPMN activities extend WfMC activities, as they can express various behaviors (looping execution, compensation, internal sub-process structuring, event catching, and so on). BPMN activities can be grouped into pools, and one pool contains all activities that are to be enacted by a given process participant. Within a pool, we use BPMN lanes to distinguish different user types that interact with the specific peer. The flow of the process is described by means of arrows, representing either the actual execution flow, or the flow of exchanged messages. Another type of arrows represents the association of data objects to activities; these are meant just as visual cues for the reader, and do not have an executable meaning. Gateways are process flow control elements; typical gateways include decision, splitting, merging and synchronization points. Table 1 briefly summarizes the main visual constructs provided by BPMN. An example of full BPMN model is represented in Figure 4.

**Table 1.** BPMN main constructs.

<table>
<thead>
<tr>
<th>Gateways</th>
<th>Activities and Flows</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Or gateway" /></td>
<td><img src="image" alt="Activity" /></td>
</tr>
<tr>
<td><img src="image" alt="Xor gateway" /></td>
<td><img src="image" alt="Pool and Lanes" /></td>
</tr>
<tr>
<td><img src="image" alt="And gateway" /></td>
<td><img src="image" alt="Sequence flow" /></td>
</tr>
<tr>
<td><img src="image" alt="Start" /></td>
<td><img src="image" alt="Message flow" /></td>
</tr>
<tr>
<td><img src="image" alt="End" /></td>
<td><img src="image" alt="Data Association" /></td>
</tr>
<tr>
<td><img src="image" alt="Intermediate" /></td>
<td></td>
</tr>
</tbody>
</table>
2.2 Model-driven Web application design

Several approaches in the Web engineering field provide methodologies, conceptual models, notations, and tools for the design of Web applications. Among them, we can cite OO-HDM [36], OO-Method [20], Strudel [19], and WebML [10]. In this paper, we will adopt the WebML methodology. The specification of a Web application according to WebML consists of a set of models: the application *data model* (an extended Entity-Relationship model), one or more *hypertext models* (i.e., different site views for different types of users), expressing the navigation paths and the page composition of the Web application; the *presentation model*, describing the visual aspects of the pages. A hypertext site view is a graph of pages; pages are composed by units, representing publishing of atomic pieces of information, and operations for modifying data or performing arbitrary business actions (e.g. sending e-mails). Units are connected by links, to allow navigation, passing of parameters between the components, and computation of the hypertext.

The WebML conceptual model includes a service model comprising a set of *Web service units* [7, 32], corresponding to the WSDL classes of Web service operations, and components for workflow management and tracking. The Web services units consist of *Request-Response* and *One-Way* operations, which model services invocation and are triggered by the navigation of an input link; and of *Notification* and *Solicit-Response*, that are triggered by the reception of a message, thus they represent the publishing of a Web service, which is exposed and can be invoked by third party applications. The model supports both the *grounding* of Web services to the XML format of Web service messages, and *data-mediation capabilities*.

![Figure 1. Example of WebML hypertext model with invocation of remote service](image)
Figure 1 shows an hypertext example that includes the model of a Web service call and of the called service. Supply Area of Figure 1a is an area of a Web site for supply management. The employee can browse to the SupplySearch page, in which the SearchProducts entry unit permits the input of search criteria. From this information, a request message is composed and sent to the RemoteSearch operation of a Web service. The user then waits for the response message, containing a list of products satisfying the search criteria. From these options, a set of instances of Product are created, and displayed to the user by means of the Products index unit in the Products page; the user may continue browsing, e.g., by choosing one of the displayed products and looking at its details. Figure 1b represents the model of the RemoteSearch service invoked by the previously described hypertext. The interaction starts with the solicit SearchSolicit unit, which denotes the reception of the message. Upon the arrival of the message, an XML-out operation extracts from the local data source the list of desired products and formats the resulting XML document. The SearchResponse unit produces the response message for the invoking Web service.

WebML covers also the development of B2B Web applications implementing business processes, thereby supporting collaborative workflow-based applications, spanning multiple individuals, services, and
organizations. The data model is extended with the meta-data necessary for tracking the execution of the business process, by registering the execution advancement of cases (i.e., instances of processes) and activities as relational data. The hypertext model is extended by specifying activity boundaries and business-dependent navigation links. Activities are represented by areas tagged with a marker “A”; workflow links traverse the boundaries of activity areas and are associated with workflow logic: every link entering an activity starts the execution of the activity; every outgoing link ends the activity. If and switch units can be used to express navigation conditions. Distributed processes can be obtained by combining the workflow primitives with Web services primitives [8].

2.3 WSMO, WSML, WSMX and related design tools

The WSMO [17] initiative aims at providing a comprehensive framework for handling SemanticWeb Services which includes the WSMO conceptual model, the WSML language [12] and the WSMX execution environment [23], which also include design tools such as WSMT [29].

The Web Service Modeling Ontology (WSMO) is an ontology for describing various aspects related to Semantic Web services. The objective of WSMO and its accompanying efforts is to solve the application

---

2 Reference [8] uses an earlier notation where process metadata are managed using ad-hoc operations, instead of representing them explicitly with areas and links.
integration problem for Web services by defining a coherent technology for Semantic Web services. We briefly present the basic concepts of WSMO with respect to its conceptual specification. 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) [18]. 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. WSMO specifies different constituents as part of the description of an ontology, like concepts, relations, axioms, instances and so on. 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 properties, 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. Moreover, the interface of a Web Service describes the behavior of the Web Service from two perspectives: communication and collaboration. 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 to overcome 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, which is based on DLP [22, 25] as a least common denominator for description logic (DL) and logic programming (LP). WSML Core is separately extended in the directions of these two paradigms by the
variants WSML-DL and WSML-Flight, respectively. In particular WSML-Flight is an extension of WSML-Core which includes concepts, attributes, binary relations and instances, as well as concept and relation hierarchies and support for data types. Furthermore, it provides a powerful rule language. It adds features such as meta-modeling, constraints and non-monotonic negation. WSML-Flight is based on a logic programming variant of F-Logic [30] and is semantically equivalent to Datalog with inequality and (locally) stratified negation.

Web service Execution Environment (WSMX) is a framework for the automation of discovery, selection, mediation, and invocation of Semantic Web services. WSMX is based WSMO and, at the same time, it is a reference implementation of it. The framework also includes design tools, WSMT, for modelling ontologies and describing goals, Web Services and mediators.

2.4 Web Service discovery

Service Discovery is a normal task manually accomplished in many present business scenarios (e.g., finding a shipment service). Web Services certainly reduced the effort required in invoking a remote service exposed by a business partner, but their discovery (i.e., using UDDI [40]) always involves a human in the loop. Given that, in practice, automatic discovery of Web Services is not used.

Ontologies can play a key role in automating Web Service Discovery [39, 31, 34]. A conceptual framework within WSMO is presented in [31]; such framework provides both the reference architecture and flexible execution semantics for a WSMO compliant Discovery Engine. Reference [26] presents a concrete attempt in conceiving a conceptual model and a methodology that uses ontologies for better servicing requesters, by filtering relevant services and contracting so as to fulfil a given requester goal.

Glue [14, 15] is a novel approach that refines the conceptual model for WSMO Web Service discovery as well as the reference architecture and the related execution semantics. At a conceptual level Glue approach introduce the notion of class of goals and class of Web Service descriptions and gives mediators a central role in discovery. Glue approach recommends to use: 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, and wgMediators, as the conceptual element responsible for evaluating the matching. The resulting discovery mechanism is a composite procedure where the discovery of the appropriate mediators and the discovery of the appropriate services is combined. Glue concept was implemented upon an open source F-logic [30] inference engine called Flora-2 [43] that runs over XSB, an open source implementation of tabled-prolog and deductive database systems.

2.5 The trend towards Business Processes

Current research efforts are converging on the proposal of combining Semantic Web Services and Business Process Management to create one consolidated technology, which we call Semantic Business Process Management (SBPM) [24]. The claim is based on the observation that the problem of mechanization of BPM, i.e. the reduction of human intervention in associated tasks, can be traced back to an ontological problem, i.e. the lack of machine-accessible semantics, and that the modeling constructs of SWS frameworks, especially WSMO, are a natural fit to creating such a representation.

This paper follows this trend towards SBPM and it refines our previous work [4] focusing on the development process (see section 4.1 and 4.2), explaining how we manage ontologies (see section 4.3) and providing a set of advance primitives for querying the ontologies (see section 4.4).

3 Running example

For the discussion we will consider a running example derived by the Purchase Order Mediation scenario and the Shipment Discovery scenario proposed at the Semantic Web Service Challenge 2006 [16], properly extended to show all the components of a classical B2B application. In this scenario, two big companies, Blue and Moon, need to integrate their processes in order to create a new business partnership. In summary, as displayed by Figure 2, the architecture includes the two companies Blue and Moon, the mediation service between them, and a general-purpose Web service built by Blue for interacting with external services and an external discovery engine.
Blue usually handles its purchase orders towards its partners by using a standard RosettaNet PIP 3A4 conversation, while the Moon partner offers a set of Legacy Web Services for products purchase. Blue employees, in the Purchase department, want to “talk” in a transparent way with their counterparts in the Moon partner – using their usual RosettaNet Purchase Order Interface, therefore a mediation component is needed between the two. The mediator is in charge of (i) transforming the single RosettaNet message (containing all the order details) to the various messages needed by Moon to create and handle a purchase order; and (ii) of translating the set of confirmation messages by Moon into a whole RosettaNet Purchase Order Confirmation to be sent back to Blue. Thus, the mediation requires a data mediation and a relevant process mediation between the two totally different purchase processes of RosettaNet and Moon legacy system. The RosettaNet Purchase Order Interface offers the possibility to import products ontologies published by partners and select from them the products to be added to the Purchase Order. After

**Figure 2.** The B2B scenario derived from the Semantic Web Service Challenge 2006.
completing the purchase of a set of products, Blue employees organize the shipment of the products through the *Shipment Organize Interface*. This interface relies on a Web Service internally developed by Blue and offered to Blue partners too. The internal orchestration of the Web Service relies on a WSMX compliant *Discovery Engine* for retrieving available shipment services, and hence needs to describe the shipment goal according to the WSMO standard. When the Discovery Engine returns a list of Web Services offering a shipment service compatible with the original Goal, the Shipment Web Service invokes the returned Web Services to obtain actual shipment offers and proceeds with its orchestration.

4 Design of Semantic Web Service applications

This section describes our proposal for semi-automatically generating WSMO-compliant semantic specifications of a Web application. Our approach extends the WebML methodology presented in section 3.3 towards the design of semantic Web services and Web applications.

4.1 Development process

The phases of the development process of a semantic Web application are shown in Figure 3. 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.

![Figure 3. Phases in the development process of data- and process-intensive Web applications.](image-url)


**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 importing of existing ontologies. **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 3 are outside the scope of this paper. A comprehensive description of them can be found in [10].

### 4.2 Design of the business process

The business process (BP) design task, focusing on the high-level schematization of the processes underlying the application, results in one or more Business Process diagrams. The Business Process design is an high level conceptual task. The resulting BP diagram is a representation of the sequences and constraints among the tasks to be executed at abstract level, therefore it does not reflect the structure of the application yet. Business Processes can be internally structured using a variety of constructs: sequences of activities, AND, OR, XOR and complex conditional gateways for splitting and merging of branches, iterations of one or more activities, pre- and post-conditions, and so on.
Once the specification of the control flow is defined, the designer can annotate the model by selecting the kind of expected implementation of each activity and service. In particular, with respect to service invocation, it is possible to specify whether the service has to be modeled and implemented internally (**internal**), shall be invoked as a non-semantic already implemented service (**external**), or needs to be searched and invoked by means of semantic discovery and invocation techniques (**semantic**).

**Figure 4.** Workflow representing the interaction of the running example (BPMN notation).
The BP diagram of the running case is represented in Figure 4: for sake of clarity, the process is split into two sub-processes: part (a) describes the purchase and part (b) describes the shipment management. Notice that the two processes shall be executed sequentially in the full scenario.

We assume that the MOON services are external, the mediator services are internal, and the shipment services are semantic.

In the following, we will exemplify the design of the mediator of part (a), and the extraction of ontology, capability and choreography of part (b). Note that the two workflow diagrams have well-defined workflow semantics (once iteration conditions and conditional branches are fully specified), while the representation of Figure 2 doesn’t; building a BPMN (or equivalent) representation of the business process is a first step of the model driven design of any complex software system.

4.3 Design of the data model and extraction of the ontologies

The elicitation of the ontologies involved in the application is addressed by four distinct steps, each addressing different aspects of the application ontology:

1. First, existing remote ontologies, possibly provided by third parties, can be imported.
2. Then, the data model is considered as a piece of ontology. This means that an appropriate transformation of the WebML data model transform the extended E-R internal representation of the data into a WSMO-compliant ontology, which is then registered on the WSMX resource manager [23]. The expressive power of the WebML E-R model can be compared to the WSML-Flight language. Table 2 shows the comparison of the expressive power of WebML in respect to OWL-DL and WSML-Flight extending the comparison presented in [13]. As WSML-Flight, the WebML data model

Table 2. Expressive power comparison between WebML data model and other ontology languages.

<table>
<thead>
<tr>
<th>OWL Abstract Syntax</th>
<th>DL syntax</th>
<th>WSML-Flight</th>
<th>WebML E-R and OQL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Axioms</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class(A partial C_1 \ldots C_n)</td>
<td>A \subseteq C_i</td>
<td>+</td>
<td>+ (C \neq \perp; A \neq T)</td>
</tr>
<tr>
<td>Class(A complete C_1 \ldots C_n)</td>
<td>A = C_1 \cap \ldots \cap C_n</td>
<td>+</td>
<td>+ (A, \perp \neq \perp; T)</td>
</tr>
<tr>
<td>EnumeratedClass(A o_i \ldots o_n)</td>
<td>A = {o_i, \ldots , o_n}</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SubClassOf(C_1 \subset C_2)</td>
<td>C_1 \subseteq C_2</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>EquivalentClasses(C_1 \ldots C_n)</td>
<td>C_1 = \ldots = C_n</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>DisjointClasses(C_1 \ldots C_n)</td>
<td>C_1 \cap C_2 \subseteq \perp</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>ObjectProperty(R super(R_1) \ldots \supersub(R_n))</td>
<td>R \subseteq R_1</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>domain(C_1) ... domain(C_n)</td>
<td>\forall R \cdot C_i</td>
<td>+</td>
<td>+ (C \neq \perp)</td>
</tr>
<tr>
<td>range(C_1) ... range(C_n)</td>
<td>\forall R \cdot C_i</td>
<td>+</td>
<td>+ (C \neq \perp)</td>
</tr>
<tr>
<td>[InverseOf(R)]</td>
<td>R = R^\top \subseteq \perp</td>
<td>+</td>
<td>- (R and R ' with the same name)</td>
</tr>
<tr>
<td>[Symmetric]</td>
<td>R \subseteq R^\top \subseteq \perp</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>[Functional]</td>
<td>R \subseteq \perp \subseteq \perp</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>[InverseFunctional]</td>
<td>Trans(R)</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>[Transitive]</td>
<td>R \subseteq R \subseteq \perp</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>SubPropertyOf(R_1 \ldots R_n)</td>
<td>R_1 \subseteq R_2</td>
<td>+</td>
<td>+ (domain(R), range(R) \neq T)</td>
</tr>
<tr>
<td>EquivalentProperty(R_1 \ldots R_n)</td>
<td>R_1 \equiv \ldots \equiv R_n \subseteq \perp</td>
<td>+</td>
<td>+ (domain(R), range(R) \neq T)</td>
</tr>
<tr>
<td>Individual(o type(C_1) ... type(C_n))</td>
<td>o \in C_i</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>value(R, o_i) ... value(R, o_n)</td>
<td>{o_i, o_n} \in R_i</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>SameIndividual(o_i \ldots o_n)</td>
<td>o_i = \ldots = o_n</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DifferentIndividual(o_i \ldots o_n)</td>
<td>o_i \neq o_j, i \neq j</td>
<td>-***</td>
<td>-***</td>
</tr>
</tbody>
</table>

**Descriptions (C)**

<table>
<thead>
<tr>
<th>URI Reference</th>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>owl:Thing</td>
<td>A</td>
<td>+</td>
</tr>
<tr>
<td>owl:Nothing</td>
<td>T</td>
<td>-</td>
</tr>
</tbody>
</table>

| intersectionOf(C_1 \ldots C_n) | C_1 \cap \ldots \cap C_n | + | rhs** |
| unionOf(C_1 \ldots C_n) | C_1 \cup \ldots \cup C_n | + | rhs** |
| complementOf(C_i) | -C_i | - | - |
| oneOf(o_i \ldots o_n) | \{o_i \ldots o_n\} | - | - |

| restriction(R someValuesFrom(C)) | \exists R, D | rhs* | rhs** |
| restriction(R allValuesFrom(C)) | \forall R, D | rhs* | rhs** |
| restriction(R value(o)) | \exists R, o | + | rhs** |
| restriction(R minCardinality(1)) | \geq 1 R | - | rhs** |
| restriction(R minCardinality(o))(\neq 1) | \geq o R | - | rhs** |
| restriction(R maxCardinality(o)) | \leq o R | - | rhs** |

* May only be used on the left-hand side (as the first argument) of SubClassOf
** May only be used in partial class definitions and on the right-hand side (as the second argument) of SubClassOf
*** Notice that already all individuals in an WSML Flight knowledge base and WebML repository are different; therefore, an explicit assertion would be superfluous
comprises a rule language called WebML-OQL, which allows for calculating derived information, but not for defining constraints. Moreover, as WSML Flight, WebML doesn’t have neither Thing nor Nothing concepts, but, differently from WSML Flight, it cannot deal with equality related constructors (i.e., EquivalentClasses, SameIndividual, DisjointClasses, Different Individuals, and Functional/Inverse Functional properties). Anyway, the similarity between WebML extended E-R and WSML-Flight is enough to enable a partial extraction of WSMO descriptions.

3. Then, the process ontology is extracted from the BPMN specification of the underlying business process of the application. The elements of the workflow model (e.g., activity names, lanes, and so on) are extracted as semantic concepts and used to build an additional piece of the ontology that will be useful in defining the state signature of the choreography interfaces of the Web services;

4. Finally, the BPMN model and the WebML data model are annotated with concepts imported from existing ontologies.

This approach is oriented towards T. Berners-Lee vision for Web applications connected by concepts annotations [2], since it provides a framework for importing existing ontologies and defining new ones, thus sharing knowledge among the different applications on the Web.

Figure 5 shows the E-R diagram used by the Shipment Web Service to describe goals and to invoke external Shipment Web Services. The E-R has three main domain entities Shipment, describing each shipping, ShipmentService, describing BLUE shipment partners, and Location, describing the geographical entities involved in the shipment process. The E-R diagram includes also the two process entities used to describe the status of the process (see Section 3.3). Each Shipment instance is related to a ShipmentService, to a origin and a destination Location and, finally to an Activity to indicate its current state. The ShipmentService entity is connected to the Location entity through the shipTo relationship, i.e. each shipment partner has a set of possible shipment locations, and through the hasLocation relationship, each carrier has a set of valid pick up points. Both the Location and the ShipmentService entities have some sub entities in order to easily specialize their characteristics.
The process of WSML ontologies generation starts by importing external ontologies used in the WebML E-R model to enrich WebML data types definitions. Then, for each entity in the E-R, a correspondent WSML concept is generated with its super direct concept, attributes (also E-R relationships are mapped to attributes) and possible axioms. Axioms are derived from WebML OQL expressions defined on the data. WebML E-R model can be easily converted to a WSML-Flight ontology maintaining all its constraints. For instance, the `EuropeanShipmentService` entity is a sub entity of the `InternationalShipmentService` that is located in `Europe`. This subentity is described in the WebML-OQL syntax as:

```
InternationalShipmentService(as SuperEntity) where
InternationalShipmentService.hasLocation isa Europe.
```

A fragment of the WSML-Flight translation of the E-R in Figure 6 and its WebML-OQL constraints is:

```plaintext
concept ShipmentService
   hasName ofType _string
   hasLocation ofType (0 *) Location
...
concept EuropeanShipmentService subConceptOf InternationalShipmentService
   nfp dc#relation hasValue { EuShipmentServiceDef } endnfp
axiom EuShipmentServiceDef definedBy
```
The process of WSML ontologies generation starts by importing external ontologies used in the WebML E-R model to enrich WebML data types definitions. Then, for each entity in the E-R, a correspondent WSML concept is generated with its super direct concept, attributes (also E-R relationships are mapped to attributes) and possible axioms.

### 4.4 Querying ontologies

A large part of the design of Semantic Web applications (i.e., WSMO semantic description, extraction of ontologies) can be performed by means of existing design tools and primitives, some additional elements might be needed for addressing complex query and reasoning over native semantic data.

Hence we enhanced WebML modifying existing primitives and introducing new ones to allow a native interaction with ontologies. The aim of this new primitives that are able to perform query over ontological data is not too replace the original ones: even if part of the information underlying the system is represented by ontological knowledge, we think that for substantial sections of Web applications conventional (e.g., relational) representation of data is still effective. Therefore, we claim that integrating ontologies and relational data and allowing seamless interaction between them is a desiderata of Semantic Web applications. Thanks to the unit-based model of WebML, where each unit exchanges data with other units by passing parameters over links, interaction between ontology's instances and database instances can be quite straightforward. The main asset of WebML, i.e., its ability of effectively capturing the parameters passed between the different components (units) of the Web application, is still valid even when we are considering Semantic Web Applications. Indeed, it does not matter if the underlying data are extracted from ontologies or databases; a Web application is anyway composed by the same hypertextual primitives. Parameters on the links become the actual contact point between traditional and semantic data. In this way, we achieve a good orthogonalization between data model and hypertext model of the applications.
4.4.1 Basic primitives for queries on ontologies

Basic WebML primitives for data access can be exploited for queries over ontologies: Index unit, Multidata unit, and Data unit are traditionally used to navigate database instances, but their semantic is general purpose, and hence their paradigm is suitable to navigate and show ontology instances too. They only need to be extended to support different kinds of data sources (in particular, ontologies) and some additional options for the queries. For instance, the Index unit can be used to produce lists of instances of a particular class of the domain ontology. A needed feature is the possibility to show only direct instances or also inferred instances. Moreover, a Index unit over an ontology class may show also the direct or the inferred subclasses of the selected class. The developer can choose at design time the combination he prefers. The same discussion applies to Multidata unit and Data unit.

WebML already introduced the concept of Hierachical Index unit, which acquires a first class role in the context of Semantic Web applications. Hierachical Index unit is ideal to publish a portion of a ontology: i.e., the hierarchic tree underlying a certain class comprising its subclasses and instances.

4.4.2 Advanced primitives for queries on ontologies

The evolution of the basic data access primitives, introduced in the previous paragraph, is not enough to exploit the rich set of possible queries over semantic instances and classes. Therefore, we introduce a new set of operational primitives to describe advanced queries over ontological data. 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, depending on the type of parameters, execute differently. Table 3 summarizes the syntax and the semantics of such units.
Note that these units are natively able to manage ontological data sources, therefore they can manage concepts and references by means of the standard uri (Uniform Resource Identifier) mechanism, while the original WebML units only rely on OID identification mechanisms.
Besides the units for ontological data query, we introduce a unit to compose the set resulting from obtained from these units (Figure 6). The Set Composition operation unit is able to perform classic set operations over 2 input sets of uris (i.e., union, intersection, difference). The operations are performed by taking into account the hierarchical structure of the uris involved in the operation: e.g. suppose we have two set of classes A={wine, liquor, food} and B={drink, food}, and assume that the drink class is a superclass of wine and liquor; then A INTERSECTION B gives {wine, liquor, food} and A UNION B gives {drink, food}. The Import Ontological Source unit adds a remote or local data source that must be consistent with ontological model of the Web application (it's validated against it before being added to the ontology model). It is also possible to express the url of an ooMediator to be used to import the ontology and convert it to the internal ontological model. The Describe unit returns the RDF description of an uri, thus enabling automatic Web page annotation and data exporting.

Figure 7 shows a portion of the Purchase Order Interface used to navigate the Product ontology and to import ontologies provided by business partners in the contest of our B2B scenario. In the Import Ontologies page, users can fill the form with url to remote ontologies and to the suitable ooMediator, if the ontology provided by the business partner is not consistent with the internal ontological model. When the user submits the filled form, the Import Ontology unit uses the data in the form to acquire the ontology and eventually to call the ooMediator needed to import the new ontology. Available ooMediators are published by a WSMX repository and designed with the tool WSMT. Once an ontology datasource is successfully imported, the user is redirected to the Select Product page, where pre-existing data and newly imported are visible. In the Select Product page, the hierarchical unit Products Hierarchy shows a tree with the class hierarchy pertaining to the concept Product: all its direct and derived
Subclasses are shown on the tree. When the user selects one of the concepts in the class hierarchy, the Index unit *Product Instances* is populated with all the instances of the selected concept. From the Index unit *Product Instance*, the user can navigate to the *Product Details* page where the Data unit *product* shows details on the selected product according to its class and the related properties. The Index unit *Accessories* shows all the product instances and classes that have has the current product *uri* as value of the property *isAccessoryOf*. Finally the user can view all the classes to which the current product instance belongs: the *uri* of the current product instance is passed to the InstanceOf unit that extracts all the direct and derived classes corresponding to the passed instance. Hence the set of classes *uri* returned by the InstanceOf unit is used to populate the Index unit *Related Product Classes*.

### 4.5 Design of the service and the user interfaces in WebML

Once the business process has been designed, workflow constraints must be turned into navigation constraints among the pages of the activities of the hypertext and into data queries on the workflow
metadata for checking the status of the process, thus ensuring that the data shown by the application and user navigation respect the constraints described by the process specification. This applies both to the human-consumed pieces of contents (i.e., site interfaces) and to the machine-consumed contents (i.e., Semantic Web Services interactions).

A flexible transformation, depending on several tuning and styling parameters, has been devised for transforming workflow models into skeletons of WebML hypertext diagrams. The produced WebML model shall consist of application data model, workflow metadata, and hypertext diagrams. Since no a-priori semantics is implied by the activity descriptions, the generated skeleton can only implement the empty structure of each activity along with the hypertext and queries that are needed for enforcing the workflow constraints. The designer remains in charge of implementing the internals of each activity. Additionally, it is possible to annotate the activities, thus allowing the designer to map the activity to a coarse hypertext that implement the specified behavior on the specified data. The designer is in charge of refined specification of the hypertexts. For instance, Figure 8 shows a possible WebML specification of the hypertext of the Blue Shipment service.

Figure 8. The Blue Shipment Web Service.
In the upper part of Figure 8, the **ShipmentRequest** is received by *searchShipmentRequest* and is passed to the **Goal Composition** that fill a Goal Template with the required instances, obtaining a goal description for the WSMX compliant Discovery Engine; the generated goal description is passed to the **Send Goal** which sends the goal to a Web Service exposed by the Discovery Engine. The Discovery Engine returns a result with a set of Web Services compatible with the original shipment goal and for each Web Service the description of the Lowering and Lifting operations by an appropriate XSLT stylesheet, the results of the Web Service call are stored by *Store Goal Result*. Then, for each valid Web Service returned, a request for a shipment offer is made by *WS Offer Invoker*. The XSLT stylesheets for the **Lowering** and **Lifting** are selected for each valid Web Service according the results returned by the Discovery Engine. Each offer returned by a Shipment Web Service is stored by *Add Offer*. The results for the current Shipment Request are selected by *Extract Valid Offers* and converted to the Blue data model by *Lowering* and the set of offers is returned the service requester by *SendShipOffer*. Once the service requester selects one of the offers and he sends the **ConfirmShipOfferRequest** to the **confirmShipOfferRequest** (lower part in the Figure 8), the previously stored offer is retrieved (*Extract Confirmed Offer*), hence the offer purchase message is composed by *Lowering* with an appropriate XSTL stylesheet and WS Purchase Invoker sends the message to the Shipment Web Service. Finally *Send ShipmentConfirmation* sends a confirmation message to the service requester according to the answer received by the invoked Shipment Web Service.

**Figure 9.** The Blue Web interface to organize shipments for successful orders
Figure 9 shows a WebML hypertext model representing a fragment of the Blue Web application: a home page called Select Order to Ship allows the user to choose an Order (with Status “Not shipped”) from the Order List index unit. When an order is chosen, the “S” link starts the Organize Shipment activity, showing the Order Details data unit in the Organize Shipment page, together with a form (Search Shipment Offers). The data submission triggers the invocation of a remote service (searchShipmentOffers request-response unit), whose results are lifted by the storeShipmentOffer XML-in unit. The activity is completed (link “C”) and following one is started. The Select Shipment Offer page is shown, containing a list of Shipment Offers (an index unit displaying the results of the service). The user chooses an offer and thus triggers the confirmShipmentOffer request response, whose results are stored locally. Finally, the user is redirected to the home page.

One should notice that this high level visual programming of Web applications, although of lower level if compared to BPMN specifications, is not concerned with many implementation details, which are left to the design tool, capable of deploying the code for arbitrary choices of data management and Web server architecture. Moreover, this visual programming style is natively supporting change management and reengineering.

5 Extraction of the WSMO description of the Web Services

In this section we show that the WSMO-compliant description of Web services can be semi-automatically derived from the design specification.

Figure 10 summarizes the extraction of semantic description of services from the application design. The design flow represents the main steps of the development process presented in Section 4.1. The various step produce some intermediate artifacts (BPMN models, WebML skeletons, data models, hypertext models), possibly enriched by imported ontological descriptions (on top of Figure 10) and are exploited for devising the set of WSMO specifications (at the bottom of Figure 10): the description of the mediator can be extracted from the hypertext describing the mediator (see Section 5.4); the Web services capability
description is derived from hypertext model; the choreography information is derived from BP model; and the user goals (Section 5.3) are derived from the BP model and the hypertext model.

5.1 Extraction of Web Services capabilities

The BPMN and WebML models of the Web services provide enough information for describing its behaviour. Assuming a BPMN activity as an atomic Web service call, we can exploit the BPMN data flow for providing good hints for the extraction of inputs and outputs of the service. Indeed, the data flow specifies the objects that are passed between the various activities. By isolating a single activity, it is possible to extract the WSML pre-conditions (inputs) and post-conditions (outputs).

WSML pre-conditions are obtained from the first unit of WebML chain describing a Web Service operation (Solicit Unit), while post-conditions are obtained from the last one (Response Unit). These two units contain information about the exact structure of the exchanged message and eventually the mapping of message elements to the domain model and hence to the extracted ontologies (see Section 4.4). Effects are extracted searching for WebML units that modify or create instances of entities that are related to the activities involved by the process described in WebML Web Service. Shared variables are obtained from the different generated conditions by grouping all the variables involved in the operations data flow.

Figure 11 shows a portion of the WebML XML serialization for the Blue Shipment Web Service, introduced in Section 4.4 and its correspondences with the automatically generated WSML description of
its capabilities. In particular, Figure 11 illustrates how parameters of the Solicit Unit in WebML (SolicitParameter) corresponds to WSML pre-conditions, and how parameters of the Response Unit in WebML (ResponseParameter) corresponds to WSML post-conditions. The WebML XML serialization contains also a reference to the XSD file describing the structure of the exchanged message (shipment.xsd) while the WSML description contains information about the structure of the exchanged messages (surrounded by a green box): both these parts are obtained from the data model introduced in Section 4.3.

<table>
<thead>
<tr>
<th>The WebML XML description of the Blue Shipment Web Service</th>
<th>The automatically generated WSML description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;SERVICEVIEW id=&quot;wsv3&quot; name=&quot;ShipmentWS&quot; secure=&quot;no&quot;&gt;</td>
<td>capability</td>
</tr>
<tr>
<td>&lt;PORT id=&quot;port3&quot; name=&quot;searchShipOffer&quot; secure=&quot;no&quot;&gt;</td>
<td>sharedVariables(Req)</td>
</tr>
<tr>
<td>&lt;OPERATIONUNITS&gt;</td>
<td>precondition</td>
</tr>
<tr>
<td>&lt;SOLICITUNIT id=&quot;slu6&quot; name=&quot;searchShipmentRequest&quot; nsURI=&quot;blue&quot;&gt;</td>
<td>definedBy</td>
</tr>
<tr>
<td>&lt;SOLICITPARAMETER id=&quot;slp3&quot; name=&quot;ShipmentRequest&quot; type=&quot;ShipmentObject [element]&quot; xmlSchema=&quot;shipment.xsd&quot;/&gt;</td>
<td>(Req.memberOfShipmentRequest)</td>
</tr>
<tr>
<td>&lt;SOLICITUNIT id=&quot;slu7&quot; name=&quot;ConfirmShipOfferRequest&quot; nsURI=&quot;blue&quot;&gt;</td>
<td>postcondition</td>
</tr>
<tr>
<td>&lt;SOLICITPARAMETER id=&quot;slp4&quot; name=&quot;ConfirmShipOfferRequest&quot; type=&quot;ShipmentOfferConfirmation [element]&quot; xmlSchema=&quot;shipment.xsd&quot;/&gt;</td>
<td>definedBy</td>
</tr>
<tr>
<td>&lt;/SOLICITUNIT&gt;</td>
<td>(Req)</td>
</tr>
<tr>
<td>&lt;RESPONSEUNIT id=&quot;rsu6&quot; name=&quot;Send ShipmentOffer&quot; nsURI=&quot;blue&quot;&gt;</td>
<td>precondition</td>
</tr>
<tr>
<td>&lt;RESPONSEPARAMETER id=&quot;rsp5&quot; name=&quot;ShipOfferContainer&quot; type=&quot;ShipmentOffers [element]&quot; xmlSchema=&quot;shipment.xsd&quot;/&gt;</td>
<td>(Req)</td>
</tr>
<tr>
<td>&lt;RESPONSEUNIT id=&quot;rsu8&quot; name=&quot;Send ShipmentConfirmation&quot; nsURI=&quot;blue&quot;&gt;</td>
<td>precondition</td>
</tr>
<tr>
<td>&lt;RESPONSEPARAMETER id=&quot;rsp6&quot; name=&quot;ShipmentConfirmation&quot; type=&quot;ShipmentConfirmation [element]&quot; xmlSchema=&quot;shipment.xsd&quot;/&gt;</td>
<td>(Req)</td>
</tr>
<tr>
<td>&lt;/RESPONSEUNIT&gt;</td>
<td>precondition</td>
</tr>
<tr>
<td>&lt;/OPERATIONUNITS&gt;</td>
<td>&lt;/PORT&gt;</td>
</tr>
<tr>
<td>&lt;/SERVICEVIEW&gt;</td>
<td>&lt;/SERVICEVIEW&gt;</td>
</tr>
</tbody>
</table>

Figure 11. A portion of the WebML XML serialization and the generated WSML description.

4.6.2. Extraction of the service choreography

The service choreography is a piece of information that typically requires some annotation by the designer, in order to establish all the possible interaction sequences with the service. However, at least one of the choreography sequences can be extracted from the BPMN model, by analyzing the order of invocation of the different operations of the service. Obviously, this does not guarantee that all the possible scenarios are considered, since only one enactment can be analyzed. The extraction of this kind of information is rather simple: provided that a lane describes a single Web service, we can assume that all the control flow links traversing its borders contribute to specifying a possible invocation order of the operations, i.e., a choreography interface of the Web service.
The following is the WSML description of the Web Service choreography interface automatically generated.

```xml
interface
  choreography
  stateSignature
  in
    ShipmentRequest withGrounding [...]
    ConfirmShipOfferRequest withGrounding [...]
  out
    ShipOfferContainer withGrounding [...]
    ShipmentConfirmation withGrounding [...]
  controlled oasm#ControlState
transitionRules
  forall {?x, ?state} with {?
    state[oasm#value hasValue oasm#InitialState] memberOf oasm#ControlState and
    ?x memberOf ShipmentRequest
  } do
    add(state[oasm#value hasValue ShipmentOfferRequested])
    delete(state[oasm#value hasValue oasm#InitialState])
    add(_# memberOf ShipmentOfferContainer)
  endForall
  forall {?x, ?state} with {
    state[oasm#value hasValue ShipmentOfferRequested] and
    ?x memberOf ConfirmShipOfferRequest
  } do
    add(_# memberOf ShipmentConfirmation)
  endForall
```

5.2. Extraction of the service orchestration

The process that leads to the extraction of the service orchestration is very similar to the previously introduced process for the extraction of the service choreography. While the WSMO choreography describe the supposed order of call of the operation provided by a Web Service, the WSMO orchestration describes how a published Web Service internally interacts with other Web Services. At the moment the language to describe the WSMO orchestration interface [37] is still under development, but we already know that it will be based on the same ASM model used to describe WSMO choreography interface. For example, taking in account the second lane of the BPMN diagram in Figure 4a, that represents the Web Service that provides the implementation of a wwMediator, we are able to automatically extract the orchestration sequences considering all the control flow links traversing its borders with the bellow lane, containing the operations of the remote Web Service.
5.3 Extraction of user’s goal

Extraction of user’s goals can be performed combining information available at the BPMN level with information available at the WebML level. A first level of goal elicitation can be achieved by extracting the sequence of conditions and objects passed to the Web services by the user’s lane in the BPMN diagram.

A deeper level of details requires using the WebML hypertext models and analyzing the semantics embedded in the navigation and composition of the pages. Such refined goal is detailed in terms of the tasks performed by the user and of the data manipulated, thus increasing the significance of the WSMO goals that can be generated. Goals are generated in correspondence with Web Service call to provide a semantic description of the objective persecuted by the user/machine invoking the Web Service. In the scenario we introduced, for example, a Goal template description is automatically generated a design time and populated at runtime to describe the user’s aims implied by the Organize Shipment Interface interaction with the Shipment Web Service. In particular, the shared variables are extracted from the data instances passed to the Web Service operation call, while the effects are obtained considering the WebML operations modifying data associated to the extracted shared variables. For example, the XML-In unit in Figure 9, stores the confirmed shipment orders creating a shipment data instance with a particular status. The following is the WSML description of a Goal automatically generated.

```
capability
sharedVariables ?shipment
postcondition definedBy
  ?shipment [ 
    requestedPickupLocation hasValue stanford, 
    requestedDeliveryLocation hasValue rome, 
    requestedDeliveryDateTime hasValue _date(2006,4,28), 
    goodDimension hasValue _d [ 
      l hasValue 10, 
      w hasValue 50, 
      h hasValue 100] memberOf dimension, 
    maxAcceptedShipmentPrice hasValue 50 
  ] memberOf requestsShipmentService.
```

Given that goals capture user’s needs, this part of the process is perhaps the least easy to automatize, and manual verification and intervention is generally required.
5.4 Design of wwMediators with WebML

One of the main strength points of the approach is the ease of design and implementation of complex wwMediators. If a lane is identified as a wwMediator at the BPMN level, the basic information about the design of the mediation services can be extracted from the high-level BPMN description of the interactions (in particular, information about possible choreography of the service and basic interface and parameter specification). The skeleton model of the mediator is automatically generated and the designer can refine it at a conceptual design level. Then, the WSMO description of the mediator can be derived from the WebML diagrams.

![Diagram of WWMediator Web Service](image)

Figure 12. The WebML model of wwMediator Web Service.
Figure 12 presents the detailed specification of the wwMediator within WebML. This specification can be used to generate a working Web Service providing mediation between Blue and Moon Web Service. The WebML specification includes some Lowering and Lifting operations corresponding to WSMO ooMediators and provides mediation between the data model of the source Web Service and the destination one. In WebML this mediation consists in XSLT stylesheets generated by a graphic tool described in Section 6.1.

Note that the mediation activities are in one-to-one correspondence with the BPMN activities of Figure 4. The chain of operation consists of: (a) receiving a Web service call invocation; (b) transforming input application data into suitable internal format (lifting); (c) storing internal application data; (d) transforming application data into suitable format for the subsequent Web service call (lowering); (e) calling the next Web service in the chain; (f) tracking the status change of the wwMediator (with no manual intervention).

At deploy time WebML the following WSML description of the wwMediator need to register it on the WSMX runtime is generated:

```xml
wwMediator ...
  nonFunctionalProperties
dc#type hasValue "http://www.wsmo.org/2004/d2/v1.2/#mediators"
endNonFunctionalProperties
source "http://blue.org/RosettaPurchaseOrder"
target "http://moon.org/OMService"
```

6 Implementation

A proof of concepts of the integration with the semantic aspects discussed in this paper has been presented at the SWS Challenge 2006 [5, 16]. The two phases of the challenge allowed us to prove the advantages of a Software Engineering approach to Semantic Web Services design. In particular, in the first phase we presented the WebML design and implementation of the wwMediator of the running case addressed in this paper (Figure 12) and the usage of the CASE tool WebRatio in the context of Semantic Web applications. Moreover, we integrated Glue WSMO discovery engine by invoking the Web Services
that Glue exposes both for publishing a new shipment service and for discovering the most convenient shipment service. In the second phase we proved to be able to deal with changes without writing additional code, but only modeling the additional requirements and exploiting the same framework adopted in the first phase.

6.1 WebRatio

The presented approach relies on solid implementation of the background concepts: the WebML methodology is supported by a commercial CASE tool called WebRatio (www.webratio.com), providing visual design interfaces and automatic code generation. The architecture of the tool enabling semantic Web design is an extension of the WebRatio architecture [11]. At compile time (Figure 13, top), each application specification is used to generate an XML descriptor for each unit, link, Web page, service call, and service publishing primitive. At runtime (lower part of Figure 13) the application is instantiated and executed. Clients issue requests to the server, which in turn calls the execution engine; the generated code includes customized JSP tags that define the placement of the WebML units into the pages. The content required by the user is assembled by the WebML runtime component, using the descriptors and the data sources in the Data Layer.

Web service calls are managed by a SOAP sender/listener, and a Conversation Manager for conducting long-running interactions with a service.
Design support for the grounding of services is provided too. In particular, we developed a prototype mapping tool for visual lowering and lifting of data. The tool allows to design content transformation from local domain ontologies to XML format and vice-versa. With this tool we don’t aim at covering as much as possible of query power. What is needed is a simple and intuitive, yet formal, modelling of typical XML transformations in the context of semantic Web. The designer can ground a service to an XML implementation by specifying a set of lowering and lifting transformations defined by drawing simple lines between the origin and target format. The full description of the tool can be found in [6], while a snapshot of the GUI is visible in Figure 14. The information about the transformation drawn by the designer is stored into an XML file, which is used to create appropriate XSL files to be used in Adapter Units within WebML applications, for applying transformations on the actual XML documents.
6.2 BPMN editor and WebML generator

The modeling of the business process requirements and their transformation into WebML skeletons are implemented in a prototype tool [3] that allows to design BPMN diagrams and to automatically generate the corresponding WebML hypertext models. The workflow schema is stored as an XML document and WebML models are generated using XSLT technology. In addition, the transformation takes into account the annotations that the designer provided on the kind of implementation of the services (internal, external, semantic). Generation rules for the hypertext have been built based on the experience and the theory presented in [8]. In particular, the tool produces a WebML model in terms of application data model, workflow metadata, hypertext diagrams, and involved services implementation. In case of internal implementation of services, the transformation prepares the mockup of the service implementation too; if the implementation is external, a simple SOAP invocation of an endpoint is devised; if a semantic service is needed, the whole cycle of discovery, mediation, and invocation is generated. Once the transformation has been accomplished, the result can be opened automatically within WebRatio, thus allowing the designer to refine the generated hypertext and to implement the internals of each activity.

6.3 WSML descriptions generators

For validating our approach, we developed several prototypical transformers that generate WSMO-compliant descriptions of Web applications and services starting from WebML models of the applications.
and BPMN specifications of the processes. The pieces of WSMO specification presented in Sections 4.3, 4.5, and 5.3 are samples of the generated output of the transformations. Since both WebRatio and the BPMN editor are designed to be flexible and extensible, and they can manage user-defined properties of objects and new XSLT transformations of the workflow models, we added our generation engines for semantic description of the application directly to the tools.

6.4 GLUE discovery engine

Web Service discovery has been addressed by black-box invocation of Glue WSMO discovery engine which 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 fulfil a given requester goal.

Following the process described in [15], Glue was configured by

1. importing the ontologies generated following the approach described in section 4.3;
2. importing the classes of shipment Web Services and of goals generated following the approach described in section 5; and
3. manually modeling the required wgMediator that encodes the rules for matching a request for shipmen (modeled as an instance of shipment goal) against the published instances of shipment Web services.

7 Conclusions and future work

This paper presented an overall approach for designing Semantic Web applications by exploiting software engineering techniques. The following results have been shown:

- Ontologies can be imported as models of the data necessary for the cross-enterprise application. They can be extended for addressing the specific needs of the application and registered as shared resources in WSMX.
• WSMO Web Services functional capabilities for delegating sub-processes execution from one enterprise to another are automatically provided for each Web Service modelled in WebML.

• Choreography interfaces can be derived by combining information in the Business Process Model and in the WebML hypertext model. In particular, the local choreography can be derived by taking the point of an external observer of the Web Services that must know the order in which operation can be invoked and the constrains for their successful invocation. In a similar manner we can derive the orchestration interface by translating in WSMO some aspects of the BPMN model of the application, as sketched in Section 5.2.

• WSMO goals can be produced (e.g., goals that triggers the discovery component of WSMX) from gathering data required to perform a given action of the business process, whereas its choreography interface is derived by the explicit representation of workflow primitives within the hypertext.

• mediation services can be modeled as WebML applications and registered in WSMX according to their roles (e.g., a wwMediator).

In summary, in this paper we propose a complete method for the semi-automatic extraction of the components of the WSMO architecture by using existing software engineering abstractions. In our previous work [4] we presented a first sketch of the approach, with special attention to the extraction of the semantic description of services and design of wwMediators (i.e., current Sections 5.1 and 5.4). The current paper extends and details the existing preliminary work, by defining the extraction mechanisms for goals and orchestration, as well as specifying in details and by example the transformations that are needed. Moreover, we provide now a comprehensive view of the methodology and we propose a self-contained design approach for Semantic Web applications and services. We now provide the possibility of importing ontologies, services and mediators descriptions, and we define a set of new hypertext primitives for querying ontological information. Finally, we compared the expressive power of WebML information modeling with respect to well known ontology definition languages.
At the current stage of development, by means of “conventional design” (although supported by an advanced visual design studio), we build software that can run on conventional Web technology and at the same time is ready to become part of a WSMO execution environment (i.e. WSMX). Our future research agenda, which we will schedule in parallel with the wide-spreading and enhancement of WSMO standards, will concentrate upon empowering our design abstractions by incorporating the WSMO concepts as “first-class citizens” in the design process, lifting them up in the artifacts design hierarchy, so as to further improve and simplify the design of native WSMO components.

References