A Framework for Design and Composition of Semantic Web Services

Asunción Gómez-Pérez and Rafael González-Cabero
Departamento de Inteligencia Artificial, Facultad de Informática.
Campus de Montegancedo s/n, Universidad Politécnica de Madrid, 28660 Boadilla del Monte, Madrid. Spain.
asun@fi.upm.es; rgonza@delicias.dia.fi.upm.es

Manuel Lama
Departamento de Electrónica e Computación, Facultad de Física.
Campus Sur s/n, Universidade de Santiago de Compostela, 15782 Santiago de Compostela, A Coruña. Spain.
lama@dec.usc.es

Abstract

Semantic Web Services (SWS) are Web Services (WS) whose description is semantically enhanced with markup languages (e.g., OWL-S). This semantic description will enable external agents and programs to discover, compose and invoke SWSs. However, as a previous step to the specification of SWSs in a language, it must be designed at a conceptual level to guarantee its correctness and avoid inconsistencies among its internal components. In this paper, we present a framework for design and (semi) automatic composition of SWSs at a language-independent and knowledge level. This framework is based on a stack of ontologies that *(1)* describe the different parts of a SWS; and *(2)* contain a set of axioms that are really design rules to be verified by the ontology instances. Based on these ontologies, design and composition of SWSs can be viewed as the correct instantiation of the ontologies themselves. Once these instances have been created they will be exported to SWS languages such as OWL-S.

Introduction

Web Services (WSs) are interfaces that describe a collection of operations that are network-accessible through standardized Web protocols, which are described using a standard functional description language (Kreger 2001; Cubera, Nagy, and Weerawana 2001). To enable WS accessible to programs or external agents, it is necessary an infrastructure (Kreger 2001) that defines *de facto* standard languages for WS publishing (UDDI (Bellwood et al. 2002)); functional description (WSDL (Christensen et al. 2001)); and composition (BPEL4WS (Thatte 2003)).

Moreover, Semantic Web Services (SWSs) are Web services in the semantic Web (Berners-Lee, Hendler, and Lassila 2001), which means that SWSs are described using semantically enriched markup languages (McIlraith, Son, and Zeng 2001). This semantic description will facilitate external agents to understand both functionality and internal structure of SWSs to be able to discover, compose, and invoke SWSs (Hendler 2001). This language could be DAML+OIL (Horrocks and van Harmelen 2001) or OWL (Dean et al. 2003), but it must be combined with WS standard languages to be able to use the current infrastructure of Web services (Sollazo et al. 2002). Following this approach, the OWL-S specification (formerly DAML-S (Ankolenkar et al. 2002)) has been proposed to describe services in a semantic manner, using OWL in combination with WSDL and SOAP (Box et al. 2000).

However, as a previous step to the specification of SWS in a semantic Web-oriented language, the SWS should be designed at a knowledge or conceptual level (Newell 1982) to avoid inconsistencies or errors among the services that constitute the SWS. In this context, SWS *design* consists in specifying he descriptive, functional, and structural features of a service. SWS *composition*, moreover, deals with the combination of different services to obtain a new service. Both design and composition of SWSs are very similar, but composition operates with services already created, and it emphasizes to be a (semi) automatic process; whereas design means that services are (manually) created by users through a graphic interface, although some support for composition would be advisable.

In this paper, we present a framework for design and composition of SWSs that is based on *(1)* a stack of ontologies that describe explicitly the different features of a SWS. Each one of these ontologies has a set of axioms used to check the consistency and correctness of the ontology instances, and, thus, of the service represented by the ontologies; and *(2)* the assumption that a SWS is modeled as a problem-solving method that describes how the service is decomposed into its components, and which is the control of the reasoning process to execute the service. Based on this modeling, manual or (semi) automatic design and composition of SWSs are enabled at a knowledge and language-independent manner.

The paper is structured as follows: in the following section we present a detailed description of the ontologies that represent what a SWS is; then, the proposed framework for SWS design and composition is introduced, and we present an example that illustrates how the framework operates. Finally, we compare the framework with related work and summarize the contributions of the paper.

SWS Description Ontologies for SWS Design

Conceptual architectures of both SWSs and WSs (Sollazo et al. 2002; Fensel and Bussler 2002; Kreger 2001) schematize the service design as the specification of a set of layers that would potentially cover all the service features. These features, which enable programs or external agents to discover, invoke and compose new services, are the following:

- *Access (or communication) features* describe the communication protocol (e.g., SOAP (Box et al. 2000) or HTTP) that is required to invoke the service execution.
- *Descriptive features* detail the e-commerce properties of a SWS such as its geographical location, commerce classification (e.g., UNSPSC) or provider. These features are generally used to define the domain (e.g., *minerals* in UNSPSC) where the service operation is carried out, and they could guide the service discovery by rejecting the services that operates in another different domain (e.g, *medical*).
- *Functional features* specify the SWS capabilities, described in terms of their input/output data, effects and pre/post-conditions of execution. These features enable an external agent to determine, once the service domain has been established, whether the service execution can obtain the requested results. Furthermore, to invoke a service it is necessary to specify the input/output data.
- *Structural features* describe the internal structure of a composite service; that is, which are its structural components (so-called *sub-services*) and how those components are combined among them to execute the service. Typically, agents will use these features for service composition, since they determine whether there are interactions between the sub-services and the other services used to compose a new service.

These features could be considered as the different, but complementary, views of a service. Depending on the operation to be performed by the agent that require the service (invocation, discovery/publishing, or composition), the feature set used to describe the SWS is different. For example, to invoke a given SWS the agent need to specify both its access and functional features (input/output parameters), whereas the agent does not need to know the internal structure of the service (how it is executed). The aim of designing SWSs is making explicit the four features previously mentioned and specially each one of its structural components, guaranteeing the *correctness* of the proposed design, and avoiding the *inconsistencies* among the sub-services that are manual or automatically combined to achieve the requirements of the service. For example, sub-services should have the same commercial classification (e.g., *medical software* in UNSPSC) to be able to operate in the same domain.

Ontology-based SWS Description. Since our aim is to design and compose (semi) automatically SWSs, we will need to perform inferences about the service features to determine whether the proposed design is correct. This means that the service features (and the service itself) should be *explicitly and semantically* described, and, for it, the use of *ontologies* seems to be the most appropriate solution. This approach has been also followed by other authors (Ankolenkar et al. 2002), who use a semantic-enriched markup language to create an ontology (so-called OWL-S) that describes the service features. Our proposal differs from OWL-S in that it claims to develop an ontology set that describes the SWSs at the conceptual (or knowledge) level and being independent of the language used to specify the service. However, once the SWS model has been created, it must be accessible to external agents. Thus, the SWS needs to be translated to a SWS-oriented language such as OWL-S.

Figure 1 shows the stack of ontologies that describes all the features of a SWS (and the service itself) using well-known specifications or *de facto* standards. This will favor the interoperability of the framework with applications or solutions constructed following one of those specifications. The stack is composed of the following ontologies: ***(1)*** an ontology to describe *problem-solving methods* that will be used to represent both the internal structure and functional features of a SWS; ***(2)*** an ontology describing the upper-level concepts that define the features of a *semantic Web service*; ***(3)*** an ontology to define the *knowledge representation* entities used to model a SWS and a domain ontology at the knowledge level; and ***(4)*** an ontology to describe the *data types* to be used in the domain ontology. We will explain each of these ontologies in the following sections.

Problem-Solving Method Description Ontology

Internal structure of both SWSs and WSs has been usually modeled as a (business) *process* (Narayanan and McIlraith 2002; Leymann 2001), in which a set of activities or actions are carried out to execute the process (Schlenoff et al. 2000). In this approach, a process (or equivalently a ser-

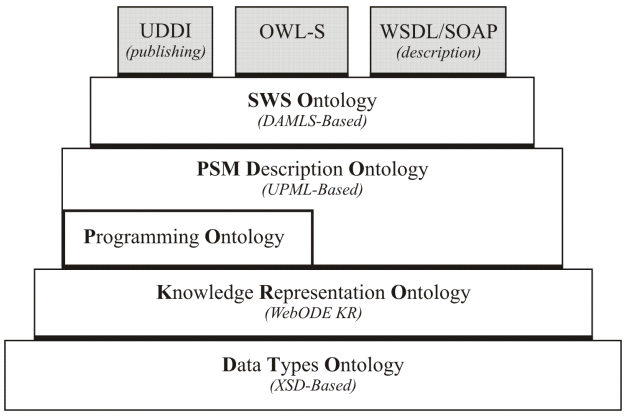

Figure 1: Ontology set identified in the framework to SWS design. These ontologies have been developed based on well-known specifications and *de facto* standards.

vice) is broken down into activities whose interactions can be modeled as workflow patterns (van der Aalst et al. 2003) that basically describe the coordination of those activities in the process execution. Taking this into account, some process-based languages such as BPEL4WS (Thatte 2003) (for WS) and OWL-S (Dean 2004) (for SWS) have been proposed. They describe the internal structure of a service using a predefined set of workflow-like patterns (sequence, choice, parallel split, etc.).

From a point of view of modeling, the main drawback of this approach is the lack of an explicit and declarative *decoupling* between the functional features of a process (what) and the structural description of such process (how). This means that the functional features are directly linked to the parameters used in the internal structure of a process, and, therefore, that process will be specifically designed to carry out a particular operation (e.g., *to book*) in a particular domain (e.g., *flight booking*). With this approach, reuse of processes among domains is difficult and service composition, where software agents (re)use services to obtain a new service, must be programmatically solved. For example, a service that deals with *theatre booking* share some operations with a service in *flight booking* (select seat, check credit card, confirm booking, etc.). Processes that execute such operations should be (quasi) reusable among both services, but that requires an explicit separation between the description of those operations and how they are solved.

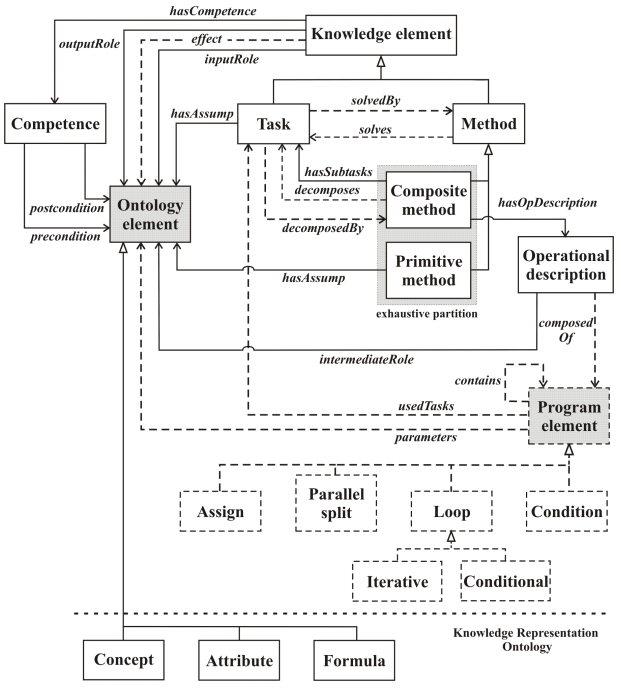

Figure 2: Ontology of Problem-Solving Method based on the UPML specification and a programming ontology that describes the different elements that compose the operational description of a method.

Problem-Solving Method-based approach. To decouple the functional features of a service from its internal specification, we propose to apply Problem-Solving Methods (PSM) (Benjamins and Fensel 1998) for modeling SWSs, because of they claim to be knowledge components that are reusable among different domains and tasks. The Unified Problem-solving Method Language (UPML) (Fensel et al. 2003) is a *de facto* standard that describes the components of a PSM (Figure 2):

- *Task* describes the operation to be solved in the execution of a method that solves such task, specifying the input/output parameters and the pre/post-conditions (competence) required to be applicable. This description is *independent* of the method used for solving the task.
- *Method* details the control of the reasoning process to achieve a task, describing both the decomposition of the general tasks into sub-tasks and the coordination of those sub-tasks to achieve the required result (control flow). The UPML, however, does not define a set of program elements to specify the control flow of a method.
- *Adapter* (Fensel 1997) specifies mappings among the knowledge components of a PSM, adapting a task to a method and refining tasks and methods to generate more specific components. Therefore, the adapters are used to achieve the reusability at the knowledge level, since they bridge the gap between the general description of a PSM and the particular domain where it is applied.

The UPML language was developed in the context of the IBROW project (Benjamins et al. 1999) with the aim of enabling the (semi) automatic reuse and composition of PSMs distributed throughout the Web. This objective seems to be very similar to composition of services, and it can be considered that the IBROW project highlights the close relation between PSMs and SWSs (Benjamins 2003).

PSM Description Ontology. Based on the UPML specification we have created a PSM ontology that enhances the description of the UPML elements by adding new relationships among those elements, concepts, and axioms that describe more exactly the main PSM components. These add-ons, showed as dashed lines in Figure 2, are the following:

- *Explicit relationships* between tasks and methods are defined: a method, which can be primitive *or* composite, *solves* a task; composite methods *decompose* a task into sub-tasks, and they have an operational description that is *composed of* a sequence of program elements which specify the control flow required to solve the general task. Moreover, both tasks and methods have *effects*, which are associated with changes in the environment that are not related with input/output data. These effects should be modeled as concepts of an *effect ontology* to favor the interoperability among services.
- A minimal set of *programming primitives* to describe the operational description of a composite method have been proposed. These primitives will operate with concept instances (*parameters*) and tasks (*usedTasks*), and they can

express instance assignments; conditions applied to instances and tasks; and loops (both conditional and iterative) and parallel execution of other program elements or tasks. Although those primitives are selected from programming languages, a combination of them allow us to derive several basic workflow-like patterns such as sequence or exclusive and multiple choice (Schlenoff et al. 2000; van der Aalst et al. 2003).

- A set of *axioms* that describe interactions among instances of the PSM ontology that should be avoided: there are instantiations of both method and task concepts that are inconsistent with the PSM theory. For example, a method cannot decompose any of the sub-tasks of which it is composed of (subsumption problems); inputs of a (general) task shall be included in the collection of inputs of its sub-tasks, since the composite method that describes the interactions among those sub-tasks must have the same inputs as the general task; and so on. Therefore, these axioms can be used to decide whether a task is correctly decomposed by a composite method or not.

The PSM ontology does not incorporate details related to how a method can be network-accessible, since we assume that communication protocol description is directly associated with the SWS itself.

Semantic Web Service Ontology

As Figure 3 shows, SWS ontology replicates the upper-level nodes (or concepts) of the OWL-S ontology, where a service is described in terms of its descriptive and functional features (*profile*); its access protocol (*grounding*); and its internal structure (*model*). The SWS ontology incorporates all the OWL-S concepts and attributes of the service grounding and profile (except its functional features, which are replaced by task descriptions); whereas the concepts associated with the service model, which is process-based, are completely substituted by method descriptions. Considering this, the SWS ontology defines the following relationships with the PSM ontology (Figure 3):

- *Profile concept* establishes a relationship (*hasTask*) with the concept *task* of the PSM ontology. That is consistent with the notion of both tasks and functional features: they specify an interface to describe the capabilities of a service in terms of its input/output data and the pre/post-conditions of execution of the task (or service). Therefore, a service profile will have *only* a task that defines the functional features of the service.
- *Model concept* defines a relationship (*hasMethod*) with the concept *method* of the PSM ontology. This means that a service will be executed by a method, which, furthermore, solves or decomposes the task associated with the profile of the service itself. Moreover, consistencies in the relationships between tasks, methods, and services are guaranteed: if a service is functionally described by a task, and that task is solved by a method, then the service should be associated to such method. Although a task can be solved by several methods, a service must be associated with only one of those methods. For example, a *theatre booking* service could have a method in which the sub-task *user register* is mandatory *or* a different method without such task.
- *Grounding concept* establishes relationships with an ontology based on the WSDL/SOAP specification (like in OWL-S). This ontology defines relationships with the inputs and outputs of the task related to the service profile.

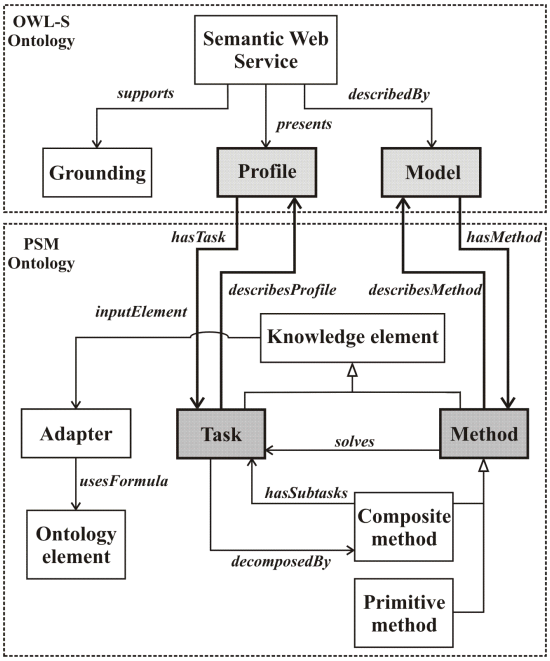

Figure 3: Semantic Web Service ontology that is based on both the OWL-S specification and the PSM description ontology.

According to the SWS ontology, a composite method describes the internal structure of a service (so-called *composite service*), and decomposes a task into *sub-tasks*. Each one of these sub-tasks specifies the functional features of the profile of a service (so-called *sub-service*), which has an internal structure described by a method that solves or decomposes such sub-task. Therefore, unlike process-based approaches (like OWL-S), the description of sub-services in the SWS ontology follows the same structure as for composite services. It means that sub-services are services *per se*: they are described *independently* of any service.

This approach facilitates the SWS design and composition, since analysis of the service features does not depend on the services in which they are integrated. For example, if *check credit card* is a sub-service of *theatre booking*, then to decide whether it is included in a *car booking* service, it is not necessary to analyze the features of *theatre booking*. Moreover, in this approach, service (or PSM) adapters are specified based on description of the features of each service.

Knowledge Representation (KR) Ontology

This ontology describes the primitives of the knowledge representation (KR) model in which the domain ontology, that contains descriptions about the knowledge and data used by the SWS, is represented. This *explicit* representation of the domain ontology is necessary to perform inferences about the input and output data used by the SWS in its operations. For example, in service composition we need to determine the primitive type of its inputs (e.g., a concept) to look for services that use inputs with such primitive.

Taking this into account, we have selected the WebODE knowledge model (Arpírez et al. 2001) as the KR ontology. WebODE is frame-based and it incorporates formulae (first-order logic) to represent ontology axioms. Moreover, WebODE is a workbench for ontological engineering that offers reasoning capabilities (using ontology axioms) and facilities to export/import ontologies into/from semantic Web-oriented languages.

Data Types (DT) Ontology. As Figure 1 shows, the KR ontology is constructed on the top of an ontology that describes the types of the concepts and attributes. This ontology will be based on the XML Schema Datatypes (XSD) (Biron and Malhotra 2001), that is a standard specification formally incorporated to the SWS (and WS) oriented languages such as OWL-S (and WSDL). Using this ontology we will facilitate the translation from the SWS conceptual model into the SWS languages used to service specification.

Framework for SWS Design and Composition

The proposed framework for SWS design and composition is directly based on the stack of ontologies that describe all the features of a SWS. The framework details how to create a SWS with the capabilities required by an external agent or a user (Figure 4):

- *Instance model*. Design and composition of SWS means to instantiate all the ontologies that describe what a service is: the domain ontology used by the service is instantiated in both DT and KR ontology, whereas the service features are instances of both PSM and SWS ontologies. The whole instances constitute a model that specifies the SWS at the knowledge level. This specification can be carried out through a graphic interface that facilitates a user to introduce easily the service features, and generates the instances through wrappers from the graphic representations.
- *Checking model*. Once the instance model has been created, it is necessary to guarantee that the ontology instances do not present inconsistencies among them. For example, inputs/outputs of a composite method must be included in the collection of all the inputs/outputs of the sub-tasks of that method; sub-services should have the same commerce classification; and so on. Therefore, we will need *design rules* to check that there are not inconsistencies or errors when ontology instances are automatically created. These rules are codified as *axioms* of each one of the ontologies that constraint how the instances are related among them. As Figure 4 shows, if an axiom is violated by an instance or set of instances, they shall be replaced with correct instances.

  In this framework we will suppose that the domain ontology has been previously and correctly created through a platform for ontology development (like WebODE), and, thus, instances of both DT and KR ontology do not need to be checked.
- *Translate model*. Although a service is modeled at the knowledge level, it must be specified in a SWS-oriented language to enable programs and external agents to access to its capabilities. SWS ontology instances must be translated to languages such as OWL-S combined with WSDL/SOAP. These translations are usually carried out through wrappers and we can assume that the translation process will be correct.

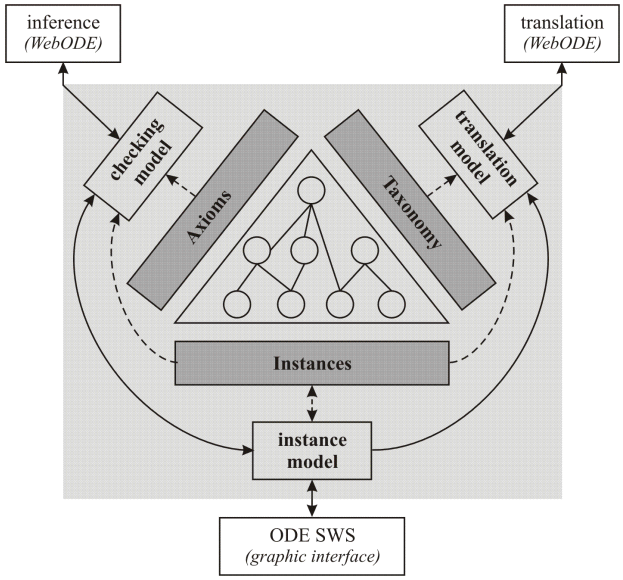

Figure 4: Framework for design and composition of SWSs based on the ontologies that describe the service features.

This framework enables the (semi) automatic composition of SWSs using ***(1)*** PSM refiners and bridges to adapt the PSM ontology instances to the required capabilities of the new service; and ***(2)*** design rules to reject both PSM and SWS ontology instances that present errors or inconsistencies among them. Design rules are used to reduce the service candidates combined to obtain the new service. For example, to check whether the required inputs of the service to be composed are included in the collection of the inputs of the candidate sub-services, it is necessary to adapt, when possible, those sub-service inputs to the ontology used by the composite service. Then, axiom checking

could be performed once this adaptation has been carried out (probably establishing ontology mappings).

Finally, following this framework we have implemented a prototype of an environment for development of SWSs, called *ODE SWS* (Corcho et al. 2003a, 2003b) that has been integrated in WebODE.

Framework Operation Example

Let us suppose that, through a graphic interface of an environment for development of SWSs at a conceptual level (as ODE SWS), a user designs a service, called *theatre booking*, with the aim of booking a theatre ticket for a particular film ($film_F$) in a given city ($city_C$). Figure 5 shows the design of this service using the ODE SWS development environment: the user specifies graphically the input/output interactions among the sub-services that compose the theatre booking service (left side). As we can see, this service is composed of three sub-services: *select theatre*, *select timetable*, and *buy ticket*. Information about credit card is compiled in the user data ($used\ data_U$) and its validation is performed by the service *buy service*.

Once the service has been graphically designed, instances of the all framework ontologies are created. Figure 6 shows the instantiation of the main concepts of both PSM and SWS ontologies for the service *theatre booking*. The method *theatreBookingMethod* decomposes the tasks associated with this service (*theatreBookingTask*) into sub-tasks related to the sub-services.

Following the framework, once the ontology instantiation has been carried out, it is necessary to check whether the service is correctly designed or not. From the input and output data of the services, shown in Figure 6, the violation of an axiom is detected: input $city_C$ of the service *theatreBooking* does not appear as an input of any sub-service. This means that $city_C$ is not used in the execution of the service, and the method that decomposes the service into sub-services (*theatreBookingMethod*) does not provide the capabilities required by the task associated with the service *theatreBooking*. Thus, design of this service shall be revised to include a method that solves correctly the task related to such service.

Finally, once the design has been checked, wrappers perform the translations from the instances of the framework ontologies into the OWL-S specification.

Conclusions and Related Work

There exists some proposals to support (semi) automatic composition of both SWSs and WSs. Sirin, Hendler, and Parsia (2003) describe a service in OWL-S, and they use directly the reasoning capabilities of DAML+OIL to determine whether the features of the services to be composed match the functionality required for the new service or not.

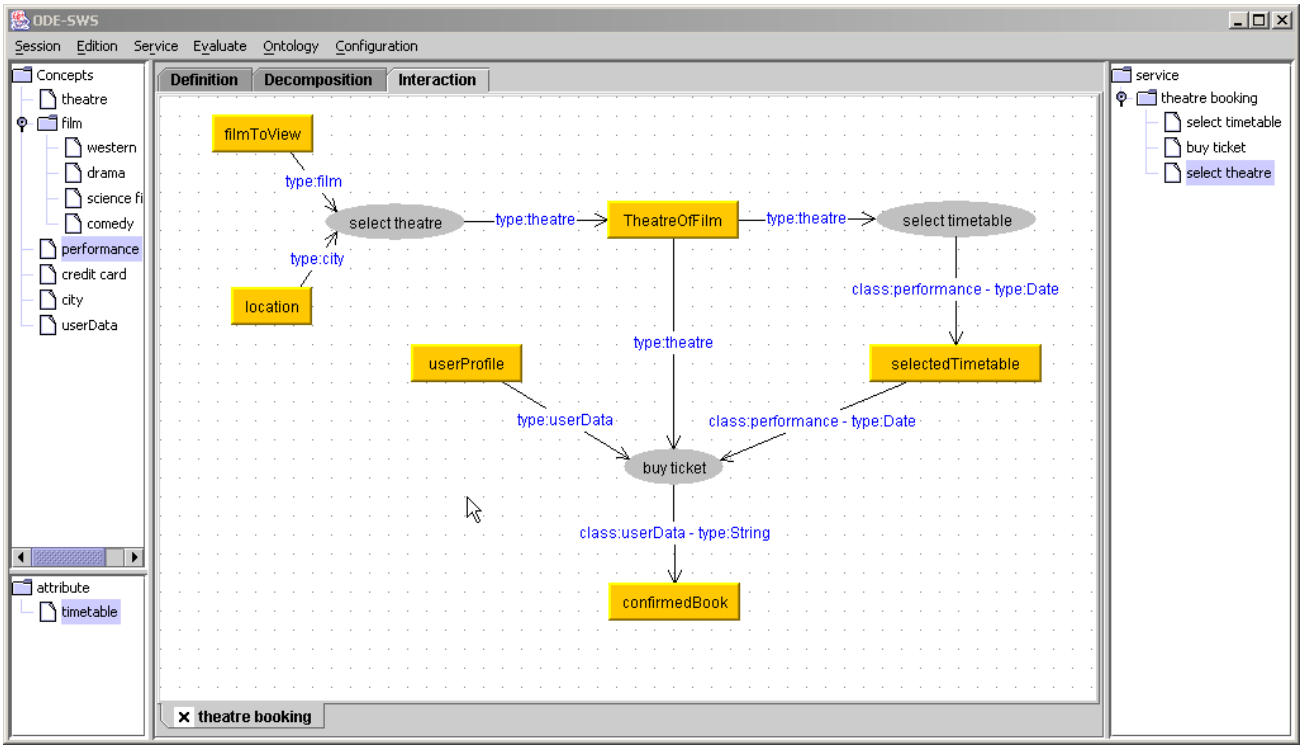

Figure 5: Design of the service *theatre booking* using ODE SWS, where services are denoted as ellipses whereas data (concepts and attributes) are identified as rectangles.

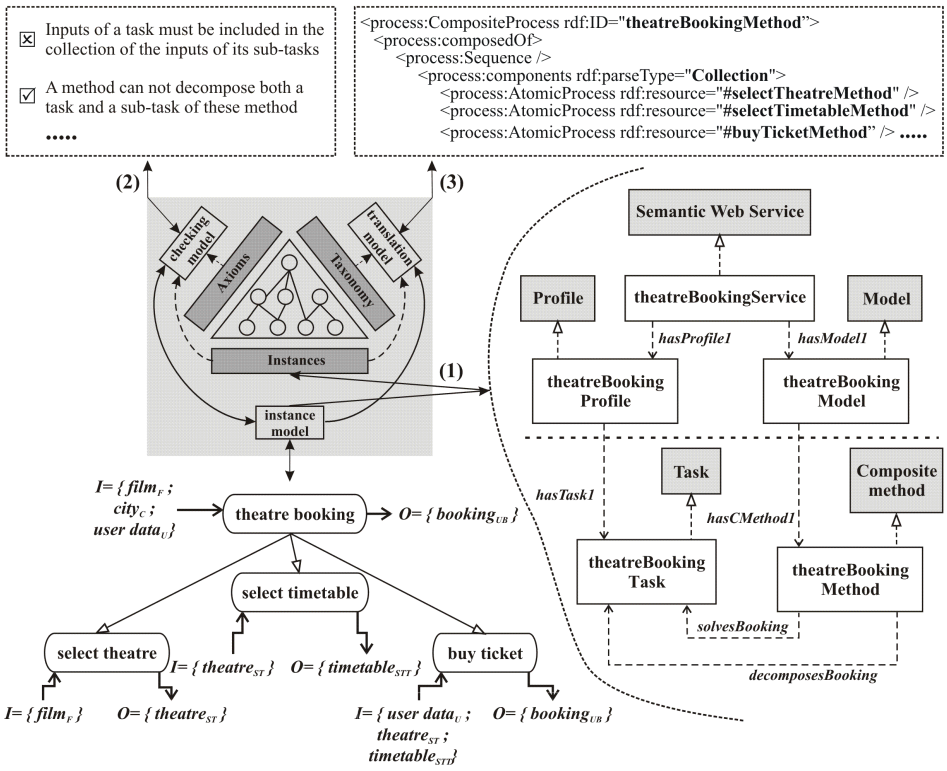

Figure 6: Operation of the framework for design of the service *theatre booking*, where the steps of the framework execution are specified: *(1)* ontology instantiation; *(2)* checking of the ontology axioms; and *(3)* translation from the ontologies into OWL-S.

Narayanan and McIlraith (2002) translate the semantic behind the OWL-S specification to a first-order logic language, obtaining a set of axioms that describes the service features. Based on these axioms, they use the Petri Nets formalism to represent the process-based service model for reasoning about the interaction among the processes that compose the structure of a service. This approach seems similar to our framework, since it uses an ontology that describe what a service is (OWL-S), and, then, it translates this ontology to a formalism for reasoning about the structure and features of the service.

However, the main difference between both proposals and our framework is related to the modeling of the service. We have consider a PSM-based approach where the tasks specify the functional features of a service, the methods describe the interactions among the components of a service, and the adapters describe in a declarative manner how reuse is achieved among services to be composed.

Acknowledgments

Authors would like to thank the Xunta de Galicia and the Esperonto project (IST-2001-34373) for their financial support in carrying out this work.

References

Ankolenkar, A.; Burstein, M.; Hobbs, J.R.; Lassila, O.; Martin, D.L.; McIlraith, S.A.; Narayanan, S.; Paolucci, M.; Payne, T.; Sycara, K.; and Zeng, H. 2002. DAML-S: Web Service Description for the Semantic Web. In *Proceedings of the First International Semantic Web Conference*, 348–363. Sardinia, Italy.

Arpírez, J.C.; Corcho, O.; Fernández-López, M.; and Gómez-Pérez, A. 2001. WebODE – A Scalable Ontological Engineering Workbench. In *Proceedings of the First International Conference on Knowledge Capture*, 1–13. Victoria, Canada: ACM Press.

Bellwood, T.; Clément, L.; Ehnebuske, D.; Hately, A.; Hondo, M.; Husband, Y.L.; Januszewski, K.; Lee, S.; McKee, B.; Munter, J.; and von Riegen, C. 2002. Universal Description Discovery & Integration (UDDI) Specification. *http://uddi.org/pubs/uddi-v3.00-published - 20020719.htm*.

Benjamins, V.R., and Fensel, D. eds. 1998. Special Issue on Problem-Solving Methods. *International Journal of Human-Computer Studies (IJHCS)*, 49(4): 305–313.

Benjamins, V.R.; Wielinga, B.; Wielemaker, J.; and Fensel, D. 1999. Brokering Problem-Solving Knowledge at the Internet. In *Proceedings of the European Knowledge Acquisition Workshop (EKAW-99)*: Springer-Verlag.

Benjamins, V.R. 2003. Web Services Solve Problems, and Problem-Solving Methods Provide Services. *IEEE Intelligent Systems*, 18(1):76–77.

Berners-Lee, T.; Hendler, J.; and Lassila, O. 2001. The Semantic Web. *Scientific American*, 284(5):34–43.

Biron, P.V., and Malhotra, A. 2001. XML Schema Part 2: Datatypes. *http://www.w3.org/TR/xmlschema-2*.

Box, D.; Ehnebuske, D.; Kakivaya, G.; Layman, A.; Mendelsohn, N.; Nielsen, H.F.; Thatte, S.; and Winer, D. 2000. Simple Object Access Protocol (SOAP) Version 1.1. *http://www.w3.org/TR/2000/NOTE-SOAP-200000508*.

Christensen, E.; Curbera, F.; Meredith, G.; and Weerawarana, S. 2001. Web Service Description Language (WSDL) 1.1. *http://www.w3c.org/TR/2001/NOTE-wsdl-20010315*.

Corcho, O.; Fernández-López, M.; Gómez-Pérez, A.; and Lama, M. 2003a. An Environment for Development of Semantic Web Services. In *Proceedings of the IJCAI-2003 Workshop on Ontologies and Distributed Systems*, 13–20. Acapulco, México. http://CEUR-ORG.com/Vol-71/

Corcho O.; Fernández-López, M.; Gómez-Pérez, A.; and Lama, M. 2003b. ODE-SWS: A Semantic Web Service Development Environment. In *Proceedings of the VLDB-2003 Workshop on Semantic Web and Databases*, 203–216. Berlin, Germany.

Curbera, F.; Nagy, W.A.; and Weerawana, S. 2001. Web Service: Why and How?. In *Proceedings of the OOPSLA-2001 Workshop on Object-Oriented Services*. Tampa, Florida.

Dean, M.; and Schreiber, G. eds. 2003. OWL Web Ontology Language Reference. W3C Candidate Recommendation. *http://www.w3c.org/TR/owl-ref/*

Dean, M. ed. 2004. OWL-S: Semantic Markup for Web Services. *http://www.daml.org/services/owl-s/1.0/owl-s.pdf*

Fensel, D. 1997. The Tower-of-Adapter Method for Developing and Reusing Problem-Solving Methods. In *Proceedings of the Seventh Knowledge, Modeling and Management Workshop*, 97–112: Springer-Verlag.

Fensel, D. and Bussler, C. 2002. The Web Service Modeling Framework WSMF. In *Proceedings of the NFS-EU Workshop on Database Information System Research for Semantic Web and Enterprises*, 15–20. Georgia, USA.

Fensel, D.; Motta, E.; van Harmelen, F.; Benjamins, V.R.; Crubezy, M.; Decker, S.; Gaspari, M.; Groenboom, R.; Grosso, W.; Musen, M.A.; Plaza, E.; Schreiber, G.; Studer, R.; and Wielinga, B. 2003. The Unified Problem-Solving Method Development Language UPML. *Knowledge and Information Systems (KAIS): An International Journal*. Forthcoming.

Hendler, J. 2001. Agents and the Semantic Web. *IEEE Intelligent Systems*, 16(2):30–37.

Horrocks, I., and van Harmelen, F. eds. 2001. Reference Description of the DAML+OIL Ontology Markup Language, Technical Report, *http://www.daml.org/2001/03/reference.html*.

Kreger, H. 2001. Web Services Conceptual Architecture. *http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf*.

Leymann, F. 2001. Web Service Flow Language (Version 1.1). *http://www.ibm.com/software/solutions/webservices/pdf/WSDL.pdf*.

McIlraith, S.; Son, T.C.; and Zeng, H. 2001. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53.

Narayanan, S. and McIlraith, S. 2002. Simulation, Verification and Automated Composition of Web Services. In *Proceedings of the Eleventh International World Wide Web Conference (WWW-2002)*, 77-88. Hawaii, USA.

Newell, A. 1982. The Knowledge Level. *Artificial Intelligence*, 18(1):87–127.

Schlenoff, C.; Gruninger, M.; Tissot, F.; Valois, J.; Lubell, J.; and Lee, J. 2000. The Process Specification Language (PSL):Overview and Version 1.0 Specification. Technical Report NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD.

Sirin, E.; Hendler, J.; and Parsia, B. 2003. Semi Automatic Composition of Web Services using Semantic Descriptions. In *Proceedings of the ICEIS-2003 Workshop on Web Services: Modeling, Architecture and Infrastructure*.

Sollazo, T.; Handshuch, S.; Staab, S.; and Frank, M. 2002. Semantic Web Service Architecture – Evolving Web Service Standards toward the Semantic Web. In *Proceedings of the Fifteenth International FLAIRS Conference*. Pensacola, Florida.

Thatte, S. eds. 2003. Business Process Execution Language for Web Services (Version 1.1). *http://www.ibm.com/developerworks/library/ws-bpel*.

van der Aalst, W.P.; ter Hofstede, A.H.; Kiepuszewski, B.; and Barros, A.P. 2003. Workflow patterns. *Distributed and Parallel Databases*, 14(2):5–51.