ODE-SWS: A Semantic Web Service Development Environment

Óscar Corcho[1], Asunción Gómez-Pérez[1], and Mariano Fernández-López[1]
Manuel Lama[2]

[1] Departamento de Inteligencia Artificial. Facultad de Informática.
Campus de Montegancedo, s/n. Universidad Politécnica de Madrid.
28660 Boadilla del Monte, Madrid, Spain.
{ocorcho,mfernandez,asun}@fi.upm.es
[2] Departmento de Electrónica y Computación. Facultad de Física.
Campus Sur, s/n. Universidad de Santiago de Compostela.
15782 Santiago de Compostela, A Coruna, Spain.
lama@dec.usc.es, davidal@usc.es

Abstract. Web Services (WS) are software modules that perform operations that are network-accessible through XML messaging. Web Services in the Semantic Web, that is, Semantic Web Services (SWS), should describe semantically their structure and capabilities to enable its automatic discovery, invocation and composition. In this work we present a development environment to design SWS in a language-independent manner. This environment is based on a framework that defines an ontology set to characterize how a SWS should be specified. The core ontology of this framework describes the SWS problem-solving behaviour and enables the SWS design at a conceptual level. Considering this framework, the SWS development environment is composed of (1) a graphical interface, in which the conceptual design of SWSs is performed, and (2) a tool set, which instantiates the framework ontologies according to the graphical model created by the user, verifies the completeness and consistency of the SWS through instance evaluation, and translates the SWS conceptual model description into SWS (and WS) languages, such as DAML-S, WSDL or UDDI. This tool set is integrated in the WebODE ontology engineering workbench in order to take advantage of its reasoning and ontology translation capabilities.

1 Introduction

Web Services (WSs) are software modules that describe a collection of operations that can be network-accessible through standardized XML messaging [1]. WSs are distributed all over the Internet, and in order to enable this accessibility and interactions between WSs, it becomes necessary an infrastructure offering mechanisms to support the WS discovery and direct invocation from other services or agents. Nowadays, there are a number of proposals (usually ecommerce-oriented) that claim to enable partial or totally this required infrastructure, such

as ebXML [2], E-Speak [3], or BPEL4WS [4]. However, the approach that has emerged as a de facto standard, due to its extended use and relative simplicity, is the Web Service Conceptual Architecture [1]. This framework is composed of a set of layers that, basically, enable: (1) *WS publication*, where the UDDI specification [5] is used to define the WS capabilities and characterize its service provider; (2) *WS description*, which use the WSDL language [6] to specify how the service can be invoked (input-output messages), and SOAP [7] as the communication protocol for accessing web services; and (3) *WS composition*, which specifies how a complex service can be created combining simple ones. The language used to describe this composition is WSFL [8].

In this context, the Semantic Web [9] has risen as a Web evolution where the information is semantically expressed in a markup language (such as DAML+OIL [10]) and, thus, both agents and services could access directly to it. This approach considers that the Web Services in the Semantic Web, so called Semantic Web Services (SWSs), should specify their capabilities and properties in a semantic markup language [11], [10]. This markup would enable other services to reason about the SWS, and, as a result, decide whether it matches their requirements. Taking this into account, two frameworks, SWSA [13] and WSFM [14], have been proposed to describe a semantic Web infrastructure for enabling the automatic SWS discovery, invocation and composition. Both frameworks use the DAML-S specification [15], which is a DAML+OIL ontology for SWS specification, and emphasize the SWS integration with de facto standard WS, in order to take advantage of its current infrastructure.

On the other hand, Problem-Solving Methods (PSMs) describe explicitly how a task can be performed [16]. PSMs are intended to be *reusable* components applicable to similar tasks but in different domains. A PSM description specifies the tasks in which the PSM is decomposed (methods-tasks tree); the input-output interactions between the tasks; the flow control that describes the task execution; the conditions in which a PSM can be applied to a domain or task; and, finally, the ontology used by the PSM (method ontology). The UPML specification [17] provides containers in which these PSM views can be described, and, also, it incorporates elements that enable the PSM reuse. UPML has been developed in the context of the IBROW project [18] with the aim of enabling the semi-automatic reuse of PSMs. This objective could be interpreted as a composition of PSMs.

In this work we provide a SWS development environment, called *ODE-SWS*, which would allow the user to design SWSs on the basis of PSM modelling, enabling its description and composition at a conceptual level. This environment also performs verification about the consistency and completeness of the design created by the user. Once the design is verified, the user will select the specific languages in which the SWS will be specified. Thus, the SWS development process supported by this environment does not depend on a specific SWS specification language. On the other hand, ODE-SWS is integrated in WebODE [19], an ontology development workbench that offers an infrastructure in which ontology services (such as merging, evaluating and reasoning with ontologies) can

be reused by other services or applications. In this way, ODE-SWS development has been facilitated with its integration in WebODE.

The structure of the paper is as follows. In section 2, a PSM-based framework that enables the SWSs (and WSs) development is presented. In section 3, the software architecture of the environment that supports this framework and how it has been integrated in WebODE is described. In section 4, the current capabilities of its graphical interface are explained. Finally, in section 5, the main contributions of the work are summarized and other proposals to develop SWS are discussed.

2 Framework for SWS Development

Relationships between SWSs and PSMs have been emphasized by several authors [20], [14]. When both SWSs and PSMs are applied, they execute an operation (or equivalently a method) to perform a task in a domain. As a result of this execution, either new domain information is obtained or an effect is provoked in the real world. Taking this similarity into account, it seems to be reasonable to use the PSM paradigm to define the SWS features related to their internal structure (SWS description and composition). Thus, we propose a framework in which the SWS development is *based on* PSM descriptions, which could be extended with knowledge about ecommerce features (to facilitate SWS discovery) and communication protocols (to provide network-accessibility).

On the other hand, the design of the framework has been guided by a set of requirements that establish the conditions to define an open and extensible framework to develop SWSs. These requirements are as follows:

1. *SWS conceptual modeling.* SWS development must be carried out at a conceptual level and, therefore, characterization and description of the SWS capabilities and internal structure (for composition and description) cannot depend on specific languages that could limit the expressiveness of the SWS model. Our aim is to allow the users to develop SWSs in a language-independent manner; the environment that supports the framework will be responsible to translate the SWS design into the required SWS languages.
2. *Integration of SWS with Web Service standards.* SWS specifications should be integrated with Web service de facto standards (both frameworks and languages) to be able to use the current infrastructure that supports these standards [13], [20]. This requirement is compatible with the need of enabling a SWS conceptual design, because this integration is carried out once the SWS conceptual model has been created.
3. *Modular design.* The framework must be composed of a set of independent, but related, modules, which contain knowledge about different views of the SWS development process. This criterion guarantees the extensibility of the framework, because we could introduce new modules without have to modify the others.

2.1 Layered-Based Framework

To cover these requirements we propose a framework with a layered design, whose layers are identified following a generality criterion, from the data types (lower layer) to the specific languages in which SWSs will be expressed (higher layer). Each layer is defined by an *ontology* that describes its elements on the basis of well-known standards. These ontologies (or layers) are the following (see figure 1):

- *Data Types (DT) Ontology.* It contains the data types associated to the concept attributes of the domain ontology. The data types included in the DT ontology are the same as the ones defined in the XML Schema Data Types specification [21].
- *Knowledge Representation (KR) Ontology.* It describes the representation primitives used to specify the domain ontology managed by SWSs in its operations. That is, the components of the domain ontology will be KR instances. KR ontology is needed because higher framework ontologies (PSM and SWS) could need to reason about the domain ontology. For example, preconditions of a method could impose that the input-output data should be attributes. Usually, the KR ontology is associated to the knowledge model of the tool used to develop the domain ontology.
- *PSM Description Ontology.* This ontology describes the elements that compose a PSM, which, as we have previously discussed, can be used to generate SWS descriptions. The PSM ontology is constructed following the UPML specification [17], that has been extended with (1) a *programming structure*

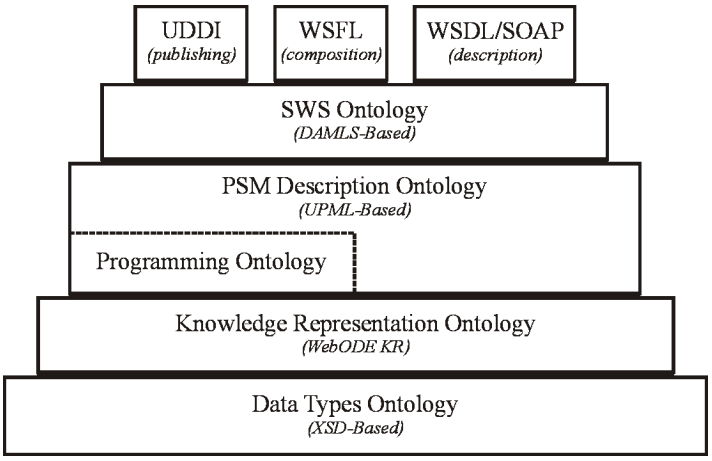


Fig. 1. Framework for SWS development. This framework is composed of a set of design layers, each one defined by an ontology that is based on well-known specifications of the components that it describes

ontology, which describes the primitives used to specify the PSM flow control (such as conditional and parallel loops, conditional statements, etc.); (2) *inferences*, which are new PSM elements defined as in the CommonKADS knowledge model [22], that is, as building blocks for reasoning processes; and (3) relations between PSM elements to explicitly declare whether an element may be executed *independently* of the others or not and whether they can be invoked by an external agent (or service). In figure 2 an excerpt of the PSM ontology is showed. On the other hand, the PSM ontology contains a number of axioms that constrain how PSM element instances are created. This guarantees the consistency of the PSM model. For example, there exists an axiom establishing that the input method must be covered by the inputs associated to the tasks that compose the method.

– *SWS Ontology.* This ontology is constructed on the basis of the PSM description ontology, which is extended with both knowledge related to ecommerce interactions, which enable the publication and advertisement of services, and communication protocols. These extensions are performed using the DAML-S specification as reference [15], because it describes containers to include these types of knowledge.
– *Standard language ontologies for Web Services.* They describe the elements associated to the de facto Web standard languages for service publication (UDDI), description (WSDL/SOAP), and composition (WSFL). These on-

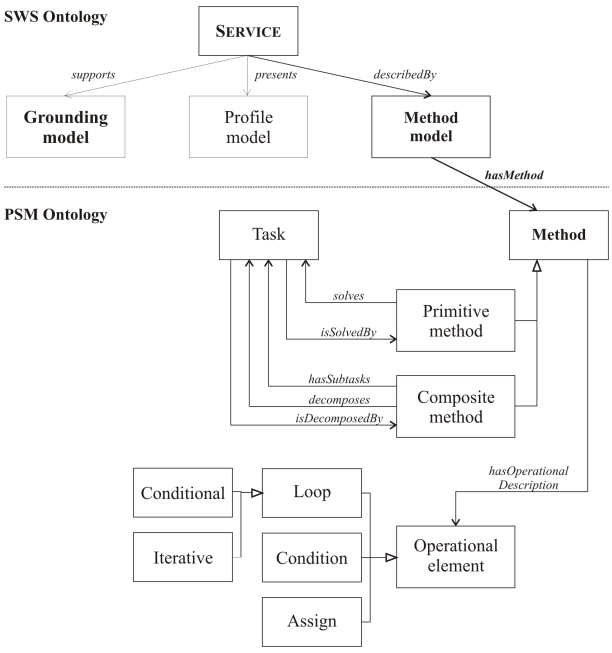


Fig. 2. Excerpt of the PSM ontology and how it is related with the SWS ontology

tologies complete the SWS specification, because they facilitate its integration in the current infrastructure of the Web.

This framework verifies the design requirements: conceptual modeling of SWSs is performed in the PSM layer, which is not constructed following a specific language, but modelled at knowledge level [23]; integration with Web service standards is explicitly enabled in the higher framework layer, which, if required, could be easily extended to include new standards; and, finally, modular design is achieved through the layered approach itself.

3 SWS Development Environment

To provide support for the framework, we have designed a SWS development environment, in which users can design the conceptual model of SWS through a graphical interface. Once finished, the model must be checked to guarantee its consistency and correctness. Then the SWS model can be converted into a DAML+OIL specification (such as DAML+OIL), which will be complemented with Web service standard languages. The software architecture of this environment, which is called *ODE-SWS*, has been designed following the framework requirements, that is, to develop an open and extensible environment, which, if required, could be easily modified to support new SWS (and WS) specification languages or frameworks.

3.1 Software Architecture

According to the proposed framework, the SWS development could be viewed as the process of instancing an ontology set that contains the knowledge needed to generate the SWS specifications. ODE-SWS software architecture is based on this consideration and it is composed of: a *graphical interface*, which allows the users to develop SWSs at a conceptual level (section 4); and a set of services (or tools), *called ODE-SWS services*, which process the SWS graphical descriptions (previously created by the users) to generate the instances of the framework ontology at which each service is *connected*. That is, each framework layer is associated to a ODE-SWS service which operates with the knowledge contained into the ontology that describes that layer.

Figure 3 shows the general structure of a ODE-SWS service. Usually, a service is activated by the ODE-SWS graphical interface to (1) verify the consistency and completeness of the SWS conceptual model; or (2) translate this model from its graphical description into a specific language. In both cases, however, it is necessary to generate an instance set of the ontology connected to the service. In the first case, the SWS conceptual model is verified applying the ontology axioms to the instance set; the ODE-SWS service contains a module that will activate the reasoning with the ontology axioms. In the second case, it is also necessary to check the consistency and completeness of the SWS model to avoid errors in the specification of the SWS. Once this verification has been carried out, an

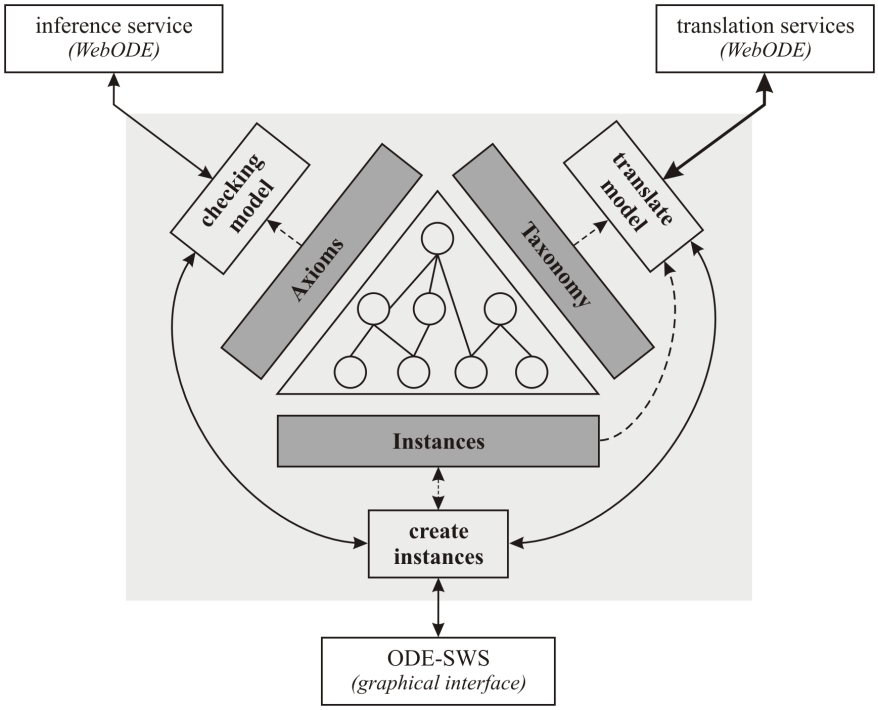


Fig. 3. General structure of a ODE-SWS service, where the ontology with which the service operates must be *one* of the ontologies identified in the SWS development framework

ODE-SWS service module will export the ontology to the language selected by the user.

On the other hand, ODE-SWS is completely integrated in WebODE [19], which is a workbench for ontology development that provides additional services for exporting ontologies to different languages (such as DAML+OIL, RDF, etc.), merging and evaluating ontologies, and reasoning with ontologies using their axioms. The WebODE software architecture is scalable and easily extensible, and it is divided in three layers (figure 4). In the first layer, the *ontology development services* are included. They verify the ontology consistency, enable the access to the ontologies stored in a relational database, reason with ontology axioms, and export/import the ontologies to/from different languages.

In the second layer the *middleware services* are located. They use the ontology development services in their operations and provide additional capabilities to WebODE, such as merging or evaluation. The ODE-SWS services are integrated in this layer. Thus, they directly use: (1) the *WebODE inference* service to evaluate the ontologies by means of their axioms; (2) the *WebODE ontology access* service to manage the framework ontologies (which are stored in We-

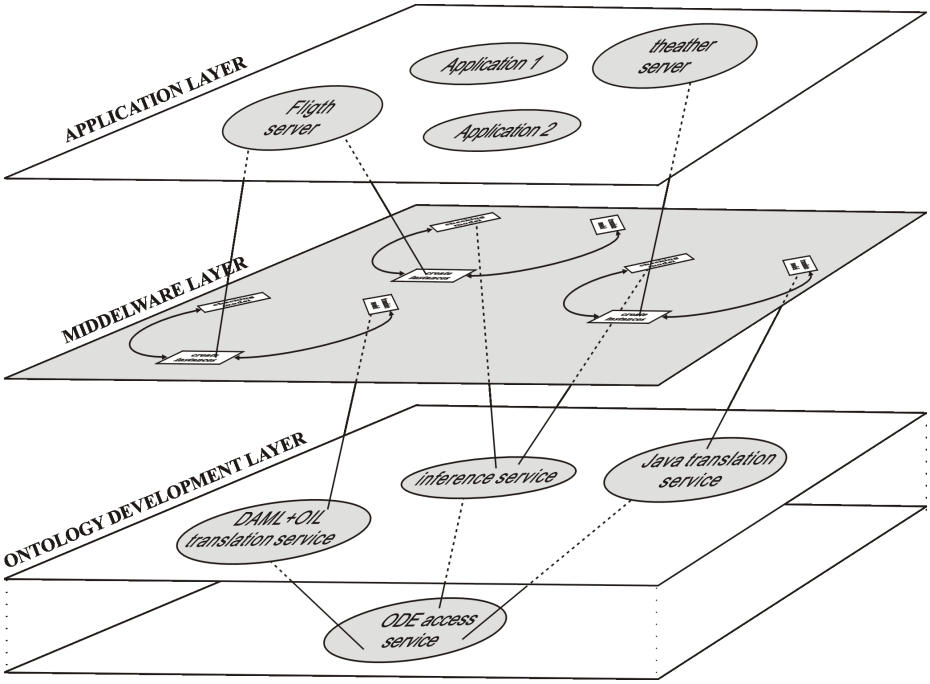


Fig. 4. Integration of ODE-SWS services in the WebODE architecture

bODE); and (3) the *export* services to translate the SWS model into a specific SWS language. In this layer the ODE-SWS graphical interface is also included and uses the ODE-SWS services and the WebODE ontology access service.

Finally, in the third layer the *applications* that mainly use the middleware services in their operations are constructed. For example, a *theatre server* application that offers SWSs to allow the users to book tickets for a particular film projected in the theatre, will probably use ODE-SWS because it provides capabilities needed in the application definition. Therefore, WebODE platform could be considered as an application development environment, in which new services can be easily integrated and reused by other applications by means of the *infrastructure* provided by the platform.

3.2 ODE-SWS services

ODE-SWS services are directly invoked from the ODE-SWS graphical interface when the users, once they create the SWS conceptual model in a graphical manner, require to export that model to well-known SWS languages or when the graphical interface itself needs to verify whether an operation carried out by the user has generated a SWS inconsistent model or not. Taking this into account, we identify the following ODE-SWS services (figure 5):

– *KR service*. This service gets as input the ontology used in SWS operation (usually the domain ontology) and establishes the instances associated to the KR and Data Types ontologies. The domain ontology can be available in WebODE or could be imported from an ontology language into the WebODE specification. In both cases, this service will invoke the ODE service to access the domain ontology components stored in a database.
– *PSM service*. It uses the graphical descriptions of the SWS model to generate an instance set that describes completely the PSM model (internal structure and flow control). Once the instance set is created, this service must invoke the WebODE inference service [24] to verify the consistency and completeness of the PSM model. In this verification, the axioms that constrain how the PSM elements can be combined with each other are used. For example, if we would define a general service that is decomposed in two sub-services,

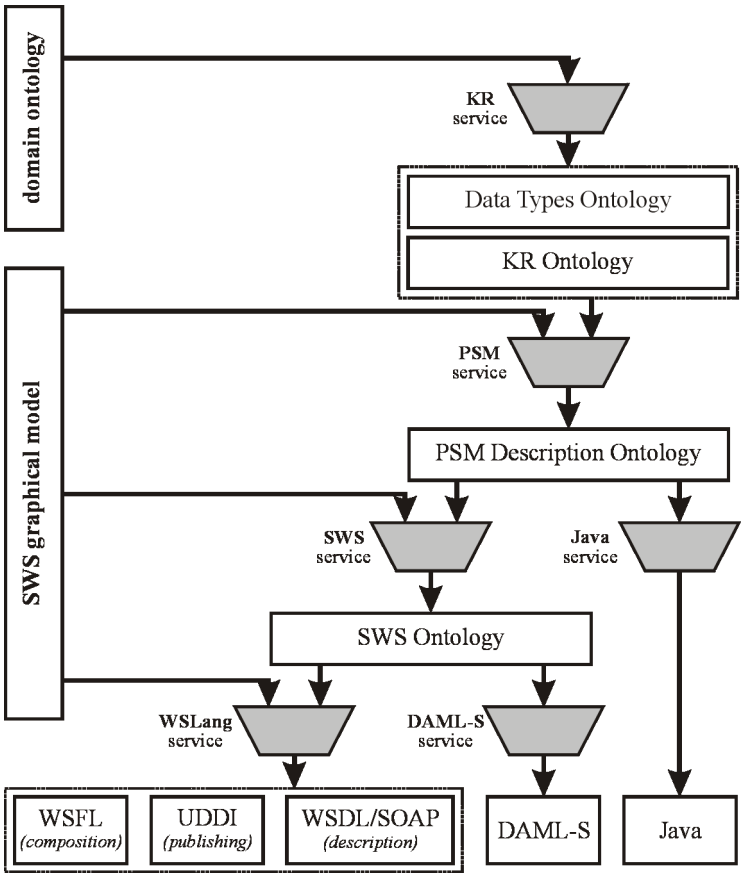


Fig. 5. Input-output relations between ODE-SWS services in order to generate the SWS model and its specification in a SWS language

it would be necessary to verify that the inputs of these sub-services would be of the same (or subsumed) type as the general service inputs. In order to perform this verification, the PSM service must operate with an explicit description of the representation primitives in which the domain ontology will be instanced.
- *SWS service*. Instances created by this service will enhance the knowledge included in the PSM model by adding the information related to ecommerce interactions. This information will be directly obtained from the ODE-SWS graphical interface.

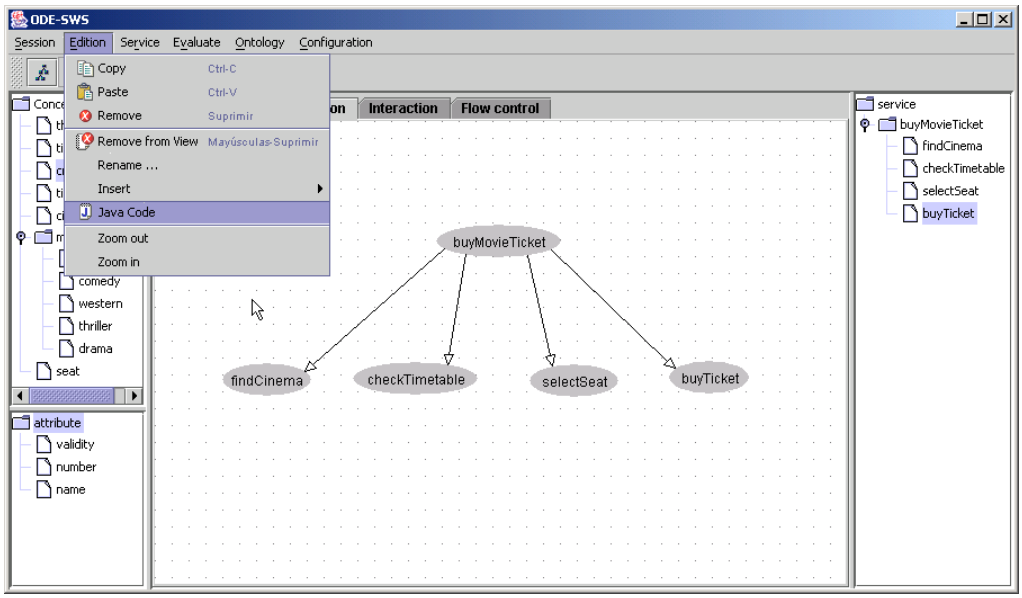
These three services constitute the *ODE-SWS core*, because they support the generation of the SWS conceptual model (from the SWS graphical descriptions) and their operation does not depend on the specific languages in which the SWS will be described. Therefore, these services will be modified only if their associated framework layers are also changed.

- *WSLang service*. It gets as inputs the SWS ontology instances and generates an instance set from which the SWS model is specified in UDDI, WSDL/SOAP and WSFL de facto standard languages.
- *DAML-S service*. It provides the DAML-S specification of the SWS having as inputs the instances of the SWS ontology. Nevertheless, this operation is not straightforward because in the DAML-S ontology a service is modeled as a *process*, whereas in our framework a service is considered to be a *specialization of a PSM* (or method). Once this operation is performed, this service must invoke the WebODE service, which exports an ontology into the DAML+OIL language.
- *Java service*. Using the PSM ontology instances as inputs, this service will generate the skeleton of the programming code (Java beans) needed to execute the SWS and to perform its operations. Once this code has been generated by the service, the user must fill in the methods responsible of carrying out the operations modelled in the PSM.

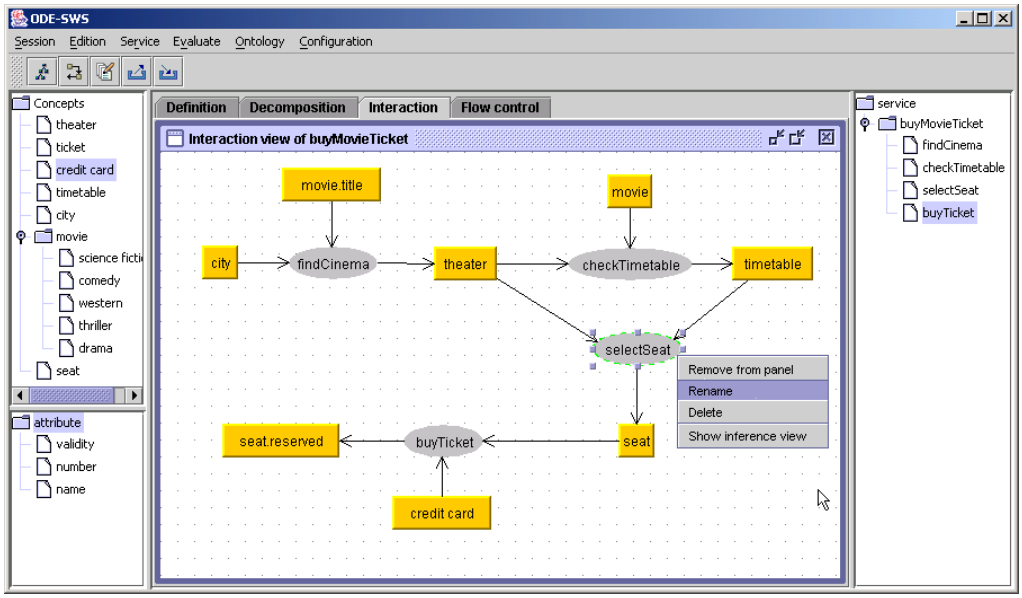
These three services represent ODE-SWS *additional services*, because they have been specifically included in the environment to support the translation from the SWS model into the languages in which the SWS will be expressed. This means that these services would be changed (or substituted) if it was required to use other languages or if the core services were also modified.

4 Graphical Interface

ODE-SWS graphical interface is based on the assumption that the design and development of a service should be performed from different, but complementary, points of view (such as in PSM modelling). These different views help the user to understand the internal structure of a service and the interactions between its components (sub-services); that is, these views facilitate the SWS description and composition. Taking this into account, the graphical interface contains the following views (see figure 6):



(a)



(b)

Fig. 6. ODE-SWS graphical interface

- *Definition view.* In this view the user defines a service by specifying its name (mandatory) and, optionally, by introducing the information needed to enable service discovery and advertisement, such as a description of the provider that offers the service, the types of business for which the service is oriented (industry classifications), etc.
- *Decomposition view.* This view allows the user to define (and also create) the services (sub-services) that would be executed when a (composite) service is activated. That is, a service hierarchy can be specified. This view, therefore, enables *service composition* by creating a hierarchy in which the sub-services of a composite service are activated if it verifies their execution conditions. Figure 6.(a) shows how the service *BuyMovieTicket* is decomposed in its sub-services. On the other hand, this view can be used to detect possible inconsistencies between different views. For example, in the flow control of a service cannot appear services that do not belong to its hierarchy.
- *Interaction view.* In this view the input-output interactions between the sub-services of a composite service are specified. This operation requires that the domain ontology would be previously loaded from WebODE database to the graphical interface. Figure 6 shows the main window of the ODE-SWS, where the specification of the interactions between the sub-services of *buyMovieTicket* composite service can be seen. All these services have been created in the decomposition view (or in the definition view), which will generate the service tree shown in the right side of figure 6.(b).
- *Flow control view.* In this view the user specifies the flow control of a service, where its sub-services are combined with programming structures to obtain a description of the service execution. This view, which is not implemented yet, will be used to model the service composition by means of several diagrams that describe the different compositions of services. On the other hand, this view and the decomposition view could be used to export to languages (as WSFL) that specify the service composition.

The graphical interface guarantees the consistency and completeness of the models that have been created in each one of its views. For example, if the user specifies that a service is composed of three sub-services (decomposition view), the graphical interface will invoke the PSM service to assure that the interaction view contains exactly those three services (as in the example shown in figure 6).

5 Conclusions

ODE-SWS enables the users to develop SWSs following a PSM-oriented design, which is based on a language-independent framework for SWS development. Furthermore, ODE-SWS will assure the consistency and completeness of the SWS designs. Once the SWS design correctness is verified, the user can select the languages in which the SWS will be described. Thus, in ODE-SWS the user does not need to know specific details about the languages used to specify the SWSs.

On the other hand, the ODE-SWS integration in WebODE has simplified its software architecture and implementation, because (1) it uses directly the WebODE services, which offer support for ODE-SWS operations; and (2) it uses the infrastructure itself that WebODE provides for including software modules as services, which could be easily accessed form the graphical interface. Thus, the integration in WebODE favors the ODE SWS modularity, which is a key requirement to adapt the environment to new standard languages or frameworks.

Finally, there exists some development environments which offer capabilities for SWS composition and consistency verification [26], [25]. Both environments are based on the DAML-S ontology and they use the reasoning capabilities associated to the DAML+OIL language to verify the SWS model consistency. These environments are language-dependent and the SWS conceptual modelling depends on the DAML+OIL mark-up, which, therefore, highly difficult its translation to others languages or frameworks. On the other hand, none of these two environments are supported by an infrastructure that could offer other useful capabilities such as evaluation or reasoning about ontologies.

References

1. H. Kreger: *Web Services Conceptual Architecture (WSCA 1.0).* http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf, May 2001.
2. D Webber and A. Dutton: Understanding ebXML, UDDI and XML/edi. http://www.xmlglobal.com/downloads/ebXML_understanding.pdf, October 2000.
3. S. Graupner, W. Kim, D. Lenkov, and A. Sahai: *E-Speak – An Enabling Infrastructure for Web-based E-Services.* Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet, L'Aquila, Italy, July August 2000.
4. F. Curbera, Y. Golan, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana: *Business Process Execution Language for Web Services. Version 1.* http://www.ibm.com/developerworks/library/ws-bpel, July 2002.
5. T. Bellwood, L. Clément, D. Ehnebuske, A. Hately, M. Hondo, Y.L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, and C. von Riegen: *UDDI Version 3.0. Published Specification.* http://uddi.org/pubs/uddi-v3.00-published-20020719.htm, July 2002.
6. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana: *Web Services Description Language (WSDL) 1.1.* http://www.w3c.org/TR/2001/ NOTE-wsdl-20010315, March 2001.
7. D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte, and D. Winer: *Simple Object Access Protocol (SOAP) 1.1.* http://www.w3.org/TR/2000/NOTE-SOAP-20000508, May 2000.
8. F. Leymann: *Web Service Flow Language (WSFL 1.0).* http://www.ibm.com/software/solutions/webservices/pdf/WSDL.pdf, May 2001.
9. T. Berners-Lee, J. Hendler, and O. Lassila: *The Semantic Web.* Scientific American, 284(5):34-43, 2001.
10. J. Hendler and D. McGuinness: *The DARPA Agent Markup Language.* IEEE Intelligent Systems, 15(6):72-73, 2000.

11. S.A. McIlraith, T.C. Son, and H. Zeng: *Semantic Web Services*. IEEE Intelligent Systems, 16(2):46-53, 2001.
12. J. Hendler: *Agents and the Semantic Web*. IEEE Intelligent Systems, 16(2):30-37, 2001.
13. T. Sollazzo, S. Handshuch, S. Staab, and M. Frank: *Semantic Web Service Architecture – Evolving Web Service Standards toward the Semantic Web*. Proceedings of the Fifteenth International FLAIRS Conference, Pensacola, Florida, May 2002.
14. D. Fensel and C. Bussler: *The Web Service Modeling Framework WSMF*. Proceedings of the NSF-EU Workshop on Database and Information Systems Research for Semantic Web and Enterprises, pages 15-20, Georgia, USA, April 2002.
15. A. Ankolenkar, M. Burstein, J.R. Hobbs, O. Lassila, D.L. Martin, S.A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng: *DAML-S: Semantic Markup for Web Services*. Proceedings ot the First Semantic Web Working Symposium, pages 411-430, July August 2001.
16. V.R. Benjamins and D. Fensel: *Special Issue on Problem-Solving Methods*. International Journal of Human-Computer Studies (IJHCS), 49(4):305-313, 1998.
17. D. Fensel, E. Motta, F. van Harmelen, V.R. Benjamins, M. Crubezy, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, M. Musen, E. Plaza, G. Schreiber, R. Studer, and B. Wielinga: *The Unified Problem-Solving Method Development Language UPML*. Knowledge and Information Systems (KAIS): An International Journal, 2003. To appear.
18. V.R. Benjamins, B. Wielinga, J. Wielemaker, and D. Fensel: *Brokering Problem-Solving Knowledge at the Internet*. Proceedings of the European Knowledge Acquisition Workshop (EKAW-99), Lecture Notes in Artificial Intelligence, LNAI 1621, May 1999.
19. J.C. Arpirez, O. Corcho, M. Fernández-López, and A. Gómez-Pérez: *WebODE – A Scalable Ontological Engineering Workbench*. Proceedings of the First International Conference on Knowledge Capture, Victoria, Canada, October 2001.
20. V.R. Benjamins: *Web Service Solve Problems, and Problem-Solving Methods Provide Services*. IEEE Intelligent Systems, 18(1):76-77, January/February 2003.
21. P.V. Biron and A. Malhotra: *XML Schema Part 2: Datatypes*. http://www.w3c.org/TR/2001/REC-schema-2-20010502, May 2001.
22. G. Schreiber, H. Akkermans, A. Anjevierden, R. de Hoog, H. Shadbolt, W. van de Welde, and B. Wielinga: *Knowledge engineering and management. The CommonKADS Methodology*. MIT Press, Cambridge, Massachusets.
23. A. Newell: *The Knowledge Level*. Artificial Intelligence, 18(1):87-127, 1982.
24. O. Corcho, M. Fernández-López, A. Gómez-Pérez, and O. Vicente: *WebODE – An Integrated Workbench for Ontology Representation, Reasoning and Exchange*. Proceedings of the Thirteenth International Conference on Knowledge Engineering and Knowledge Management (EKAW'02), LNAI 2473, pages 138-153, Sigenza, Spain, October 2002.
25. E. Sirin, J. Hendler, and B. Parsia: *Semi-automatic Composition of Web Services using Semantic Descriptions*. Proceedings of the Workshop on Web Services: Modeling, Architecture and Infrastructure in conjunction with ICEIS2003. 2003. Accepted.
26. S. Narayanan and S.A. McIlraith: *Simulation, Verification and Automated Composition of Web Services*. Proceedings of the Eleventh International World Wide Web Conference (WWW-2002), pages 77-88, Hawaii, USA, May 2002.