An Environment for Development of Semantic Web Services

Oscar Corcho[(1)], Mariano Fernández-López[(1)], Asunción Gómez-Pérez[(1)] and 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.
email: {ocorcho, mfernandez, asun}@fi.upm.es

[(2)] Departamento de Electrónica y Computación. Facultad de Física.
Campus Sur, s/n. Universidad de Santiago de Compostela,
15782 Santiago de Compostela, A Coruña, Spain
email: lama@dec.usc.es

Abstract

Semantic Web Services (SWSs) are specified in a semantic markup language to enable other services (and agents) to reason about their capabilities, in order to decide whether a SWS should be invoked or not. In this paper an environment for development and description of SWSs is presented. This environment, called ODE SWSDesigner, consists of a graphical interface, which allows users to carry out the design and characterization of SWSs at a conceptual level, and a set of software modules, which verify the design correctness and perform the translations from the graphical descriptions to the languages used to specify SWSs. ODE SWSDesigner provides support for a layer-based framework that we have proposed with the aim of enabling a language-independent development of SWSs. This framework is based on the use of problem-solving methods that are considered as high-level specifications from which SWS descriptions can be generated and verified.

1 Introduction

Web Services (WSs) are software modules that describe a collection of operations that can be network-accessible through standardized XML messaging [Kreger, 2001]. WSs are distributed all over 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 [Webber and Dutton, 2000], E-Speak [Graupner et al., 2000], or BPEL4WS [Curbera *et al.,* 2002]. 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 [Kreger, 2001]. This framework is composed of a set of layers that, basically, enable: (1) *WS publication*, where the UDDI specification [Bellwood *et al*., 2002] is used to define the WS capabilities and characterize its provider; (2) *WS description*, which use the WSDL language [Christensen *et al.,* 2001] to specify how the service can be invoked (input-output messages), and SOAP [Biron and Malhotra, 2001] as the communication protocol for accessing to WS; and (3) *WS composition*, which specifies how a complex service can be created from the combination of other services. The language used to describe this composition is WSFL [Leymann, 2001].

In this context, the Semantic Web [Berners-Lee *et al.,* 2001] has risen as a Web evolution where the information is *semantically* expressed in a markup language (such as DAML+OIL [Hendler and McGuinness, 2000]) 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 [McIlraith *et al.,* 2001; Hendler, 2000]. This markup would enable other services to reason about the SWS, and, as a result, decide whether it match their requirements. Taking this into account, two frameworks, SWSA [Sollazzo *et al.,* 2001] and WSFM [Fensel and Bussler, 2002], 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 [Ankolenkar *et al.,* 2001], 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 [Benjamins and Fensel, 1998]. The aim of the PSMs is to be *reusable* components applicable to different, but similar, domains and tasks. 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*), that is specified in a general manner to become PSM reusable in different domains (characterized by a domain ontology). The UPML specification [Fensel *et al.,* 2003] provides containers in which these PSM views can be described, and, also, it incorporate elements that enable the PSM reuse. UPML has been developed in the context of the IBROW project [Benjamins et al., 1999] with the aim of enabling the semi-automatic reuse of PSMs. This objective could be interpreted as a composition of PSMs.

In this work our aim is to provide a *development environment* of SWSs, which would allow the user to design SWSs on the basis of PSM modeling (at a conceptual level). This environment also should perform verification about the soundness 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 language. These two features (PSM-based and language-independent design) are the main differences between our environment and other tools [Narayanan and McIlraith, 2001; Sirin *el al*., 2003], which use DAML-S to specify SWS description and composition, and to verify the SWS consistency.

The structure of the paper is as follows. In section 2 a PSM-based framework to develop SWSs (and WSs) is presented. In section 3 we describe the software architecture of the environment that supports this framework, and in section 4 the current capabilities of its graphical interface are explained. Finally, in section 5 the main contributions of the work are summarized.

2 Framework for SWS development

The framework that we propose for SWS development is based on the assumption that, in essence, SWSs (and WSs) could be considered as PSM *specializations*. This specialization means that SWSs do not need to be expressed in a general manner, because they are not aimed to be reusable in different domains or tasks. Therefore, the PSM method ontology is the same ontology as the one used in the SWS specification.

Relation between PSMs and SWSs

Both SWSs and PSMs are paradigms in which an operation (or *equivalently* a method) is executed to perform a task in a domain, and, as a result, it may obtain new domain information or provoke an effect in the real world. Taking this common objective into account, it seems to be reasonable to analyze whether PSMs may be used to enable the publication, description and composition of both WSs and SWSs.

- *Publication*. A PSM definition does not usually show detailed information about its provider or the industry segment in which the PSM could be included. Although the UPML specification provides some information, in order to publish and discover SWSs it becomes necessary to extend it with data typically used in ecommerce interactions, such as quality or geographical situation of the provider.
- *Description*. PSM specification details the input-output interactions between the PSM components (task interactions and method ontology). Figure 1 shows how the elements which define the WSDL specification (de-

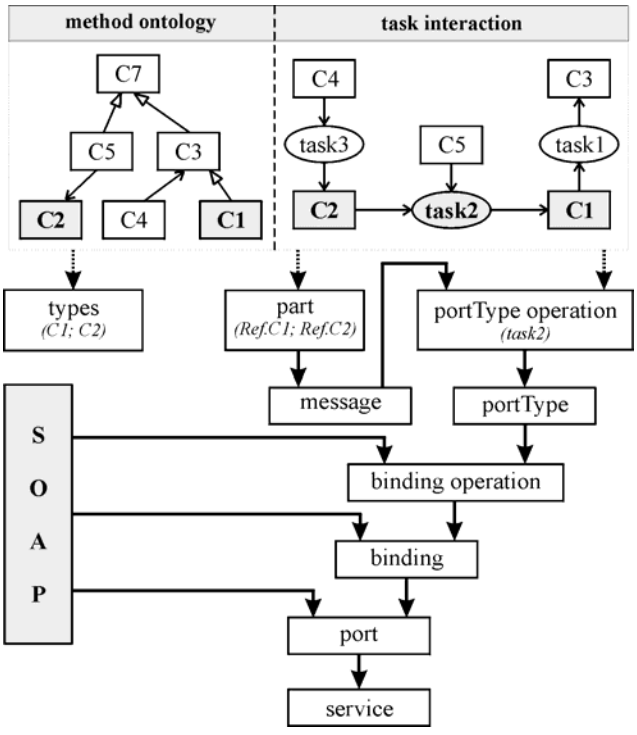

Figure 1. Obtaining WS descriptions (in WSDL) on the basis of PSM specifications where . We have assumed that the method ontology is the same ontology as the one used in the WS specification. If these ontologies were not the same, it would be necessary to establish the mappings between them.

noted with white boxes) can be completely extracted from a description of both task interactions and method ontology (dashed arrows), and from other WSDL elements (solid arrows). This knowledge will be enough to describe the SWS in order to enable its invocation. However, PSMs do not specify the communication protocol that allows them to be invoked through a network.

- *Composition*. PSMs specify in detail how a task must be executed (flow control) and of which elements the PSM is composed of (methods-tasks tree). These specifications include the conditions in which the PSM elements (or subtask) should be executed and how those elements are combined to obtain the required result. On the basis of this information, the SWS composition could be enabled.

Considering this analysis we can conclude that there is a direct relation between PSMs and SWSs: PSMs can be used to specify SWSs (and WSs) features that are related to their internal structure (description and composition). However, we need to *extend* the PSM specification with knowledge related to ecommerce features, to enable SWS discovery, and communication protocols, to provide network-accessibility.

Framework Requirements

The design of the framework has been guided by a set of criteria (or requirements) that establish the conditions for defining an open and extensible framework for SWS development. These criteria are the following:

1. *SWS conceptual modeling*. SWS development must be carried out at conceptual level: 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.
2. *Integration with Web Service standards*. SWS specifications should be integrated with Web service *de facto* standards (both frameworks and languages) to be able to use its benefits and the current infrastructure that supports these standards [Sollazzo *et al.,* 2001]. This criterion complements the SWS conceptual modeling, because it fixes the specific languages the SWS model must be translated to.
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 can include new modules without modifying the others.

2.1 Layer-Based Framework

In order to cover these criteria 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 described by an ontology that defines its elements on the basis of well-known standards. These ontologies (or layers) are the following (see figure 2):

- *Data Types (DT) Ontology*. It contains the data types associated with 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 [Biron and Malhotra, 2001].
- *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 tools that use the framework higher ontologies (PSM and SWS ontologies) could need to reason about the domain ontology itself. For example, preconditions of a method could impose that the input-output data should be "attributes". Usually, the KR ontology will be associated with 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 [Fensel et al., 2003], that has been extended with (1) a *programming structures* ontology, which describes the primitives used to specify the PSM flow control (such as conditional and parallel

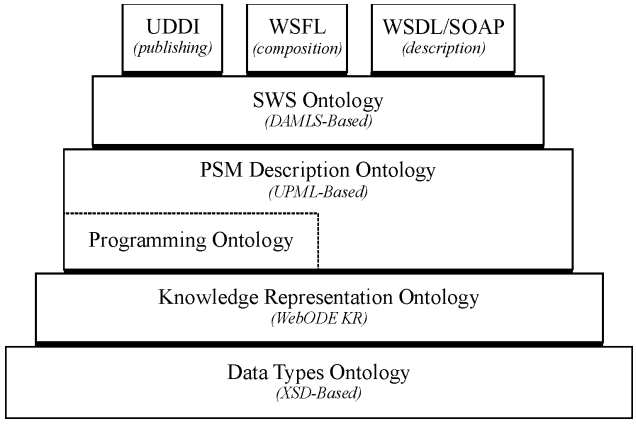

Figure 2. Framework for SWS development. It is composed of a set of design layers, each one described by an ontology that is based on well-known specifications (*de facto* standards).

loops, conditional statements, etc.); (2) *inferences*, which are new PSM elements defined as in the CommonKADS knowledge model [Schreiber *et al*., 1999], that is, as building blocks for reasoning processes; and (3) relations among 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). On the other hand, the PSM ontology contains a number of axioms that constrains how PSM element instances are created. That guarantees the soundness of the PSM model. For example, it exists an axiom establishing that the inputs method must be covered by the inputs associated with 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 [Ankolenkar *et al.,* 2001], because it describes containers to include these types of knowledge.
- *Standard Language Ontologies for Web Services*. They describe the elements associated with the *de facto* Web standard languages for service publication (UDDI), description (WSDL/SOAP), and composition (WSFL). These ontologies complete the SWS specification, because they facilitate its integration in the current infrastructure of the Web.

This framework satisfies the design requirements. In effect, conceptual modeling of SWSs is performed in the PSM layer, which is not constructed following a specific language, but is modeled at knowledge level [Newell, 1982]; integration with Web service standards is explicitly enabled in the framework highest layer, which, if required, could be easily extended to include new standards; and, finally, modular design is associated with the layered approach itself.

3 Environment for SWS development

In order to provide support for the framework, we have developed an environment with the aim of allowing users to design the conceptual model of a SWS by means of a graphical interface. Once this model is created, it could be exported to a DAML+OIL specification (such as DAML-S), which will be complemented with Web service standard languages. This environment, called *ODE SWSDesigner*, has been designed following the framework requirements: to develop an open and easily extensible environment that, if required, could be adapted to support new SWS (and WS) specification languages or frameworks.

In addition, ODE SWSDesigner is integrated into WebODE [Arpírez et al., 2001], 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. Integration in WebODE will allow SWSDesigner to invoke those services whose capabilities needs in its operation, such as services for exporting an ontology (SWS and PSM) to a specific language (DAML+OIL and Java respectively) or checking constrains in ontologies.

3.1 Software Architecture

In accordance with the proposed framework, the design and development of SWSs could be viewed as the process of *instantiating* an ontology set that contains the knowledge needed to generate the SWS specifications. Software architecture of ODE SWSDesigner is based on this consideration and, as we can see in figure 3, it is composed of two different types of modules: a *graphical interface*, which allows the users to develop SWSs at a conceptual level, and a set of *instance processors*, which are software modules that process the SWS graphical descriptions (created by the users) to generate the instances associated with the ontologies of the framework, and, if required, to check the correctness of the generated instances. The instance processors, which have been included in WebODE as *services*, are the following:

- *KR service*. This processor gets as input the ontology used in SWS operation (usually the domain ontology) and establishes the instances associated with the KR and Data Types ontologies. The domain ontology can be available in WebODE or could be translated from an ontology language into the WebODE specification. In both cases, this processor will invoke the ODE service to access to the domain ontology elements, which are saved in a database (figure 3).
- *PSM service*. It uses the graphical descriptions of the SWS model created by the user to generate an instance set that describes completely the PSM internal structure and flow control (PSM model). Once the instance set is created, this processor must invoke the inference WebODE service [Corcho *et al*., 2002] to verify the soundness 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 defined a general service that is decomposed in two sub-services, it is necessary to verify that the inputs of these sub-services have the same (or subsumed) type as the general service inputs. In order to perform this verification, the PSM processor must operate with an explicit description of the representation primitives in which the domain ontology will be instanced.

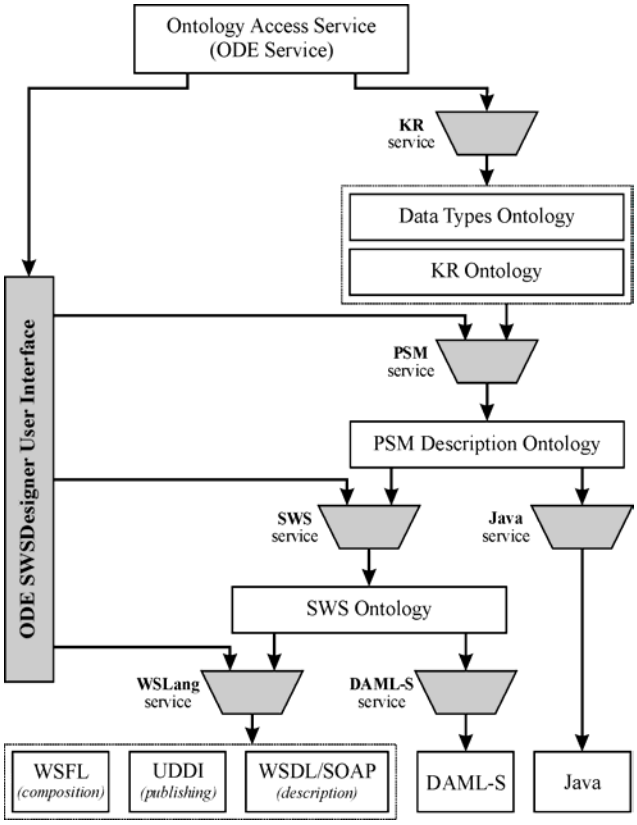

Figure 3. ODE SWSDesigner software architecture. The instance processors are integrated into WebODE as *services*, which perform translations between the adjacent ontologies (or layers) of the framework proposed for developing SWSs.

- *SWS service*. Instances created by this processor will enhance the knowledge included in PSM ontology instances by adding the information used in ecommerce interactions. This information will be directly obtained from the graphical interface.

These three instance processors represent the *SWSDesigner core*, because they support the generation of the SWS model and their operation does not depend on the languages in which the SWS will be expressed. Thus, these processors will be modified only if their associated layers are 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 languages.
- *DAML-S service*. It obtains the DAML-S specification of the SWS getting as inputs the instances of the SWS ontology. This operation, nevertheless, is not straightforward because in the DAML-S ontology a service is modeled as a *process*, while in our framework a service is considered to be a specialization of a PSM (or method). Once this operation is performed, this processor must invoke the WebODE service that exports an ontology to the DAML+OIL language.
- *Java service*. Using the PSM ontology instances, this processor generates the skeleton of the programming code (Java beans) needed to execute the SWS and perform its operation. Once this code has been created, the user must fill in the methods responsible of carrying out the operation modeled in the PSM.

These three processors represent *SWSDesigner additional processors*, because they have been specifically included into the framework to obtain SWS (or WS) specifications in various languages. This means that these processors would be changed (or substituted) if it was required to use other languages or if the core processors were also modified.

On the other hand, instance processors are directly invoked from the graphical interface when the users, after creating the SWS conceptual model, require to export that model to well-known WS languages or when the graphical interface itself needs to verify whether an operation carried out by the user has generated an inconsistent model of the SWS.

4 Graphical Interface

ODE SWSDesigner 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 modeling). These different views help the user to understand the internal structure of a service and the interactions between its components (sub-services). Taking this into account, the graphical interface contains the following views, which reflect how PSMs are designed (see example of figure 4):

- *Definition view*. In this view the user defines a service by specifying its name (mandatory) and, optionally, by introducing the information needed for enabling 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 specify (and also create) the services (sub-services) that could be executed when a service (composite) is activated. That is, a service hierarchy is specified. This hierarchy could be used for service composition and for checking inconsistencies in the other graphical interface views.
- *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 be previously loaded from WebODE database to the graphical interface. Figure 4 shows the main window of the ODE SWSDesigner, where we

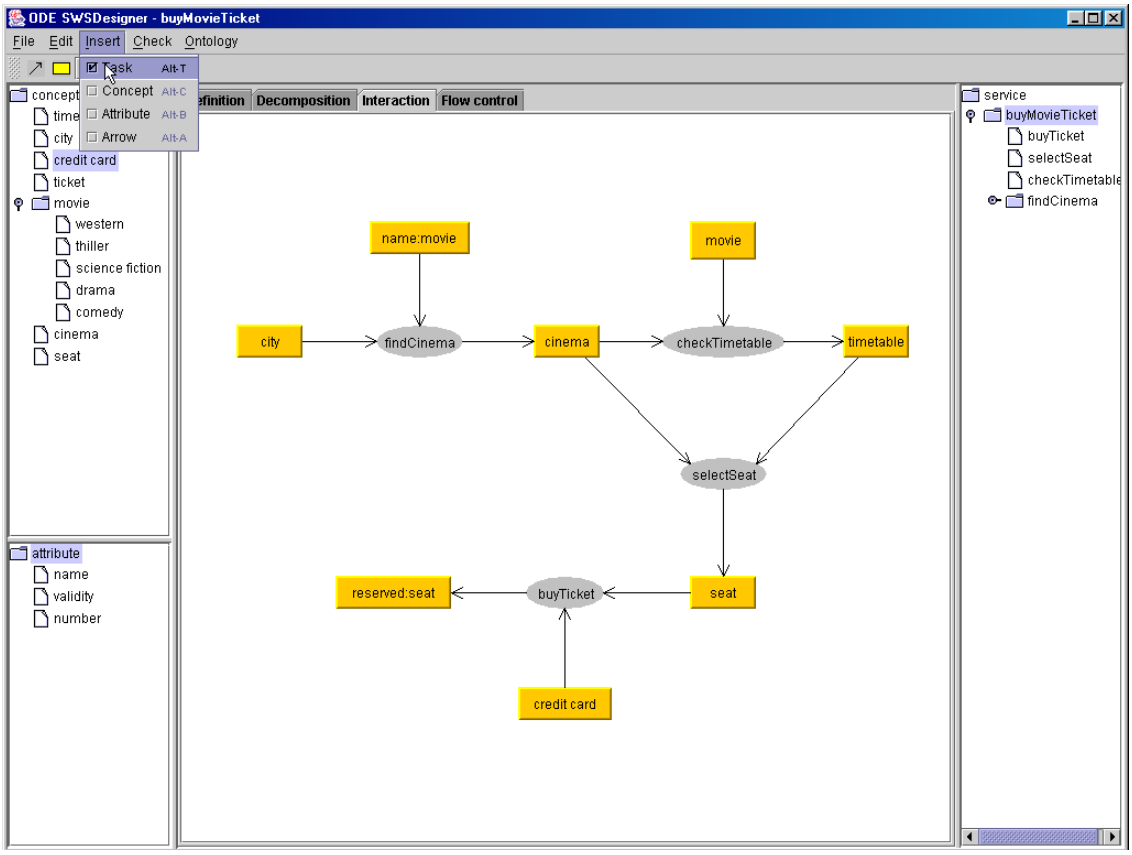

Figure 4: Main window of the ODE SWSDesigner graphical interface. The example shows the input-output interactions among the sub-services that compose the *buyMovieTicket*, where ellipses and rectangles represent sub-services and concepts/attributes respectively.

can see the specification of the interactions between the sub-services of *buyMovieTicket* composite service. All these services have been created in the decomposition view, which generates the service tree shown in the right side of figure 4.

- *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 the user will create to 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 soundness 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 processor to assure that the interaction view contains exactly those three services (as shown in figure 4).

5 Conclusions

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

Nowadays, we have implemented the definition, composition and interaction views of the graphical interface, and DAML-S is the current SWS language into which the designed SWS are translated. In addition, we are extending the PSM and SWS ontology with new axioms. These extensions will allow us to cover additional conditions to check SWS consistency and completeness.

On the other hand, the ODE SWSDesigner integration in WebODE has simplified its software architecture and implementation, because (1) it uses directly the WebODE services, which offer support for ODE SWSDesigner 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 SWSDesigner modularity, which is a key requirement to adapt the environment to new standard languages or frameworks.

References

[Ankolenkar *et al*., 2001] A Ankolenkar, M Burstein, JR Hobbs, O Lassila, DL Martin, SA 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.

[Arpírez et al., 2001] J.C. Arpírez, O. Corcho, M. Fernández-López, and A. Gómez-Perez. WebODE: a scalable ontological engineering workbench. Proceedings of the First International Conference on Knowledge Capture, Victoria, Canada, October 2001.

[Bellwood *et al*., 2002] 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.

[Benjamins et al., 1999] 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.

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

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

[Biron and Malhotra, 2001] PV Biron and A Malhotra. XML Schema Part 2: Datatypes. *http://www.w3c.org/TR/2001/REC-schema-2-20010*502, May 2001.

[Box et al., 2000] 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.

[Christensen *et al*., 2001] 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.

[Corcho *et al*., 2002] 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, Sigüenza, Spain, October 2002.

[Curbera *et al*., 2002] 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.

[Fensel and Bussler, 2002] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Proceeding of the NSF-EU Workshop on Database and Information Systems Research for Semantic Web and Enterprises*, pages 15–20, Georgia, USA, April 2002.

[Fensel et al., 2003] 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, a B. Wielinga. The Unified Problem-solving Method Development Language UPML. *Knowledge and Information Systems (KAIS)*: *An International Journal,* 2003. To appear.

[Gómez-Pérez and Corcho, 2002] O Corcho and A Gómez-Pérez. Ontology languages for the Semantic Web. *IEEE Intelligent Systems*, 17(1):54–60, 2002.

[Graupner et al., 2000] 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.

[Hendler, 2000] J. Hendler. Agents and the Semantic Web. *IEEE Intelligent Systems*, 16(2):30–37, 2001.

[Hendler and McGuinness, 2000] J. Hendler and D. McGuinness. The DARPA Agent Markup Language. *IEEE Intelligent Systems*, 15(6):72–73, 2000.

[Kreger, 2001] H Kreger. Web Services Conceptual Architecture (WSCA 1.0). *http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf*, May 2001.

[Leymann, 2001] F Leymann. Web Service Flow Language (WSFL 1.0).

http://www.ibm.com/software/solutions/webservices/pdf/WSDL.pdf, May 2001.

[McIlraith *et al*., 2001] SA. McIlraith, TC Son and H Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46−53, 2001.

[Narayanan and McIlraith, 2001] S Narayanan and SA 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.

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

[Schreiber *et al*., 1999] 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, 1999.

[Sirin *el al*., 2003] E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of Web Services using Semantic Descriptions. *Proceeedings of the Workshop on Web Services: Modeling, Architecture and Infrastructure* in conjunction with ICEIS'2003. Accepted.

[Sollazzo *et al*., 2001] 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.

[Webber and Dutton, 2000] D Webber and A. Dutton. Understanding ebXML, UDDI and XML/edi. *http://www.xmlglobal.com/downloads/ebXML_understanding.pdf*, October 2000.