ODE SWS: A Semantic Web Service Development Tool

Asunción Gómez-Pérez
Universidad Politécnica de Madrid
Depto. de Inteligencia Artificial
Campus de Montegancedo s/n
28660 Boadilla del Monte, Spain
Phone: +34 91 3367439
asun@fi.upm.es

Rafael González-Cabero
Universidad Politécnica de Madrid
Depto. de Inteligencia Artificial
Campus de Montegancedo s/n
28660 Boadilla del Monte, Spain
Phone: +34 91 3367467
rgonza@fi.upm.es

Manuel Lama
Universidade de Santiago de Compostela
Depto. de Electrónica e Computación
Campus Universitario Sur s/n
15782 Santiago de Compostela, Spain
Phone: +34 981 563100 ext 13571
lama@dec.usc.es

ABSTRACT
ODE SWS is a development environment to design Semantic Web Services (SWS) at the knowledge level. ODE SWS describe the service following a problem-solving approach in which the SWS are modeled using tasks, to represent the SWS functional features, and methods, to describe the SWS internal structure. In this paper, we describe the ODE SWS architecture and the capabilities of its graphical interface, which enables users to design SWS *independently* of the semantic markup language used to represent them.

Categories and Subject Descriptors
I.2 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods.

General Terms
Design, Languages.

Keywords
Web Services, Semantic Web, Problem-Solving Methods, Design Tools.

1. INTRODUCTION
A Web Service (WS) is an interface that describes a collection of operations, which are network-accessible through standardized XML-messaging, and that is specified following a standard XML-based language [1]. A WS in the Semantic Web [2], called *Semantic Web Service* (SWS) [3], is an interface with a description based on an ontology that semantically describes all the WS features, and that is expressed in a semantic markup language like OWL [4]. Once the SWS has been specified in that language, software agents would use the reasoning capabilities to discover and compose new services whose features would match the requirements of those agents [5].

However, as a previous step to the specification of SWS in a semantic markup language, the SWS should be designed at a knowledge or conceptual level to avoid inconsistencies or errors among the services that constitute the SWS. In this context, SWS design consists in specifying the non-functional, functional, and structural features of the service. Currently, there are some enviroments to edit/design SWSs [6,7,8] that present the following features:

- They are *language-dependent*, because their graphical interface consists of a set of containers that represent all the service features. Thus, to design a SWS, the user must fill out these containers by typing the features in a semantic language (like OWL or F-Logic). This process design favours the occurrence of errors and inconsistencies in the service development.
- Some environments [7,8] are really *ontology editors* (developed as plug-ins of Protégé), where the user designs the service by instantiating the concepts and relations of the ontology that describes the service features. Therefore, in these environments the SWS structure is hidden (that is, the service is not visually developed), and if there were a change in the SWS ontology, the graphical interface itself should be modified to be adapted to this change.

To solve those drawbacks, we have developed an environment, called *ODE SWS*[1] [9], to design SWSs at the knowledge and independent-language level. This environment is based on the assumption that a SWS is modeled as a Problem-Solving Method (PSM) [10], where a task defines the service functional features, and the methods that solve such task describes both the internal components of the service and the control of the reasoning process required to execute it.

In this paper, we describe the architecture and main features of ODE SWS, and, specially, we will focus on the capabilities of its graphical interface, called *SWSDesigner*. This interface is directly based on the views of the PSM classical modeling: (1) in the interaction and logic views, the task associated to the service and the methods that solve such task are defined; (2) in the task-method hierarchy view, the internal components of the service are specified; (3) in the knowledge flow view, the data flow between the components of the service is described; and (4) in the control flow view, the coordination of the service components is specified.

This paper is structured as follows: in the following section the main features of ODE SWS are presented; then, we will show the software architecture of the design environment, and, finally, in the section 4 the graphical interface of ODE SWS is described.

2. ODE SWS Features

2.1.1 Semantic Web Service Description
ODE SWS follows a problem-solving approach for modeling the services. According to this approach, tasks, methods, and adapters are introduced to describe the service features and favour the reuse of the services:

- *Task*, describes the functional features of a service; that is, its input/output roles, pre/post-conditions, and effects. A task describes, on one hand, the roles and conditions required to exe-

[1] ODE SWS is available at http://kw.dia.fi.upm.es/odesws.

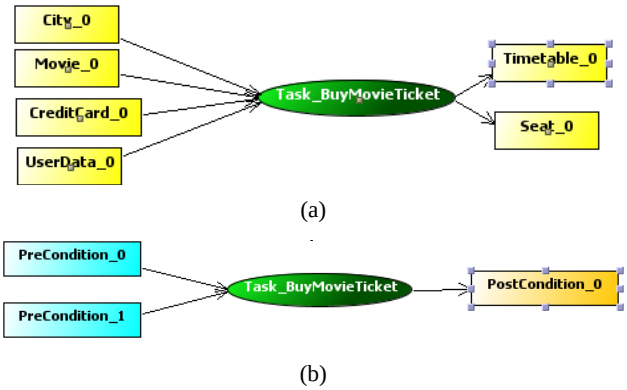

Figure 1. A task has (a) input and output roles; and (b) a set of pre/post-conditions.

cute the service (input roles and pre-conditions), and, on the other hand, the results of the service execution (output roles, post-conditions and effects). Figure 1 shows the representation (as an ellipse) of the task *Task_BuyMovieTicket*, which defines the functional features of the service *BuyMovieTicket*.

- *Method*, describes how a task can be solved; that is, they specify the control of the reasoning steps needed to solve a given task. A method is used to define the internal structure of the service: its components and the coordination of the execution of those components (usually represented as a workflow). A method is defined by a set of input/output roles, the pre-conditions to be verified for executing the method, and the post-conditions that describe the world state once the method has been executed.

  A method also describes the internal components that are executed to solve a task. These components are tasks (so-called sub-tasks), each of which is solved by other method that can be composed of other sub-tasks, and so forth. Figure 2.(a) shows an example of the internal description of a method (represented as a rounded corner square): the task *Task_BuyMovieTicket* is solved by the method *Method_BuyMovieTicket*, which is composed of the sub-tasks *Task_FindCinema*, *Task_SelectSeat*, *Task_CheckTimetable*, and *Task_BuyTicket*. The description of the internal structure of the method is completed with the definition of the execution coordination of its sub-tasks; that is, a method specifies the control flow (reasoning process) among its sub-tasks. Figure 2.(b) shows a graphical example of the coordination of the sub-tasks of the method *Method_BuyMovieTicket*: first, the tasks *Task_FindCinema* and *Task_CheckTimetable* are executed to select a theater that shows a movie in a given time; then, the task *Task_SelectSeat* is executed in a loop until the user selects the desired seat; and, finally, the task *Task_BuyTicket* is executed to buy the ticket (typically using a credit card).

- *Adapters*, describe the conditions in which a method could be applied to solve a particular task, and the mappings between the method ontology and the task ontology. Typically, a task can be solved by several methods. For example, to solve the task *Task_BuyMovieTicket* a method could not include the task *Task_FindCinema* if the theatre were an input. Currently, the task-method adapters are not supported by ODE SWS, which means that the tasks could be solved by an *only* method, and both tasks and methods use the same ontology.

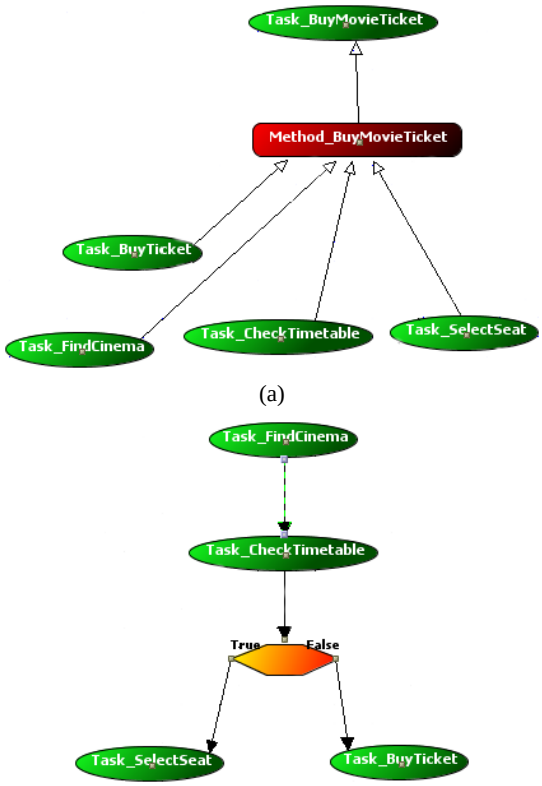

Figure 2. A method (a) is composed of a set of sub-tasks, which are solved by other methods; and (b) defines the coordination of the execution of the sub-tasks.

2.1.2 KnowledgeLevel Design

In ODE SWS, the user will design the services independent of the semantic markup language in which they will be represented to be accessed by the external agents. Thus, the services are designed at the knowledge level by means of a graphical interface that is inspired in the classical views of the PSM modeling (see section 4). As a result of the design process, the user obtains a SWS graphical model, represented in an XML-based format, that describes the content of the PSM graphical views. This model, therefore, is the representation of the SWS itself, but it could not be used for reasoning about the service features beause the XML format does not describe semantically the service. However, ODE SWS provides an export facility that translates a service from its graphical model into the semantic markup language (like OWL-S or WSMO) selected by the user to represent the service.

Therefore, in ODE SWS, the design of the services is based on the *paradigm* of the problem-solving methods, and it is independent of both the semantic markup languages and the knowledge representation models of such language. Thus, if new languages were proposed to represent SWSs, in ODE SWS the design of services would not change, but it would need to create a new export module to translate from the graphical model into those languages.

2.1.3 Ontology Management

The input and output parameters of a SWS should be described in a domain ontology (usually as concepts and attributes) to enable external agents to reason about the SWS competence. Moreover,

complex services manage several ontologies to describe different kinds of parameters. For example, a *theater booking* service could manage a *theater* ontology, and, additionally, a *transport means* ontology to describe how the user could go to the theater.

ODE SWS assumes that the ontologies used by the service will be constructed in an ontology development environment, where they could be evaluated to solve errors and inconsistencies. ODE SWS, however, can manage *multiple* ontologies in read mode, and these ontologies could be imported from OWL or RDF(S), or directly loaded from a WebODE server [11].

3. ODE SWS Architecture

The software architecture of ODE SWS is composed of three main layers, which reflect the layers introduced in a classical software design (Figure 3):

Presentation layer. This layer manages the interaction between the user and the software system. It is entirely composed of the ODE SWS graphical editor, called *ODE SWSDesigner*, that provides facilities to construct graphically a service. As result of this construction a graphical model of the service, called *ODE SWSgm*, will be generated. The main functionalities of this editor are: **(1)** the appropriate management and representation of the service model; and **(2)** the graphical processing of all the possible interactions among the elements that compose such model.

Domain layer. This layer contains all the components that operate in the domain of ODE SWS (that is, development of SWS). As Figure 3 shows, the ODE SWSDesigner directly will invoke the execution of the components of this layer to support the execution of the operations needed to guarantee the correct design of the service. The intended functionalities of these components are:

- *SWSOntologiesManager*. The aim of this module is both to offer a uniform manner for accessing to ontologies implemented in different languages, and to access to repositories of ontologies. Currently, this module can manage either ontologies implemented in RDF(S) and OWL, or ontologies stored in the WebODE platform.
- *SWSMappingsManager*. This module deals with the mappings that are (semi) automatically defined between the task ontologies and the method ontologies. Currently, the user establishes manually these mappings through the *SWSDesigner*, and the *SWSMappingsManager* automatically generates the adapters between tasks and methods.

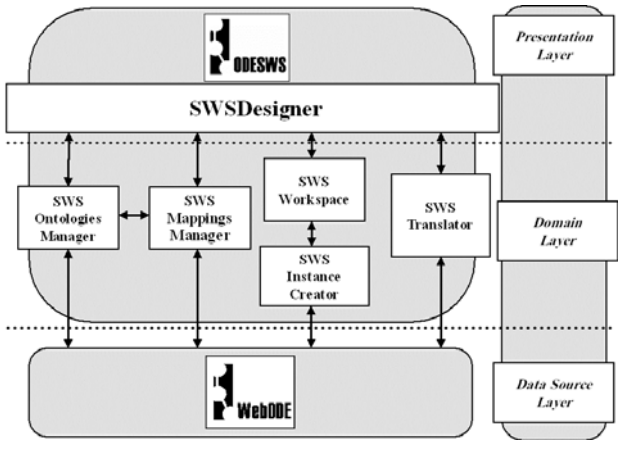

Figure 3. Software architecture of the ODE SWS environ-

- *SWSWorkspace*. In the development process of a SWS, an incomplete, and even inconsistent, representation of the service may be stored and managed (to be completed later). This module performs these activities, enabling the store and recovery of ongoing SWS.
- *SWSInstanceCreator*. This module generates the instances of the stack of ontologies that describe the features of the SWS following the problem-solving based approach (described in section 2). These instances will be created from the graphical representation of the service generated by *SWSDesigner*.
- *SWSTranslator*. This module supports the translation from the graphical representation of the service into a SWS-oriented markup language (as OWL-S). Thus, once the service has been developed, the user will select the language in which the service will be expressed. Then, the *SWSTranslator* generates automatically the translation and, additionally, it incorporates information related to the problem that might risen in the translation process. Currently, services can be exported to OWL-S and/or WSDL.

Data Source Layer. In this layer, other applications act on behalf of the ODE SWS environment to provide support for operations do not implemented in the environment. For ODE SWS, this layer will be just composed by the WebODE platform, which will provide services for the management and access to the ontologies used in the SWS development.

4. SWSDesigner: ODE SWS INTERFACE

The design of ODE SWSDesigner has been inspired in the classical modelling of the problem-solving methods, including hierarchical trees of tasks-methods, input/output interaction diagrams among the sub-tasks that compose a method, and diagrams to specify the control flow that describes the coordination of the execution of the sub-tasks. Taking this into account, in the ODE SWSDesigner we distinguish the following graphical components (Figure 4):

- *Task (Method) trees* allow users to just create the tasks (methods) associated to the functional features (internal structure) of a service. Once the tasks (methods) have been created, from these trees the user could drag the icons representing a task (method) to be dropped into the views that allow the specification of the service features. As Figure 4 shows, these trees are on the right of the SWSDesigner.
- *Ontology trees* show the concepts and attributes of the ontology (or ontologies) used to specify the service input/output roles. Although ODE SWS cannot create ontologies, it just can import their concepts and attributes that will be showed in the ontology tree. Then, the user could drag the icons representing a concept/attribute to be dropped into the views that allow the specification of the input/output roles of both tasks and methods. As Figure 4 shows, the ontology trees are on the left of the SWSDesigner.
- *Service definition View* is used to specify the functional and non-functional features of a service, including its input/output roles (interaction diagram), pre/post-conditions (logical diagram), providers, commercial classification, geographical location, and quality rating parameters.
- *Decomposition View* allows users to specify the decomposition of the method (that solves the task associated to the service) into its sub-tasks, which will be solved by other methods, and so forth. The user carries out this specification by dragging the

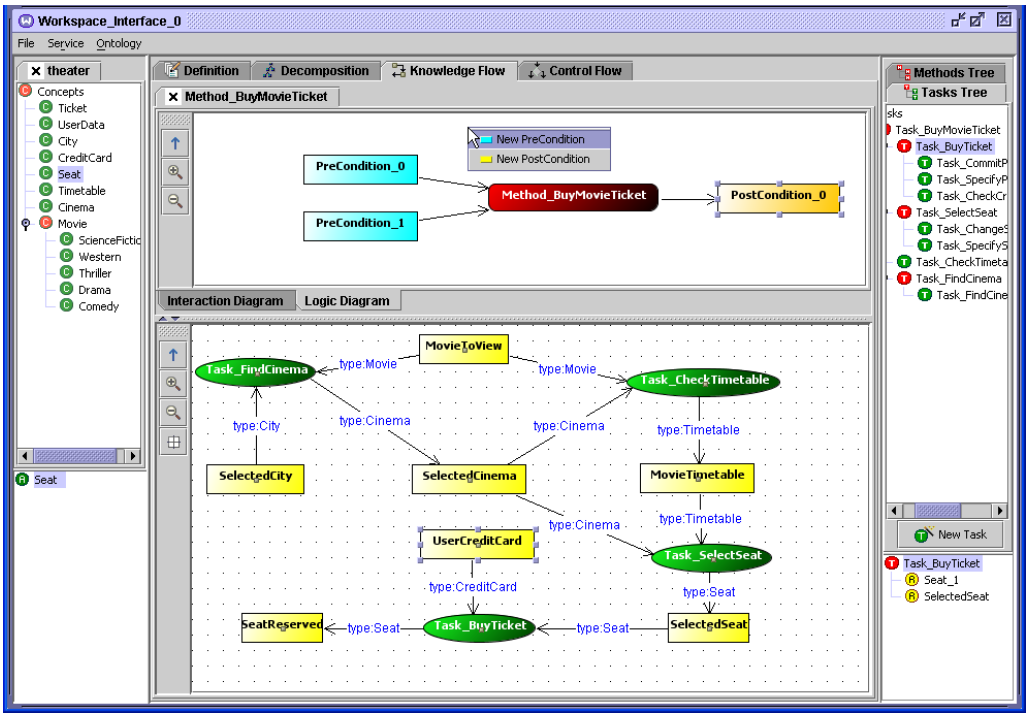

Figure 4. Knowledge flow view of the ODE SWS graphical interface (*SWSDesigner*).

icons of the tasks and methods from the related trees and dropping such icons into the view.

- *Knowledge Flow View* allows users to define the input/output interactions among the sub-tasks of a method: the user, when requires, connects the output of a sub-task to the input of other sub-task, and establish the *mappings* between the roles used in the definition of a sub-task (carried out in the service definition view) and the roles named when such sub-task is used as an internal component of a method. For example, *City_0* could be an input role of the task *Task_FindCinema* and when that task is used as part of the method *Method_BuyMovieTicket*, its input role could be named as *selectedCity*. In this view, the user also defines the *mappings* between the roles of a method and the roles of the task solved by that method: the role names of methods and tasks could be different.
- *Control Flow View* enables users to describe the control flow of the method. The elements of this view are the sub-tasks of the method, which are dragged-and-dropped from the task tree, and the workflow constructions (if, while-until, split and join), which are introduced through a contextual menu.

5. ACKNOWLEDGMENTS

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

6. REFERENCES

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

[2] Berners-Lee, T., Hendler, J., and Lassila, O. The Semantic Web. Scientific American, 284 (2001), 34-43.

[3] McIlraith, S.A., Son, T.C., and Zeng, H. Semantic Web Services. IEEE Intelligent Systems, 16 (2001), 46-53.

[4] Dean, M., and Schreiber, G. (eds.). OWL Web Ontology Language Reference. W3C Candidate Recommendation. http://www.w3c.org/TR/owl-ref (2004)

[5] Hendler, J. Agents and the Semantic Web. IEEE Intelligent Systems, 16 (2001), 30-37

[6] Dimitrov, M., Marinova, Z., and Radkov, P. SWWS Studio – A WSMO compliant editor. Proceedings of the 1st WSMO Implementation Workshop (Frankfurt, September, 2004). http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-113/pospaper3.pdf (2004)

[7] Lausen, H., and Felderer, M. WSMO Editor. WSMO Working Draft. http://www.wsmo.org/2004/d9/v0.1 (2004)

[8] Elenius, D. Modeling services with Protégé. In Proceedings of the 7th International Protégé Conference (Bethesda, 2004) http://protege.stanford.edu/conference/2004/posters/Elenius.pdf

[9] Gómez-Pérez, A., González-Cabero, R., and Lama, M. ODE SWS: A Framework for Designing and Composing Semantic Web Services, IEEE Intelligent Systems, 19 (2004), 24-32.

[10] Benjamins, R.V., and Fensel, D. (eds.). Problem-Solving Methods. International Journal of Human-Computer Studies, 49 (1998) 4 (entire issue).

[11] Arpírez, J.C., Corcho, O., Fernández-López, M., and Gómez-Pérez, A. WebODE in a nutshell. AI Magazine, 24 (2003) 37-48.