SWSDesigner: The Graphical Interface of ODESWS

Asunción Gómez-Pérez[1], Rafael González-Cabero[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.
asun@fi.upm.es, rgonza@delicias.dia.fi.upm.es

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

Abstract. ODESWS is a development environment to design Semantic Web Services (SWS) at the knowledge level. ODESWS describe the service following a problem-solving approach in which the SWS are modelled using tasks, to represent the SWS functional features, and methods, to describe the SWS internal structure. In this paper, we describe the ODESWS graphical interface (called SWSDesinger). This interface enables users to design SWS independently of the semantic markup language in which the service will be implemented, and once the design has been export the service to an SWS implementation language.

Introduction

Currently, there are some proposals to edit/design SWS, but the main drawback of these available editing tools is that they work at the representation level. Cosecuently, these tools are language-dependent, like the WSMO Editor [1], this means that: (1) SWS designed with these tools are less reusable; (2) the designs are more prone to inconsistencies or errors; (3) the design can be constrained with the chosen language characteristics. Also, many tools that claim to be SWS editors are no more than ontology editors, like the widely used option of OWL-S development with the OWL plug-in for Protegé-2000 [2]. This option not only suffers from all the problems enumerated above, but also adds the problem of working with an ontology instantiation, not with a SWS-like structure.

To solve these problems, we have proposed a framework, called ODESWS[3], for the design of SWS at the knowledge level, which is language-independent. This framework is based on: (1) a stack of ontologies that describe explicitly the different features of SWS; (2) a set of axioms used to check the consistency and correctness of the service descriptions; and (3) the assumption that a SWS are modeled as a problem-solving method (PSM) that describes how the service is decomposed into its components, and which is the reasoning process that describes the service execution.

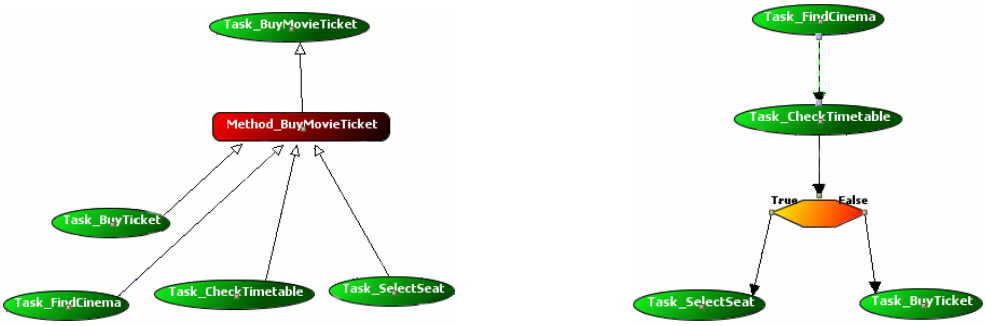


Fig. 1. A method (a) is composed of a set of sub-tasks, which are solved by other methods; and (b) defines the coordination of the execution of the sub-tasks (usually with workflows).

We also have implemented this framework, creating the ODESWS environment [3][4].

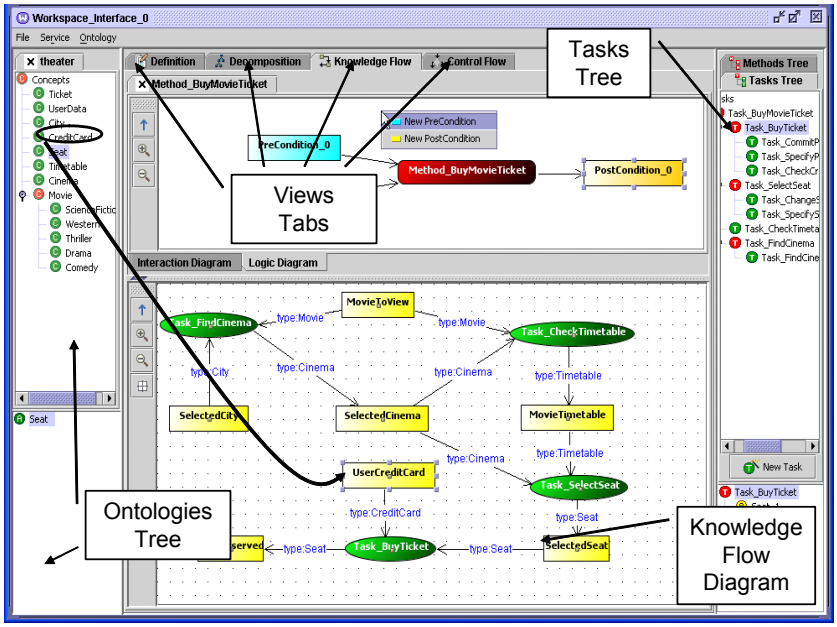
In this paper we describe SWSDesigner, the user interface of the ODESWS environment.

SWSDesigner, the ODESWS graphical interface

The design of SWSDesigner has been inspired in the classical modelling of the problem-solving methods, so it contains 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 SWSDesigner we distinguish the following general components:

- *Trees* show the hierarchy of the knowledge components needed to define the service, such as tasks, methods, and ontologies.
  - *Tasks and methods trees* (right part of Figure 2) allow users to just create the tasks and methods associated with a service, describing its functional features in the case of tasks, and internal structure in the case of methods. Once tasks and methods have been created, from these trees the user could drag the icons representing a task or method and drop them in the diagrams as needed.
  - *Ontology trees* (left part of Figure 2) show the concepts and attributes of the ontology (or ontologies) used to specify the service input/output roles. Then, the user could drag the icons representing a concept/attribute and drop them in the diagrams that enable the specification of the input/output roles of both tasks and methods.
- *Views* allow users to specify all the features of a service, and they are represented as *tabs* (see the upper part of Figure 2):
  - *Service definition View* is used to specify the functional and non-functional features of a service, containing its input/output roles (interaction diagram), pre/post-conditions (logical diagram), providers, commercial classification, geographical location, and quality rating parameter.

Fig. 2. Knowledge Flow View of a method in SWSDesginer.



- *Decomposition View* (Figure 1 a) allows users to specify the decomposition of the method (that solves the task associated with 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 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* 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* (Figure 1 b) 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-then, while-until, split and join), which are introduced through a contextual menu.

Once the user has designed the service following all the complementary views provided by the SWSDesigner, it is necessary to translate that service from the graphical representation into a semantic-oriented language such as OWL-S or WSMO. To enable this translation, the SWSDesigner invokes the SWSInstanceCreator execution, which uses the graphical model of SWS to create the instances of the

SWS description ontologies. Then, SWSTranslator is invoked to translate these instances into the language selected by the user (currently OWL-S).

Conclusions

The tools that currently enable users to design SWS depend on the capabilities of representation and reasoning of a specific SWS-oriented language. Those tools are constrained by the language expressiveness; the service must be designed using the capabilities provided by the language in which will be expressed. Furthermore, users usually introduce both errors and inconsistencies, which could be minimized using tools that operate at the knowledge level.

To solve these problems we provide tools to facilitate the design SWS in a language-independent manner. For it, we have developed the ODESWS conceptual framework and the environment that supports such framework. Once the service is completely designed using SWSDesigner it will be checked to detect inconsistencies and/or errors that could be present in the user design. If they are not detected, user will select the language in which the SWS will be expressed, and it will be automatically created.

Acknowledgements

This work has been partially financed by the Esperonto project (IST-2001-34373) and by a grant provided by the Community Autonomous of Madrid.

References

[1] Lausen, H., Felderer, M., and Roman, D., eds. (2004), "Web Service Modeling Ontology (WSMO) Editor", Available: http://www.wsmo.org/2004/d9/v01

[2] Holger Knublauch, Ray W. Fergerson, Natalya F. Noy, Mark A. Musen (2004) The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications Third International Semantic Web Conference - ISWC 2004

[3] A. Gómez-Pérez, R. González-Cabero and M. Lama (2004): ODE SWS: A framework for designing and Composing Semantic Web Services. *IEEE Intelligent Systems*, 19(4):24-31.

[4] O. Corcho, M. Fernández-López, A. Gómez-Pérez, and M. Lama (2003): An environment for Development of Semantic Web Services. Proceedings of the *IJCAI-Workshop on Ontologies and Distributed Systems*, Acapulco, Mexico.

[5] A. Gómez-Pérez, R. González-Cabero, and M. Lama (2004): Development of Semantic Web Services at the Knowledge Level. *ECOWS*. Erfurt, Germany.