WS-GEN: A Tool for the Automated Composition of Semantic Web Services*

M. Pistore, P. Bertoli, E. Cusenza, A. Marconi, and P. Traverso

University of Trento – ITC-IRST
[pistore,cusenza]@dit.unitn.it – [bertoli,marconi,traverso]@itc.it

Abstract. This demo illustrates WS-GEN, a tool supporting the automated composition of web services. WS-GEN takes as input the description of a set of available services and a "business requirement" describing the goal of the composed service, and automatically generates the executable code implementing the new composed service.

1 Introduction

One of the big challenges for the taking up of web services, and one of the main goals in the Semantic Web Services roadmap, is the provision of automated support to the composition of web services. By web service composition we mean "the automatic selection, composition, and interoperation of Web services to perform some task, given a high-level description of an objective" [3].

Currently, the problem of the composition of web services is addressed by two orthogonal efforts. From the one side, most of the major industry players propose low level process modeling and execution languages, like BPEL4WS [1]. These languages allow programmers to implement complex web services as distributed processes and to compose them in a general way. However, the definition of new processes that interact with existing ones must be done manually, and this is a hard, time consuming, and error prone task. From the other side, research within the Semantic Web community proposes a top-down unambiguous description of web services capabilities, e.g., in standard languages like OWL-S [3], thus enabling the possibility to reason about web services, and to automate web services tasks, like discovery and composition. However, the real taking up of Semantic Web Services for practical applications needs the ability of generating automatically composed services that can be directly executed, in the style of BPEL4WS programs.

In this demo we describe a tool, WS-GEN, which allows for the automated composition of web services. More precisely, the tool takes as input the description of a set of available services and a "business requirement" that defines goal of the composed web service, and automatically generates the executable process of the new service. WS-GEN currently supports two different languages for the description of the available services. They can be defined as OWL-S Process Models which provide declarative descriptions of the web service processes, or as "abstract" BPEL4WS processes which define the protocols that have to be respected in order to interact with these web services. In the first case, the "business requirement" that describes goal of the composed web service is defined in term of the semantic annotations that are part of the OWL-S model. In the second case, these semantic information, which are missing in the BPEL4WS specification, need to be defined explicitly in order to allow for the definition of the goal. In both cases, the generated web service is emitted as BPEL4WS code that can be executed on existing engines.

* The work is partially funded by the FIRB-MIUR project RBNE0195K5, "Knowledge Level Automated Software Engineering".

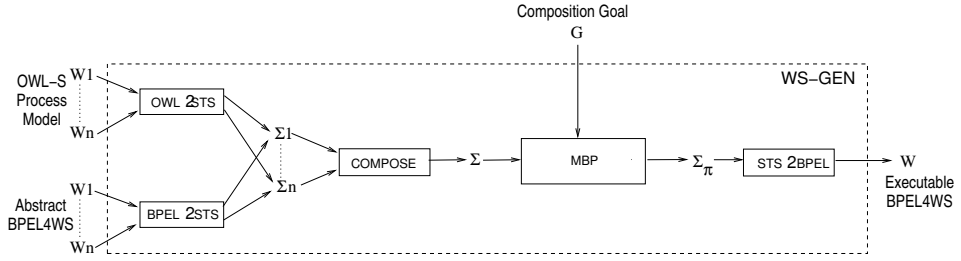

Fig. 1. Architecture of WS-GEN.

For the automated generation of the composed web service, WS-GEN exploits the AI planning approach described in [5, 6]. This approach is based on the "Planning as Model Checking" framework [4], and on the MBP tool that implements it [2]. The usage of the "Planning as Model Checking" framework is particularly relevant here, since it provides some advanced features (the possibility of managing nondeterministic domains, partial observability, and complex goal descriptions) which are necessary in web service composition. Indeed, web services exhibit nondeterministic behaviors, since the outputs of an external web service cannot be predicted prior to execution (e.g., a flight reservation service cannot know in advance whether a reservation will be confirmed or canceled). Moreover the internal status of a service (e.g., whether there are still seats available in a flight) is not available to external services, and the planner can only observe services invocations and responses. Finally, composition goals need to express complex requirements including temporal conditions (e.g., do not reserve the hotel until you have reserved the flight), and preferences among different goals (try to reserve both the flight and the hotel, but if not possible, make sure you do not reserve any of the two). All these features, which are not supported by so called "classical" planning approaches, can be captured in the "Planning as Model Checking" framework.

This demo will provide practical examples of the application of WS-GEN to the automated composition of web services. More precisely, it will show how to specify a web service composition problem starting from a set of available web services (described in OWL-S or in BPEL4WS), and it will demonstrate how the generated code for the composed web service can be executed on standard BPEL4WS engines.

2 The WS-GEN tool

WS-GEN is a tool for the automated composition of web services developed by ITC-IRST and by the University of Trento (Italy) within the ASTRO project. The tool is distributed under an OpenSource license (see http://www.opensource.org/), and is available from the project web site http://sra.itc.it/project/astro/.

WS-GEN consists of five software modules (see Figure 1). The first two modules (OWL2STS and BPEL2STS) are responsible of reading the description of an existing web service W_i (given in terms of an OWL-S Process Model or in terms of an abstract BPEL4WS specification, respectively) and of encoding it in a state transition system Σ_i. State transition systems provide a sort of operational semantics to web services. Each of them describes the corresponding web service as a state-based dynamic system, that can evolve, i.e., change state, and that can be partially controlled and observed by external agents. This way, it describes a protocol that defines how external agents can interact with the service.

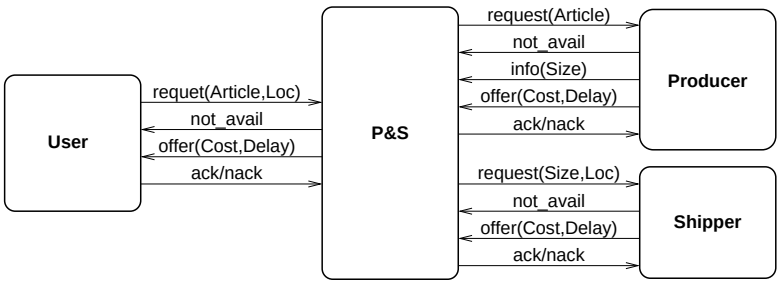

Fig. 2. A Simple Example.

From the point of view of the new composed service that has to be generated, the state transition systems corresponding to the available services constitute the environment in which the new composed service has to operate, by receiving and sending service requests. They constitute what, in planning literature, is called a planning domain, i.e., the domain where the planner has to plan for a goal. In our case, the planning domain is a state transition system Σ that combines $\Sigma_1, \ldots, \Sigma_n$. Σ represents all the possible behaviors of the available services $W_1, \ldots, W_n$ without any control performed by the service that will be generated. Module COMPOSE is responsible of composing $\Sigma_1, \ldots, \Sigma_n$ into Σ.

The fourth module is the MBP planner [2]. Given a domain Σ and a composition goal G, that imposes some requirements on the desired behavior of the planning domain, the planner generates a plan π that controls the planning domain, i.e., interacts with the external services $W_1, \ldots, W_n$ in a specific way such that the evolutions satisfy the goal G. The plan π is a state transition system that encodes the new service W that has to be generated. It dynamically receives and sends invocations from/to the external services $W_1, \ldots, W_n$, observes their behaviors, and behaves depending on responses received from the external services. If the available web services $W_1, \ldots, W_n$ are defined in abstract BPEL4WS, then the composition goal is accompanied by a set of "semantic" annotations, which are necessary to link the business requirement captured by the goal to the BPEL4WS code of the existing services. As a semantic language, OWL-S provides instead the advantageous possibility of already including these semantic annotations within the web service specifications.

The last module, STS2BPEL, is responsible of translating the state transition system π into an executable process described in BPEL4WS. This module is also responsible to provide all those information that are necessary for the deployment and execution of the generated processes.

3 A web service composition demo

The goal of the demo is to allow observing the behavior of WS-GEN during the web service composition task, as well as to test the execution of the generated web service. We will compare the two formalisms supported by WS-GEN for describing the existing services, namely OWL-S Process Models and abstract BPEL4WS, showing that the first one is more compact and better suited to the task of automated composition. In both cases, we will show how to define "business requirements" for web service composition; we will show the behavior of the different modules of WS-GEN; and we will show how the generated executable BPEL4WS code can be deployed and executed using standard execution engines like ActiveBpel.

The demo will be based on a simple example, taken from [5, 6]. It consists in providing a furniture purchase & delivery service, say the P&S service, which satisfies some user request. We

do so by combining two separate, independent, and existing services: a furniture producer Producer, and a delivery service Shipper. The idea is that of combining these two services so that the user may directly ask the composed service P&S to purchase and deliver a given article at a given place. Producer accepts requests for providing information on a given product and, if the product is available, it provides information about its size. The Producer also accepts requests for buying a given product, in which case it returns an offer with a cost and production time. This offer can be accepted or refused by the external service that has invoked the Producer. The Shipper service receives requests for transporting a product of a given size to a given location. If delivery is possible, Shipper provides a shipping offer with a cost and delivery time, which can be accepted or refused by the external service that has invoked the Shipper.

The expected protocol that the user of the P&S service will execute goes as follows. The user sends a request to get a given item at a given location, and expects either a negative answer if this is not possible, or an offer indicating the price and cost of the service. In the second case, the user may either accept or refuse the offer. Of course several interaction sequences are possible with these services; e.g., in a *nominal* scenario, none of the services answers negatively to a request; in non-nominal scenarios, unavailability of the item, user refusals or shipping service unavailability may make it impossible to reach an agreement for the purchase and delivery. Taking this into account, the business requirement for the composed service is composed of two subgoals. The "nominal" subgoal consists in reaching the agreement to purchase and delivery the item. This includes enforcing that the data communicated to the various processes is consistent with their mutual availabilities; e.g., the total service time communicated to the user cannot be less than the sum of production and delivery times. The "recovery" subgoal consists in insuring that every partner has rolled back from previous pending requests, and is only pursued when the nominal subgoal cannot be pursued anymore.

Figure 2 describes the expected data flow amongst our integrated web service, the two services composing it, and the user.

More complex examples of web service composition problems will also be used during the demo to show the practical applicability of the approach. These examples are based on realistic case studies that we are developing in projects for private companies and for the public administration.

References

1. T. Andrews, F. Curbera, H. Dolakia, J. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weeravarana. Business Process Execution Language for Web Services, 2003.
2. P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: a Model Based Planner. In *Proc. of ICAI-2001 Workshop on Planning under Uncertainty and Incomplete Information*, Seattle, USA, August 2001.
3. The OWL Services Coalition. OWL-S: Semantic Markup for Web Services. In *Technical White paper (OWL-S version 1.1)*, 2004.
4. F. Giunchiglia and P. Traverso. Planning as Model Checking. In *Proc. 5th European Conference on Planning (ECP'99)*, 1999.
5. M. Pistore, P. Bertoli, F. Barbon, D. Shaparau, and P. Traverso. Planning and Monitoring Web Service Composition. In *Proc. 11th Int. Conf, on Art. Intelligence: Methodology, Systems, Applications (AIMSA'04)*, 2004.
6. P. Traverso and M. Pistore. Automated Composition of Semantic Web Services into Executable Processes. In *Proc. 4th Int. Semantic Web Conference (ISWC'04)*, 2004.