# Semi-automatic Composition of Web Services using Semantic Descriptions

Evren Sirin[1], James Hendler[2], and Bijan Parsia[2]

[1] University of Maryland, Computer Science Department,
College Park MD 20742, USA
evren@cs.umd.edu
[2] University of Maryland, MIND Lab, 8400 Baltimore Ave,
College Park MD 20740, USA
hendler@cs.umd.edu, bparsia@isr.umd.edu

**Abstract.** As web services become more prevalent, tools will be needed to help users find, filter and integrate these services. Composing existing services to obtain new functionality will prove to be essential for both business-to-business and business-to-consumer applications. We have developed a prototype that guides a user in the dynamic composition of web services. Our semi-automatic process includes presenting matching services to the user at each step of a composition, filtering the possibilities by using semantic descriptions of the services. The generated composition is then directly executable through the WSDL grounding of the services. We tested our system by generating semantic descriptions for some of the common services available on the web such as translator, dictionary and map services. We also applied our approach to a prototype sensor network environment where each sensor provides its data as a network service.

## 1 Introduction

Web services are designed to provide interoperability between diverse applications. The platform and language independent interfaces of the web services allow the easy integration of heterogenous systems. Web languages such as Universal Description, Discovery, and Integration (UDDI) [13], Web Services Description Language (WSDL) [4] and Simple Object Access Protocol (SOAP) [14] define standards for service discovery, description and messaging protocols.

However, these web service standards do not deal with the dynamic composition of existing services. The new industry initiatives to address this issue such as Business Process Execution Language for Web Services (BPEL4WS) [5] focus on representing compositions where flow of the process and the bindings between services are known a priori. A more challenging problem is to compose services dynamically, on demand [1]. In particular, when a functionality that cannot be realized by the existing services is required, the existing services can combined together to fulfill the request.

The dynamic composition of services requires the location of services based on their capabilities and the recognition of those services that can be matched together to create a composition as described in [9]. The full automation of this process is still the object of ongoing research activity, but accomplishing this goal with a human controller as the decision mechanism can be achieved. The main problem for this goal is the gap between the concepts people use and the data computers interpret. We can overcome this barrier using Semantic Web technologies.

The Semantic Web [2] is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. This is realized by marking up Web content, its properties, and its relations, in a reasonably expressive markup language with a well-defined semantics. The Web Ontology Language (OWL) [7] is a forthcoming W3C specification for such a language which will supersede the earlier DARPA Agent Markup Language (DAML+OIL) [8]. OWL is an extension to XML and the Resource Description Framework (RDF) [3] enabling the creation of ontologies for any domain and the instantiation of these ontologies in the description of resources. The DAML-services language (DAML-S) [6] is a set of language features arranged in these ontologies to establish a framework within which the web services may be described in this semantic web context. Our work uses OWL and DAML-S to provide the semantics needed for service filtering and composition.

The remainder of this paper is organized as follows: In 2 we explain the examples of web service composition problems we address and then in section 3 we describe how semantic service descriptions are used in these examples. Section 4 describes the details of our prototype and the algorithms used in service composition. Section 5 talks about related work and we conclude in Section 6 with a discussion of possible enhancements to the system.

## 2 Motivating Examples

Our work focuses on the composition of web services that have been previously annotated with semantics and discovered by a system. As an example of composition, suppose there are two web services, an on-line language translator and a dictionary service, where the first one translates text between several language pairs and the second one returns the meanings of English words. If a user needs a *FrenchDictionary* service, neither of these can satisfy the requirement. However, together they can – the input can be translated from French to English, fed through the English Dictionary, and then translated back to French. The dynamic composition of such services is difficult using just the WSDL descriptions, since each description would designate strings as input and output, rather than the necessary concept for combining them – that is, some of these input strings must be the name of languages, others must be the strings representing user inputs and the translator's outputs. To provide the semantic concepts like *language* or *French*, we can use the ontologies provided by the Semantic Web.

Service composition can also be used in linking Web (and Semantic Web) concepts to services provided in other network-based environments. One example is the sensor network environment which includes two types of services; basic sensor services and sensor processing services. Each sensor is related to one web service which returns the sensor data as the output. Sensor processing services combine the data coming from different sensors in some way and produce a new output. These sensors have properties that describe their capabilities, such as sensitivity, range, etc., as well as some non-functional attributes, such as name, location, etc. These attributes, taken together tell whether the sensor's service is relevant for some specific task. An example task in this environment would involve retrieving data from several sensors and using relevant fusion services to process them via SOAP calls. As an example, the data from several acoustic and infrared sensors can be combined together and after applying filters and special functions, this data may be used to identify the objects in the environment. In this setting, we need to describe the services that are available for combining sensors and the attributes of the sensors that are relevant to those services. More importantly, the user needs a flexible mechanism for filtering sensor services and combining only those that can realistically be fused (for example the set representing a particular geographic area shown as a latitude/longitude box).

## 3 Creating Semantic Service Descriptions

DAML-S partitions a semantic description of a web service into three components: the service profile, process model and grounding. The *ServiceProfile* describes what the service does by specifying the input and output types, preconditions and effects. The *Process Model* describes how the service works; each service is either an AtomicProcess that is executed directly or a CompositeProcess that is a combination of other subprocesses. The *Grounding* contains the details of how an agent can access a service by specifying a communications protocol, parameters to be used in the protocol and the serialization techniques to be employed for the communication. The similarities between DAML-S and other technologies may be expressed as follows: The profile description has a similar functionality of the yellow pages in UDDI, the process model is similar to the business process model in BPEL4WS and grounding is just a mapping from DAML-S to WSDL. The main contribution of DAML-S is the ability to express the entities using the concepts defined in Semantic Web ontologies which provide expressive constructs that are suitable for the automatic discovery and composition of services.

DAML-S service descriptions are made to link to other ontologies that describe particular service types and their features. For example, an ontology can be written in OWL that is specialized for the description of sensors. This ontology contains a top level class *Sensor* to define the sensor concept. *Sensor* has subclasses such as *AcousticSensor* and *InfraRedSensor*. In the semantics of OWL, subclasses inherit the properties of superclass and may extend these attributes with additional ones.

The profile description of DAML-S services has a hierarchy as well. For example, the service provided by an *AcousticSensor* constitutes a subclass of a *Sensor* service. A profile hierarchy ontology describes this relationship and this information can be used for filtering the services that can be composed together. Each service type has non-functional attributes specific to that type. These attributes are defined via the extensible service parameter mechanism in DAML-S and are primarily used to relate the service to the its associated sensor.

## 4 Service Composition Architecture

We have developed a service composition prototype that has two basic components: a composer and an inference engine. The inference engine stores the information about known services in its Knowledge Base (KB) and has the capability to find matching services. The composer is the user interface that handles the communication between the human operator and the engine.

The inference engine is an OWL reasoner built on Prolog. Ontological information written in DAML is converted to RDF triples and loaded to the KB. The engine has built-in axioms for OWL inferencing rules. These axioms are applied to the facts in the KB to find all relevant entailments such as the class inheritance relation between two classes that may be not be directly encoded in the subclass relationships.

The composer lets the user create a workflow of services by presenting the available choices at each step. The user starts the composition process by selecting one of the services registered to the engine. A query is sent to the KB to to retrieve the information about the inputs of the service, and for each of the inputs, a new query is run to get the list of the possible services that can supply the appropriate data for this input. The composer also shows the different service classes available in the system and filters the results based on constraints which the user may specify on the attributes of a service. These functionalities are explained in detail in the following subsections.

### 4.1 Matching on Functional Properties

The composer only presents as options for composition those services whose output could be fed to a selected service as an input. The matching of two services is done using the information in the service profiles. Each service profile describes its inputs, outputs and the range of these parameters. The parameter descriptions in the profile allow defining two different types of matches between services, an exact match and a generic match. An exact match is defined between two parameters which are restricted to the same OWL class in their ServiceProfile descriptions. The services that supply an output of an exact match are more likely to be preferred in the composition and these services are displayed at the top of the matching services list.

The match between the services whose output type is a subclass of the other service's input type is called a generic match. When the output of a service is subsumed by the input, the output type can be viewed as a specialized version of the input type and these services can still be chained together. The generic matches are put at the end of the list since they are less likely to be chosen for the composition. The inference engine also orders the generic matches such that the priority of the matches are lowered when the distance between the two types in the ontology tree increases.

### 4.2 Filtering on Non-Functional Attributes

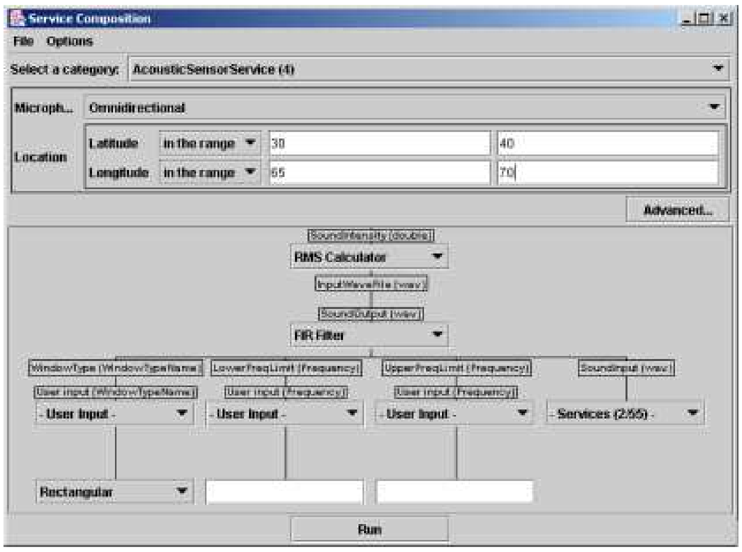
The number of services displayed in the list as possible matches can be extremely high in many cases. For example, a power grid or telephone network might have many thousands of sensors each providing several services. This will make it infeasible for someone to scroll down a list and choose one of the services simply by name. Further even if the number of services is low, the service names themselves may not be mnemonic enough to let a user know what they do, or the short text descriptions from UDDI or other services descriptions would not be enough to fully describe the services. When the name of the service does not help to distinguish the services, other non-functional attributes of the service such as location will be useful to determine the most relevant service for the current task. Thus, a sensor description, linked to a particular service, can be queried as to the sensors' locations, type, deployment date, sensitivity, etc.

In our prototype, filtering is provided based on the profile descriptions of the services. The profile hierarchies mentioned in section refdescriptions-section defines a classification which is used at the first level of filtering. Each profile subclass inherits some attributes from its container class and extends them with other attributes that apply to its category. These attributes are presented to the user and the constraints entered for these properties constitutes the second level of filtering.

Consider a example in the sensor network where we want to select a service whose input will be retrieved from a sensor service. With no other restriction, the system will present as many possible matches as the number of sensor services in the environment. If the user chooses to filter the results to the services of type *AcousticSensorService*, that decreases the number of matches significantly. The composer then queries the inference engine about the non-functional parameters of the selected service type. Based on the answer returned from the engine, the composer creates a GUI panel in which the user can enter constraints for the properties of the services as shown in Figure 1. The user's constraints are translated to Prolog queries and sent to the inference engine. The engine simply applies the new query to the previous result set and removes from consideration those services that do not satisfy the current constraints.

### 4.3 Generating Composed Services

Each composition generated by the user using the existing prototype can itself be realized as a DAML-S CompositeProcess, thus allowing it also to be advertised, discovered, and composed with other services. In the composer, we generate exactly such a CompositeProcess description, and also create the corresponding ServiceProfile with user added non-functional properties. Such a description is immediately available to the system as a named service which can be filtered



Select a category: AcousticSensorService (4)

Microph... Omnidirectional

Location: Latitude in the range 30 40; Longitude in the range 65 70

Rectangular

**Fig. 1.** Filtering is used to see only omnidirectional acoustic sensors that are located at a latitude between 30-40 and a longitude between 65-70. It is seen that only two of 55 services satisfy these constraints

and composed in the normal way. In this way, the user can quickly build up a set of complex compositions in a piece-meal fashion as the tasks at hand demand.

## 4.4 Execution of Composed Services

The current implementation of the system executes the composition by invoking each individual service and passing the data between the services according to the flow constructed by the user. This method is primarily dictated by the DAML-S and WSDL specification which both describe the web services as an interaction of either a request/response or as a notification messaging between two parties. As a consequence of this design, the client program serves as the central control authority that handles all the RPC calls to invoke individual services.

However, this centralized coordination suffers from scalability and availability problems [12]. It also can require passing redundant messages between the coordinator and other parties causing a quite inefficient use of the bandwidth which is a more severe problem when you consider the output of a such as the sensor readings of an acoustic sensor that may provide large wav files. For the efficient execution of a dynamically created composite process, we need a special framework where each node abides by a set of system rules to conduct the execution process by directly passing its result to the next service. In the prototype, we address this by adding the functionality of generating an XML workflow description that can be passed to the non-centalized system in SOAP (and forwarded as necessary) . As the standards in this area of web services are settled, it will be easy to adapt the system to the new interface.

## 5 Related Work

There are several different industry efforts to create a standards for web service composition tasks, the Business Process Execution Language for Web Services (BPEL4WS) [5] being one of the most important ones. BPEL4WS supersedes IBM's Web Services Flow Language (WSFL) and Microsoft's XLANG. BPEL4WS provides a language for the formal specification of business processes and business interaction protocols. It extends the interaction model of WSDL to define a process that provides and consumes multiple Web Service interfaces. Such a process can be thought of as composing a set of Web Services from other Web Services. However, BPEL4WS supports static binding of services in the composition, rather than discovering the possibilities on demand as implemented in the prototype described in this paper.

McIlarith and Son [10] proposed an approach to building agent technology based on the notion of generic procedures and customizing user constraints. They argue that an augmented version of the logic programming language Golog provides a natural formalism for programming Web services. These contributions are realized in their development of the ConGolog interpreter which communicates with Web services via the Open Agent Architecture (OAA) but the service and procedure ontologies are written in first-order logic. Our system more directly supports the use of existing web services by being able to ground directly in the existing WSDL or via a workflow expressed in a SOAP call, rather than creating a separate execution system for semantically described services.

The DAML-S Matchmaker [11] is a system to augment current UDDI architecture with semantic service descriptions. The matchmaker aims to improve the discovery process by allowing location of services based on their capabilities which in return will support the composition task. The basic idea used in the matchmaker is making use of the subsumption relation between the classes to find flexible matchings beyond the capabilities of UDDI. Our system would be able to incorporate matchmaker functionality if it were available online (perhaps as a service itself), but currently assumes the service definitions are available at runtime. That is, we use filtering on a set of previously discovered services, rather than dynamic matchmaking and loading. This should be more scalable, as it will allow service "crawlers" to find semantic service descriptions, and support all usable compositions among the services found.

## 6 Conclusion and Future Work

In this work, we have shown how to use semantic descriptions to aid in the composition of web services. We have developed a prototype system and shown that it can compose the actual web services deployed on the internet as well as providing filtering capabilities where a large number of similar services may be available. Our prototype is the first system to directly combine the DAML-S semantic service descriptions with actual invocations of the WSDL descriptions allowing us to execute the composed services on the Web.

As a future work, we are working on the incorporation of planning technology in the inference engine that would result in further automation of the system. We are also investigating the possibility of learning from past user activity. Generating richer ontologies with more specific descriptions will also improve the performance of the engine. As ontologies become widely used on the Semantic Web, we expect to find an increasing number of cross references between related concepts in different ontologies (and OWL supports such crossreferencing directly) and thus the impact of semantic information will become more apparent.

## References

1. B. Benatallah, M. Dumas, M.-C. Fauvet, and F. Rabhi. Towards Patterns of Web Services Composition. In S. Gorlatch and F. Rabhi, editors, *Patterns and Skeletons for Parallel and Distributed Computing*. Springer Verlag, UK, Nov 2002.
2. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.
3. D. Brickley and R. Guha. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation submitted 22 February 1999 http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/. (current May 2002).
4. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1, 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.
5. F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.0, July 2001. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.
6. DAML Services Coalition. DAML-: Web Service Description for the Semantic Web. In *The First International Semantic Web Conference (ISWC)*, June 2002.
7. M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Web Ontology Language (OWL) Reference Version 1.0. W3C Working Draft 12 November 2002 http://www.w3.org/TR/2002/WD-owl-ref-20021112/.
8. I. Horrocks, F. van Harmelen, P. Patel-Schneider, T. Berners-Lee, D. Brickley, D. Connoly, M. Dean, S. Decker, D. Fensel, P. Hayes, J. Heflin, J. Hendler, O. Lassila, D. McGuinness, and L. A. Stein. DAML+OIL, 2001. http://www.daml.org/2001/03/daml+oil-index.html.
9. Z. M. Mao, E. A. Brewer, and R. H. Katz. Fault-tolerant, Scalable, Wide-Area Internet Service Composition. U.C. Berkeley TR UCB//CSD-01-1129, Jan 2001.
10. S. McIlraith and T. Son. Adapting Golog for Composition of Semantic Web Services. In *Conference on Knowledge Representation and Reasoning*, April 2002.
11. M. Paolucci, T. Kawmura, T. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In *The First International Semantic Web Conference*, 2002.
12. Q. Z. Sheng, B. Benatallah, M. Dumas, and E. O.-Y. Mak. SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment. In Demo Session of the 28th Intl. Conf. on Very Large Databases, Sept 2002.
13. UDDI. The UDDI technical white paper, 2000. http://www.uddi.org/.

14. W3C. SOAP 1.2 Working draft, 2001. http://www.w3c.org/TR/2001/WD-soap12-part0-20011217/.