Discovering Service Compositions That Feature a Desired Behaviour*

Fabrizio Benigni, Antonio Brogi, and Sara Corfini

Department of Computer Science, University of Pisa, Italy
{benigni,brogi,corfini}@di.unipi.it

Abstract. Web service discovery is one of the key issues in the emerging area of Service-oriented Computing. In this paper, we present a complete composition-oriented, ontology-based methodology for discovering semantic Web services, which exploits functional and behavioural properties contained in OWL-S service advertisements to satisfy *functional* and *behavioural* client queries. To this aim, we build on top of the results contained in two recent articles, where we presented (1) a suitable data structure (viz., a dependency hypergraph) to collect functional information of services, and (2) a suitable notion of behavioural equivalence for Web services. We also discuss the architecture and the main implementation choices of the matchmaking system applying such a methodology.

1 Introduction

Service-oriented Computing (SoC) [1] is emerging as a new promising computing paradigm that centers on the notion of service as the fundamental element for developing distributed software applications. In this setting, Web service discovery is a major issue of SoC, as it allows developers to find and re-use existing services to rapidly build complex applications.

The standard service description language (WSDL) provides services with purely syntactic descriptions, not including neither behavioural information on the possible interaction among services, nor semantics information to describe the functionality of services. Yet, both behavioural and semantic information may be necessary, for example, to satisfy complex queries that require to compose the functionalities offered by different services, as well as to automatise the processes of service discovery and composition.

During the last years, various proposals have been put forward to feature more expressive service descriptions that include both semantics (viz., ontology-based) and behaviour information about services. One of the major efforts in this direction is OWL-S [2], a high-level ontology-based language for describing services. In particular, OWL-S service descriptions include a list of semantically annotated functional attributes of services (the *service profile*), and a declaration of the interaction behaviour of services (the so-called *process model*).

* Research partially supported by EU FP6-IST STREP 0333563 SMEPP and MIUR FIRB TOCAI.IT.

B. Krämer, K.-J. Lin, and P. Narasimhan (Eds.): ICSOC 2007, LNCS 4749, pp. 56–68, 2007.
© Springer-Verlag Berlin Heidelberg 2007

In this paper, we present a composition-oriented, ontology-based methodology for discovering OWL-S described services. In particular, we employ semantic information to select available services that can be exploited to satisfy a given query, and we employ behaviour information to suitably compose such services to achieve the desired result.

The methodology integrates the results recently presented in [3,4]. In [3] a suitable data structure (viz., a dependency hypergraph) to collect relationships among ontology-annotated inputs and outputs of services (i.e., semantic information) was introduced. It is important to stress that the construction of such a hypergraph does not affect the query answering time, as it is built off-line and updated whenever a new service is added to the local service repository. In [4] we defined a suitable notion of behavioural equivalence for Web services. Such a notion allows to establish whether two services, described by means of a simple variant of standard Petri nets, are behaviourally equivalent, i.e., such that an external observer can not tell them apart. An interesting feature of this methodology is the ability of addressing both *functional* and *behavioural* queries, i.e., respectively, queries specifying the functional attributes of the desired service, and queries also requiring a specific behaviour of the service to be found. In particular, in case of a behavioural query, the methodology – besides satisfying the query functional requirements – guarantees that the returned service features the desired behaviour.

In this paper we also present a system – called SAM, for Service Aggregation Matchmaking – implementing the discovery methodology here introduced. The main features of the new version of SAM can be summarised as follows:

- *Composition-oriented matching* – that is, the capability of discovering service compositions. When no single service can satisfy the client query, SAM checks whether the query can be fulfilled by a suitable composition of services.
- *Ontology-based matching* – that is, the ability of "crossing" different ontologies and performing flexible matching automatically. Given that different services are typically described in terms of different ontologies, SAM determines relationships between concepts defined in separate ontologies, so to establish functional dependencies among services.
- *Behaviour-aware matching* – that is, the ability of guaranteeing behavioural properties. Given a query synthetising the behaviour of a service, SAM searches for (compositions of) services which are behaviourally equivalent to the query. Each matched service (composition) can be used interchangeably with the service described by the query.

It is also worth observing that, with respect to its first version described in [5], SAM is now capable of properly coping with the problem of "crossing" different ontologies (thanks to the introduction of the hypergraph), as well as of suitably addressing behavioural queries.

The rest of the paper is organized as follows. Section 2 describes the composition-oriented, ontology-based methodology for discovering services. Section 3 is devoted to discuss the architecture, the main implementation choices, and

possible future extensions of the system applying such a methodology. Finally, some concluding remarks are drawn in Section 4.

2 A Methodology for a Composition-Oriented Discovery

In this Section, we present a methodology for discovering compositions of semantic Web services which takes into account both *semantic* and *behavioural* information advertised in the OWL-S service descriptions. In particular, we employ "semantics", namely all those ontological information regarding the functional attributes (i.e., inputs and outputs) of services, to select services with respect to "what they really do", and we employ "behaviour", namely, information concerning the order with which messages can be received or sent by each service, to guarantee some useful properties of selected services. Before presenting the discovery methodology, we describe hereafter the data structures and formalisms we employ to summarise service descriptions.

2.1 The Internal Representation of Services

As briefly mentioned in the Introduction, the complete behaviour of a service is described by the OWL-S process model, which may include conditional and iterative constructs. Hence, a service may behave in different ways and feature different functionalities. We say that a service may have different *profiles*, each of them requiring/providing different inputs/outputs. Hence, as one may expect, we represent each service with two distinct items: a *set of profiles*, to summarise the different sets of functional attributes employed by each profile of the service, and a *Petri net*, to model the whole service behaviour.

More precisely, a profile S_n represents a dependency between the set of the inputs and the set of the outputs employed by the specific behaviour n of a service S. Service profiles are collected into a hypergraph, whose nodes correspond to the functional attributes of the service profiles, and whose hyperedges represent relationships among such attributes. It is worth observing that each node v of the hypergraph, that is, each functional attribute, is associated with a concept, which is defined in one of the ontologies referred by the service employing v. The hypergraph also includes equivalent and sub-concept relationships among nodes, viz., among ontology concepts. (A formal definition of the hypergraph and the algorithms for its construction can be found in [3,6].)

Example. Let us consider the simple service T, defined as a choice of two atomic operations. The former inputs a *zipCode* (*Z*) and returns the corresponding *geographicCoordinates* (*GC*), and the latter inputs a *location* (*L*) and a *date* (*D*), and returns the computed *wheatherInformation* (*W*). The service T has hence two profiles, T_1 and T_2, represented by the hyperedges $\{Z\} \xrightarrow{T_1} \{GC\}$ and $\{L, D\} \xrightarrow{T_2} \{W\}$, respectively. Consider next the service S, which inputs a *city* (*C*) and a *nation* (*N*), and returns the corresponding *zipCode* (*Z*). Service S exposes a single profile S_1 represented by the hyperedge $\{C, N\} \xrightarrow{S_1} \{Z\}$. The

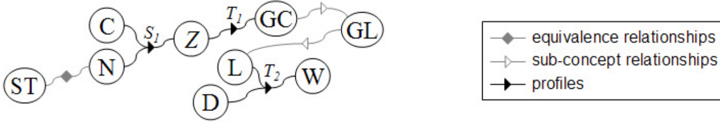

Fig. 1. A simple hypergraph

hypergraph including profiles S_1, T_1, T_2 is illustrated in Figure 1. Note the equivalent relationship linking together *state* (ST) and *nation* (viz., ST and N are synonyms), as well as, the sub-concept relationship linking *geographicCoordinates* and *geographicLocation* (GL) (viz., GC is a sub-concept of GL).

While a profile describes a particular behaviour of a service from a functional point of view, the complete interaction behaviour of a service is represented by an OCPR net. OCPR nets (for Open Consume-Produce-Read nets) [4] are a simple variant of the standard Condition/Event Petri nets, designed to naturally model the behaviour of services, and in particular the persistency of data (i.e., once a data has been produced by some service operation, it remains available for all the service operations that input it). Briefly, an OCPR net is equipped with two disjoint sets of places, namely, control and data places, to properly model the control flow and the data flow of a Web service, and with an interface, which establishes those data places that can be observed externally. Hence, whilst control places can be produced and consumed, data places can be read, produced but not consumed. We formally defined OCPR nets in [4], where a mapping from OWL-S process models to OCPR nets is also presented. Intuitively speaking (see [4] for details), transitions map (OWL-S) atomic operations, while data and control places respectively model the availability of data and the executability of atomic operations. It is worth observing that when a service is translated into an OCPR net, all the data places of the net are externally observable by default.

Example. Figure 2 (which will be explained in more detail later) illustrates four simple OCPR nets, where rectangles, circles and diamonds respectively represent transitions, data places and control places. The initial control place i as well as the final control place f of each net are emphasised in light gray. Furthermore, each net is delimited by a box which represents the net interface, namely, the set of places which can interact with the environment. Hence, those data places that lie on the box are the ones that can be observed externally.

2.2 Discovering Compositions of Services

So far, we have introduced the internal representation of services that we use to store them in a local repository. We can now propose a complete composition-oriented methodology for discovering services. The methodology takes as input the so-called *behavioural queries*, that is, queries specifying both the inputs and outputs, as well as the expected behaviour of the service to be found. A behavioural query, for example, can be expressed in terms of the OWL-S process

model describing the desired service. The set of the functional attributes of the query can be easily retrieved by its OWL-S process model, which can be in turn suitably translated into an OCPR net [4]. Hence, we can assume that a query consists of two parts: a couple (I, O) and an OCPR net, respectively describing the set of the inputs and outputs, and the behaviour of the service to be found.

The discovery methodology we are going to propose consists of two main phases: a *functional analysis* and a *behavioural analysis*, that we describe below.

Functional Analysis

This first phase consists in a sort of *functional filter*, indeed, services are selected according to their functional attributes only. More precisely, the functional analysis focuses on the first functional part of the query (viz., the couple (I, O) of inputs and outputs), and returns those set of services which satisfy the functional requirements of the query. Hence, for each set of services S passing the functional filter: (1) all the query outputs are provided by the services in S, (2) all the inputs of the services in S are provided by the query (or they can be produced by some service in S).

As described in the previous subsection, we summarise functional information of the services stored in the repository in an hypergraph. The functional analysis hence consists of a visit of the hypergraph. It is worth noting that by exploring *profiles*, we address the discovery of sets of services, as well as by exploring *sub-concept* and *equivalent* relationships we properly reason with (different) ontologies. In particular, the functional analysis explores the hypergraph starting from those nodes corresponding to the query outputs, and it continues by visiting backwards the hyperedges until reaching, if possible, the query inputs. The profile-labelled hyperedges which take part in an hyperpath from the query outputs to the query inputs determine a set of service profiles satisfying the query. A detailed discussion of the algorithm for visiting the hypergraph can be found in [6]. Furthermore, it is also worth noting that we have enriched the functional analysis with a *minimality check* [5], in order to avoid constructing non-minimal sets of service, that is, sets containing (at least) a service not strictly necessary to satisfy the query.

Example. Consider the simple hypergraph illustrated in Figure 1, and the query taking as input a *city* (C) and a *state* (ST) and providing as output the corresponding *geographicLocation* (GL). The functional analysis visits the hypergraph starting from the query output GL. Then, by exploring the sub-concept relationship $\{GC\} \rightarrow \{GL\}$ and the profile T_1, it reaches the node Z. Next, by visiting the profile S_1 and by crossing the equivalence relationship $\{ST\} \rightarrow \{N\}$, it reaches both the query inputs $\{C, ST\}$. Hence, the set of profiles $\{S_1, T_1\}$ satisfies (the functional requirements of) the query.

Behavioural Analysis

As previously described, every set of profiles determined by the functional analysis satisfies the query from a functional perspective. Consider now a specific set P of profiles. The behavioural analysis checks whether the services included in the

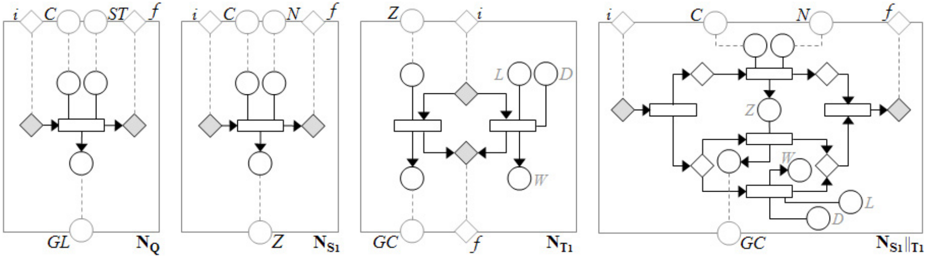

Fig. 2. OCPR nets

set, and suitably composed together, are *behaviourally* equivalent to the client query. The behavioural analysis consists of the following two main steps.

(1) Constructing the composite service. The objective of this step is to construct the OCPR net modelling the parallel composition of the service profiles included in P. Firstly, for each service profile S_n in the set, we retrieve from the local repository the OCPR net modelling the complete behaviour of the corresponding service S. As described in subsection 2.1, all the data places of an OCPR net are externally observable by default (viz., all the data places of the net belong to the net interface). Yet, a profile identifies a specific portion (i.e., behaviour) of the service, which may partially employ the inputs and outputs of the whole service. Hence, let S_n be a profile of a service S, and let N_S be the OCPR net modelling S. Then, we remove from the interface of N_S those data places which do not belong to the inputs and outputs of the profile S_n.

We can now construct the composite OCPR net $N_{\|}$ modelling the parallel composition of the OCPR nets $N_{S_1}, \ldots, N_{S_n}$ associated to the profiles $S_1, \ldots, S_n$ belonging to the set P. Note that, if the set of profiles returned by the functional analysis contains n profiles of the same service S, we insert into the composite net n copies of the OCPR net modelling S, each of them typically providing a different interface. In other words, we are considering multiple executions of the same service.

As stated before, the data places which belong to the net interface are the only ones that can interact with the external environment. Consequently, in order to compose OCPR nets, we have to operate on their interfaces. To build $N_{\|}$ we first perform the disjoint union of the transitions, data places and control places of the nets $N_{S_1}, \ldots, N_{S_n}$. Next, we collapse those data places which are equivalent and which occur in the interfaces of $N_{S_1}, \ldots, N_{S_n}$. It is worth noting that we qualify as equivalent data places, those data places which are syntactically and/or semantically equivalent. For example, we collapse two data places corresponding to two syntactically different, yet synonyms concepts. Moreover, in order to perform the parallel composition of $N_{S_1}, \ldots, N_{S_n}$, we add to $N_{\|}$ the necessary additional transitions and control places, according to the OCPR mapping of the parallel composition (viz., the OWL-S `split+join` construct) given in [4]. It is important to observe that the initial control places as well as the final control place of an OCPR net are externally observable by default.

The interface of the resulting composite net $N_{\|}$ is the union of the interfaces of the nets $N_{S_1}, \ldots, N_{S_n}$. Finally, before verifying the equivalence of the composite net with the behavioural query, we have to properly revise the interface of the composite net. Indeed, the interface of $N_{\|}$ may contain some data places with do not belong to the interface of the query net. We do not need to observe those data places, which, hence, have to be removed from the interface of $N_{\|}$.

Example. Let us continue the example previously introduced. For each profile included in the set $\{S_1, T_1\}$ returned by the functional analysis, we consider its OCPR net representation. The OCPR nets N_{S_1}, N_{T_1}, respectively representing the profiles S_1, T_1, are illustrated in Figure 2. While all the data places of N_{S_1} belong to the net interface (as they belong to the single profile S_1 of S), the interface of N_{T_1} contains only the data places employed by the profile T_1. We perform next the parallel composition of the two nets N_{S_1} and N_{T_1}. The resulting net $N_{S_1 \| T_1}$ is depicted in the right part of Figure 2. Finally, note that we removed Z from the interface of $N_{S_1 \| T_1}$, since it does not belong to the query.

(2) Analysing the service behaviour. The second step of the behavioural analysis checks whether the composition of those services previously selected during the functional analysis is capable of satisfying the query from a behavioural perspective. Let N_Q denote the net representing the behavioural query. Namely, this step checks whether N_Q and $N_{\|}$ are equivalent, that is, whether they are externally indistinguishable.

To this end, we defined in [4] a suitable notion of behavioural equivalence for Web services, which features *weakness*, as it equates structurally different yet externally indistinguishable services; *compositionality*, as it is also a congruence; and *decidability*, as the set of states that an OCPR net can reach is finite. More precisely, a state of an OCPR net is the marking of its observable places. In the initial state only the initial control place contains a token, while all the other places belonging to the net interface are empty. Then, in each state, an OCPR net can execute two types of actions, namely, it can put a token in one of the data places of its interface, or it can perform τ-transitions (i.e., it can fire transitions not requiring any additional token). Hence, intuitively speaking, in order to verify whether N_Q and $N_{\|}$ are equivalent, the second step of the behavioural analysis checks whether for each state s of N_Q: (1) there exists a state t of $N_{\|}$ which can perform all the actions executable by s; (2) for each state s' reachable from s by executing the action a, t can reach a state t' by executing the same action a, such that s' and t' are equivalent. It is important to observe that if s reaches s' by performing a single τ-transition, t can reach a state t' equivalent to s' with one or more τ-transitions. Dually, this step checks whether similar conditions hold for each state of $N_{\|}$. If so, the query and the composite net are equivalent, that is, the found service composition fully satisfies the query.

Example. Consider the nets N_Q and $N_{S_1 \| T_1}$, illustrated in Figure 2, and respectively representing the client query and the previously built composite service. According to [4], the nets N_Q and $N_{S_1 \| T_1}$ are equivalent. In particular, note that if we add a token in C and ST (namely, N, since ST and N are equivalent), N_Q

reaches the final state in a single τ-transition, while $N_{S_1 \| T_1}$ needs of performing four τ-transitions.

3 Implementation of the Methodology

We discuss below the architecture and the main implementation choices of the system (viz., SAM) applying the discovery methodology described in Section 2.

Architecture
Figure 3 illustrates the overall architecture of the matchmaking system implementing the proposed discovery methodology. The system – available as Web service – is designed to cope with two classes of users, *clients* and *providers*, which, mainly, can query the system, and add a new service to the system, as reflected by the WSDL interface depicted in Figure 3.

The client queries are handled by the *search engine* core component, which consists of two building blocks, namely, the *functional analyser* and the *behavioural analyser*, respectively implementing the functional analysis and the behavioural analysis of the discovery methodology described in subsection 2.2. In particular, functional queries can be satisfied by the functional analyser only, while behavioural queries need of the joint work of both functional and behavioural analysers. It is worth noting that the implementation of the *behavioural analyser* makes use of the algorithm presented by Fernandez and Mounier in [7] for verifying the bisimilarity of two systems. Clients can also list available services and ontologies: the *service browser* component satisfies these requests.

When a provider adds a new service to the system, the *hypergraph builder* and the *OWL-S2PNML* components translate the service into the internal representation described in subsection 2.1. The hypergraph builder determines the profiles of the service. Moreover, it exploits SemFiT [8], a tool for "crossing" different ontologies, to establish the semantic (viz., equivalence and sub-concept) relationships among ontology concepts. In particular, the hypergraph builder determines the relationships concerning those concepts defined in the new ontologies employed by the service to be added, and those concepts which belong to the ontologies previously registered to the system. A more detailed description of the behaviour of the hypergraph builder component is available in [6].

OWL-S2PNML translates the OWL-S process model of the service into an OCPR net, which is described by a corresponding PNML file. The Petri Net Markup Language[1] (PNML) is a XML-based and customizable interchange format for Petri nets. We employ PNML in order to enhance the modularity and portability of the system. Providers can also add single ontologies: in such a case, the hypergraph builder component suffices to update the hypergraph.

It is also worth noting that, before adding a new service as well as a new ontology, a provider has to login the system. The authentication service is managed by the *account manager* component.

[1] `http://www2.informatik.hu-berlin.de/top/pnml/about.html`

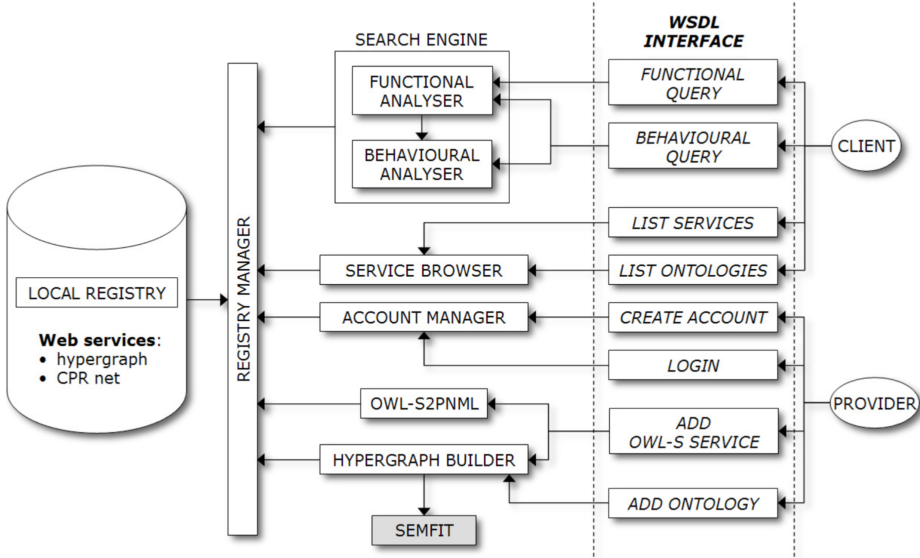

Fig. 3. System architecture

Service internal descriptions (i.e., profiles and semantic relationships, and PNML files), ontologies and account information are stored in the *local registry*. All system components can access the local registry by means of a suitable *registry manager*.

Implementation

We discuss hereafter the main implementation aspects of SAM, the matchmaking system previously described in Section 2. In particular, the implementation of SAM has been conditioned by the following requisites.

- *Portability* – the system consists of Java packages, each of them wrapped in a standard Java EE 5 component. SAM is accessible as a Web service, described by a standard WSDL interface as well as by an OWL-S advertisement.
- *Extensibility* – SAM is deployed as a multitiered Java enterprise application, which allows for high levels of modularization and ease of substitute/add logic components (e.g., the integration with SemFiT, remotely accessed by its WSDL interface). Furthermore, the use of Java language allows us to employ many existing Java libraries and tools (e.g., OWL-S parsers).
- *Scalability* – Java EE platform natively guarantees suitable performance and scalability to component-based and multitiered applications.
- *Use of standards* – Besides the use of Java EE platform, the system implementation relies on other standard languages and well-known technologies:
  - PNML, to describe OCPR nets by means of standard XML files,
  - javaDB, to deploy the database (which is accessible via JDBC API directly by the Java EE component containers),

- Mindswap OWL-S API, to validate, marshal/unmarshal OWL-S descriptions,
- PNML framework API, to marshal/unmarshal PNML descriptions.

The domain logic of SAM is implemented by three Java libraries: SamFeedLogics, which implements the *OWL-S2PNML* and *hypergraph builder* components of Figure 3, SamFunctionalLogics, that implements the *functional analyser* component, and SamBehaviouralLogics, which implements the *behavioural analyser* component. It is worth noting that each library is connected to the rest of the architecture by facade EJB (Enterprise Java Beans) components, that automatically retrieve other components' references by Java EE server injection. Hence, each functional component is totally independent from the overall architecture, and it can be tested in a Java SE 5 environment by employing suitable stubs and drivers. Furthermore, facade components declaratively instruct the application server (by means of Java 5 annotations included in the class files) to expose relevant methods as (WSDL described) Web services.
The implementation of SAM is completed by the following Java EE components.

- SamPersistence – which, by abstracting from the actual data representation, provides two interfaces to respectively view and modify (only upon authorization) the data contained in the local registry of SAM.
- SamDBBrowser – which implements a simple database browsing tool (viz., the *service browser* component).
- SamAccountMgmt – which grants (or denies) access to SamFeedLogics component. In particular, the current security management allows only registered users to submit new OWL-S descriptions and new ontologies. Moreover, it keeps trace of every submission to discourage any abuse. Future improvements of security management may be implemented in order to prevent possible leaks in quality of service.
- SamGWTServlet – which provides SAM with a friendly Web interface, developed with the Google Web Toolkit.

Extending the Implementation
For testing the system concisely presented in this section, we manually produced several OWL-S service descriptions. Although the behaviour of each single part of the system has been properly checked, currently, we are not able to provide a serious experimental assessment of the system. Please, note that no standard test collection for OWL-S service retrieval does exist, yet. To be more exact, by accurately scanning the Web, we found two OWL-S repositories[2], both generated with semi-automatic WSDL annotators. However, as one may expect, WSDL annotators generate very simple OWL-S process models, typically a list of atomic operations, as no behavioural information is available in WSDL descriptions. Obviously, such process models are not useful for testing the system.

To overcome this problem, a brute force (time-consuming and error-prone) solution could be to manually create a test collection of OWL-S services. Yet, given

[2] http://moguntia.ucd.ie/repository/owl-ds.html
http://www-ags.dfki.uni-sb.de/ klusch/owls-mx/index.html

that WS-BPEL [9] has been recently approved as an OASIS standard, we plan to extend our system in order to cope with WS-BPEL services. For instance, a possible solution is to translate BPEL processes into OWL-S services, by plugging into the system the BPEL*2*OWL-S [3] translator developed by Aslam et al. in [10]. Yet, there is still a prominent problem, namely, the lack of ontological information in WS-BPEL descriptions. Hence, when a provider adds a WS-BPEL process into the system, firstly, the WS-BPEL process is translated into a "rough" OWL-S service, and secondly, the provider is asked to complete the OWL-S description, by annotating the service parameters with ontology concepts (e.g., by employing some friendly ontology editor, such as Protégé (`http://protege.stanford.edu`). The new OWL-S service can be then registered into the system as described in the previous subsections.

Finally, given the high computational (viz., exponential) complexity of the functional and behavioural analysers, another important line for future work is to develop indexing and/or ranking techniques (as search engines do for Web pages) in order to sensibly improve the efficiency of the discovery methodology.

4 Concluding Remarks

In this paper, we have introduced an automated composition-oriented, ontology-based methodology for discovering (semantic) Web services. We have also presented a matchmaking system, called SAM, which prototypically implements such a methodology. SAM is the first matchmaker – at the best of our knowledge – that takes properly into account functional, semantic and behaviour information provided by service descriptions. More precisely, given as input a query specifying inputs, outputs and expected behaviour of the service to be found (viz., a *behavioural* query), SAM returns an ordered list of (compositions of) services, each of them (1) requiring as input a subset of the query inputs, (2) providing as output all the query outputs (or more), (3) featuring a behaviour equivalent to the query. In particular, it is important to observe that feature (3) makes our system suitable to be employed to address emerging issues of the Service-oriented Computing area, such as service replaceability.

Recently, automatic matchmaking of Web services has gained prominent importance and new approaches are frequently introduced. For the lack of space, we briefly discuss hereafter only some of the widely known approaches.

The first effort towards the automation of Web service discovery has been put forward by some of the authors of OWL-S in [11]. Their algorithm performs a functionality matching between service requests and service advertisements described as DAML-S (the predecessor of OWL-S) service profiles. This approach was the first at introducing the notion of an automatic and flexible matching by suitably considering subsumes and plug-in relationships among the ontology-annotated attributes of services and service requests. Yet, the algorithm proposed in [11] does not deal with the ontology crossing problem, that is, it is not able to determine relationships between attributes annotated with concepts of separate

[3] `http://bpel4ws2owls.sourceforge.net`

ontologies. To this aim, it is worth mentioning the service matchmaking approach presented by Klusch et al. in [12], which employs both logic based reasoning and IR techniques to properly relate concepts of different ontologies.

A common drawback of [11,12] is that they search for a *single* service capable of satisfying a client query by itself. However, as previously described, composing functionalities of different services may be necessary to satisfy a query. An approach to a composition-oriented discovery is presented by Benatallah et al. in [13], where the matchmaking problem is reduced to a best covering problem in the domain of hypergraph theory.

With respect to the approach presented in this paper, it is important to stress that none of the mentioned proposals takes into account behavioural aspects of services. Indeed, our matchmaker – differently from [11,13,12] – is capable of solving behavioural queries, guaranteeing, as well, some behavioural properties of the returned (compositions of) services.

Behavioural aspects of services are partially taken into account by the approach of Agarwad and Studer, that proposed in [14] a new specification of Web services, based on description login and π-calculus. Their algorithm consider semantic and temporal properties of services, yet, their matchmaking approach is limited to a single service discovery.

References

1. Papazoglou, M.P., Georgakopoulos, D.: Service-Oriented Computing. Communications of the ACM 46(10), 24–28 (2003)
2. OWL-S Coalition: OWL-S: Semantic Markup for Web Service (2004), http://www.ai.sri.com/daml/services/owl-s/1.2/overview/
3. Brogi, A., Corfini, S., Aldana, J., Navas, I.: Automated Discovery of Compositions of Services Described with Separate Ontologies. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 509–514. Springer, Heidelberg (2006)
4. Bonchi, F., Brogi, A., Corfini, S., Gadducci, F.: A behavioural congruence for Web services. In: Arbab, F., Sarjani, M. (eds.) Fundamentals of Software Engineering. LNCS, Springer, Heidelberg (2007) (to appear)
5. Brogi, A., Corfini, S.: Behaviour-aware discovery of Web service compositions. International Journal of Web Services Research 4(3) (2007) (to appear)
6. Brogi, A., Corfini, S., Aldana, J., Navas, I.: A Prototype fot Discovering Compositions of Semantic Web Services. In: Tumarello, G., Bouquet, P., Signore, O. (eds.) Proc. of the 3rd Italian Semantic Web Workshop (2006)
7. Fernandez, J.C., Mounier, L.: "On the Fly" verification of behavioural equivalences and preorders. In: Larsen, K.G., Skou, A. (eds.) CAV 1991. LNCS, vol. 575, pp. 181–191. Springer, Heidelberg (1992)
8. Navas, I., Sanz, I., Aldana, J., Berlanga, R.: Automatic Generation of Semantic Fields for Resource Discovery in the Semantic Web. In: Andersen, K.V., Debenham, J., Wagner, R. (eds.) DEXA 2005. LNCS, vol. 3588, pp. 706–715. Springer, Heidelberg (2005)
9. BPEL Coalition: WS-BPEL 2.0 (2006), http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

10. Aslam, M.A., Auer, S., Shen, J., Herrmann, M.: Expressing Business Process Models as OWL-S Ontologies. In: Eder, J., Dustdar, S. (eds.) Business Process Management Workshops. LNCS, vol. 4103, pp. 400–415. Springer, Heidelberg (2006)
11. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic Matchmaking of Web Services Capabilities. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 333–347. Springer, Heidelberg (2002)
12. Klusch, M., Fries, B., Sycara, K.: Automated semantic web service discovery with OWLS-MX. In: AAMAS'06, pp. 915–922. ACM Press, New York (2006)
13. Benatallah, B., Hacid, M.S., Léger, A., Rey, C., Toumani, F.: On automating Web services discovery. VLDB J. 14(1), 84–96 (2005)
14. Agarwal, S., Studer, R.: Automatic Matchmaking of Web Services. In: IEEE Int. Conference on Web Services, pp. 45–54. IEEE Computer Society Press, Los Alamitos (2006)