An ontology-based architecture for service discovery and advice system*

D. Bianchini, V. De Antonellis and M. Melchiori
Università di Brescia
Dip. Elettronica per l’Automazione
Via Branze, 38
25123 Brescia - Italy
{bianchin,deantone,melchior}@ing.unibs.it

Abstract

In this paper we present a novel, ontology-based approach to service discovery, which exploits domain knowledge and semantic service descriptions to guide the service discovery process and provide advice on service selection and instantiation in interoperable adaptive information systems. The proposed system architecture for service discovery and advice has the advantage of providing specific advice at multiple levels of granularity during the service composition process. At the highest level, the system can help to determine what kind of abstract service is required against a contextual functional request. Once all the services that can fulfill the required function are discovered, the advice system can recommend an appropriate concrete service, taking into account both problem characteristics and quality considerations. More specialized, in-depth advice can also be given, for example, on how to initialize and configure the parameters of a service. The approach and prototype have been proposed to demonstrate practical benefits in the framework of the MAIS (Multi-channel Adaptive Information Systems) project [9].

1 Introduction

Service oriented technologies are used to provide application interoperability by supporting sharing and coordinated use of digital resources in distributed, dynamic scenarios. For service description, storing and discovery, a set of industrial standard protocols such as UDDI and WSDL have been provided. For service composition, several industrial proposals such as WSFL, XLANG and BPEL4WS have been developed. Main limitations of these industrial standards are due to the fact that service semantic aspects are not considered and service composition is strictly predefined, i.e. the flow of the process and the bindings between the services are known a priori. Recently the Semantic Web technologies have been used to provide semantic service descriptions by means of ontologies using ontology description languages such as OWL [5] and OWL-S [4]. OWL is based on the $\mathcal{S H O I N}$ Description Logics and a DL reasoner (for example, RACER) can be used to semantically compare descriptions written in OWL. OWL-S is a service ontology specified in OWL. It describes a service by defining a Service Profile (what the service does), a Service Model (how the service works with the detailed description of its constituent processes) and a Service Grounding (how to use the service, i.e., how a client can actually invoke the service). In particular, OWL-S provides a grounding to WSDL.

Several approaches in literature address the problem of ontology-based service discovery. In general, the elements used for service capability representation refer to concepts properly defined in domain ontologies. Semantic relationships between concepts are exploited to enhance matchmaking between advertisements and requests. Kawamura et al. [8] use a service capability representation based on the OWL-S Service Profile, with inputs, outputs, preconditions and effects. Semantic relationships between input and output parameters in the domain ontologies are exploited to verify if: (a) all the outputs required are also provided by the advertisement and (b) all the inputs needed for advertisement execution are provided by the requester. A logical formalism based on Horn clauses is also provided to express preconditions and effects and to verify their logical implication or equivalence. Horrocks et al. [7] express the service description suggested by OWL-S Service Profile by means of Description Logics and exploit inference mechanisms of a DL reasoner to establish what degree of match exists between advertisements and requests. Wang et al. [11] use both the semantics of the identifiers of WSDL descriptions and the structure of their operations, messages and data types to assess the similarity of WSDL files used to describe e-Services. Bernstein et al. [1] compare traditional keyword-based service retrieval approaches and propose a novel approach, based on the use of process models, ensuring qualitatively higher retrieval precision. All current approaches agree on the need of a flexible matchmaking algorithm, where not only advertisements that match exactly with the request are returned, but also those that present a

---

*This work has been partially supported by the MAIS (Multichannel Adaptive Information Systems) FIRB Project funded by the Italian Ministry of Education, University and Research [9] and by NoE INTEROP IST Project n. 508011 - 6th EU Framework Program.
lower degree of match. Quality of service issues are also addressed by identifying the QoS parameters deemed useful to extend service discovery based on QoS-related information other than on interfaces only [10].

In this paper we present a novel, ontology-based approach to service discovery, which exploits domain knowledge and semantic service descriptions to guide the service discovery process and provide advice on service selection and instantiation in interoperable adaptive information systems. We will show how our approach performs a flexible and efficient matchmaking between service descriptions combining together different kinds of comparison strategies. The proposed system architecture has the advantage of providing specific advice at multiple levels of granularity during the service composition process. In Section 2 we present the proposed ontology-based architecture. Section 3 presents the semantic service discovery formalism and an example problem taken from MAIS [9] domain, while Section 4 describes the discovery and advice strategy. We conclude in section 5 with advantages offered by our approach and possible future work.

2 The proposed ontology-based architecture

The proposed approach to service discovery and advice is primarily based on knowledge provided by the domain and service ontologies. A domain ontology is used to conceptualize domain knowledge with commonly accepted vocabulary and to provide semantics to service descriptions. In fact, in our approach, semantic information is derived for services, published in a UDDI Registry, by reasoning on service properties related to concepts in the domain ontology. In particular, semantic relationships among services are identified and used to organize services in a multiple layers service ontology, as described in the following. Ontologies are two of the main elements of the COMPAT (COMPATibility) system, that is being implemented within the MAIS Project, and they are described by using Description Logics, which can be consistently translated into OWL-DL. The discovery and advice tasks are performed by the Compatible Service Manager in charge of semantically organizing services in the ontology (Semantic Publisher) and of semantically searching services by exploiting underlying ontologies (Matchmaker). Specifically, service matchmaking is performed with the support of inference mechanisms of a Description Logics reasoner. External users and applications interact with the system by means of the COMPAT API, that extends keyword-based API of traditional UDDI Registry maintaining backward compatibility with it. In the following we will describe the core elements of the system, also shown in Figure 1.

Domain Ontology. The Domain Ontology DomONT contains definitions of concepts in the domain to which elements used to describe services (input/output parameters, operation names, service categories) refer. In particular, the Domain Ontology is based on available classifications for processes (such as the MIT Process Handbook) and services (for example, UNSPSC or NAICS) to compare operation names and service categories, respectively. Concepts are related by means of the usual semantic relationships: concept generalization/specialization, concept equivalence and concept disjunction.

The Domain Ontology provides knowledge to reason about operation names, I/O parameter names and categories in service descriptions and therefore supports service matchmaking (see Section 4).

Service Ontology. The Service Ontology ServONT organizes services in a three layer structure by means of semantic relationships that will be exploited to support service discovery and selection (see Section 4). Starting from the bottom layer, we distinguish between concrete services, abstract services and subject categories, organized into Concrete, Abstract and Category layer, respectively. Both concrete and abstract services are described using Description Logics, as shown in Section 3.

Concrete services are directly invocable services, described by their public WSDL interface (operations, I/O parameters); they are published in the UDDI Registry and they are referenced in the concrete layer of the service ontology, where they are semantically organized into clusters of similar services according to properly defined coefficients that evaluate their functional similarity [6]. To evaluate these coefficients the Domain Ontology is investigated to verify if operation and I/O parameter names are semantically related. If their degree of similarity is equal or higher than a given threshold, two concrete services are grouped in the same cluster. Each cluster is associated to a corresponding abstract service in the abstract layer.

Abstract services are not directly invocable services, but are introduced to represent the functionalities of sets of similar concrete services. Abstract service functionalities are also described by means of an interface, obtained from concrete service descriptions by means of an integration pro-

![Figure 1. The COMPAT system architecture.](image-url)
cess that identifies correspondences among operations (that is, operation and I/O names) of similar concrete services in the same cluster and represents the corresponding operations in the abstract service; mapping rules are maintained among the operation and I/O names of the abstract functionalities and the original concrete ones. Abstract services are semantically organized according to two kinds of semantic relationships:

- an abstract service \( S_\alpha \) is a generalization of another abstract service \( S_\beta \) if, informally stated, \( S_\beta \) provides at least the operations of \( S_\alpha \) as established by properties of plug-in match discussed in Section 4;
- an abstract service \( S_\alpha \) isComposedOf a set \( \Phi = \{ S_1, S_2, \ldots, S_n \} \) of other abstract services if the operations of \( S_\alpha \) are included in the union set of operations of \( S_1, S_2, \ldots, S_n \); in particular, \( \Phi \) must be minimal (no redundancy); the service \( S_\alpha \) is often called the composite service, while \( S_1, S_2, \ldots, S_n \) are called the component services.

Semantic relationships between abstract services are set by the ontology designer possibly with the support of the Compatible Service Manager by exploiting domain knowledge and matching capabilities.

Finally, subject categories correspond to categories of services and organize abstract services into standard available taxonomies (such as UNSPSC or NAICS) to provide a topic-driven access to the underlying abstract and concrete services; these categories are the same that are used in the UDDI Registry to classify published concrete services; in our service ontology, each abstract service is associated to the set of all the categories related to the concrete services belonging to the corresponding cluster.

Compatible Service Manager. The Compatible Service Manager exploits the underlying ontologies to speed up the service retrieval. It is composed of two sub-modules:

- a Semantic Publisher, which is in charge of extracting from the UDDI Registry a newly registered concrete service and of placing it into the Service Ontology; it associates the new service with a suitable cluster by means of similarity analysis and modifies the corresponding abstract service to take into account the registered service interface; it can also support the ontology designer in finding semantic relationships between abstract services;
- a MatchMaker, which first selects abstract services that match a given service request from the functional perspective, according to the matchmaking algorithm exposed in Section 4; then, it selects concrete services associated with the abstract ones, taking into account non-functional aspects like quality requirements, contextual conditions or concrete service availability; finally, it returns to the requestor a list of descriptions of concrete services that match his request, together with links to their implementations (these links are stored in the UDDI Registry).

In this work we will focus on the Matchmaker functionalities to expose how the Compatible Service Manager provides semantic searching capabilities. The Compatible Service Manager is able to communicate with the Domain and Service Ontologies by means of a properly defined java-based interface and, to perform its tasks, exploits inference mechanisms of the RACER reasoner.

COMPAT Registry API. This Application Program Interface constitutes the borderline between the external users/applications and the COMPAT system. It provides the mechanisms to: (i) submit the service request description to the Compatible Service Manager (in particular, to the MatchMaker) in order to discover matching concrete services; (ii) submit a new service description to the Semantic Publisher in order to be registered in the UDDI Registry and placed into the service ontology. Moreover, the COMPAT API allows for using traditional UDDI Registry searching functionalities, ensuring backward compatibility with existing technologies and, in this sense, extending the UDDI API. The COMPAT API can be used by a human user by means of a Graphical User Interface implemented with currently existing java-based technologies (such as Java Servlet and Java Server Pages) or can be programmatically invoked by another application when the registry is used in a wider context, where automated retrieval, composition and invocation of services is required.

The proposed knowledge-based architecture for service discovery and advice system has the capability of providing specific service advice at different levels of granularity during a service composition process as required in the MAIS architecture [9]. At the highest level, the system can help to determine what kind of abstract service is required against a contextual functional request. Once all the services that can fulfill the required functionalities are discovered, the advice system can recommend an appropriate concrete service, taking into account both problem characteristics and quality considerations. More specialized, in-depth advice can also be given, for example, on how to initialize and configure the parameters of a concrete service by using mapping rules.

3 Service description

Dealing with service matchmaking algorithms, the first problem that must be addressed is the service capability representation by implicitly or explicitly describing operations, inputs, outputs and state transformations (usually expressed as preconditions and effects).

Our approach uses a hybrid service capability representation in terms of inputs, outputs, operation names and service categories. To allow DL-based reasoning on services, we express their functional description by means of Description Logics constructs. A service description is conceived as:

- a concept name, which represents the associated subject category;
• a conjunction of one or more concepts in the form \( \exists \text{hasOperation}.OP \), where \( OP \) is a concept representing an operation of the current service.

Each operation \( OP \) is described as the conjunction of:

• an atomic concept, to represent the operation name;
• a concept in the form \( \exists \text{hasInput}.I \), where \( I \) is a conjunction of concepts representing the input parameters;
• a concept in the form \( \exists \text{hasOutput}.O \), where \( O \) is a conjunction of concepts representing the output parameters.

Input and output parameters are concepts in the form \( \exists R.C \), where \( R \) is the name of the parameter and \( C \) is an atomic concept or an enumeration \( \{i_1, i_2, \ldots, i_n\} \) of individuals to represent the parameter domain. As a case study to demonstrate application of our approach we will consider the traditional touristic scenario, where a traveler is looking for a flight booking service.

**Case study** We consider a portion of our service ontology (shown in Figure 2), where three kinds of flight reservation services (that is, three abstract services) are registered in: (i) worldwide flight reservation services, (ii) flight reservation services to book flights between European cities only, (iii) flight reservation services to book flights between European and US cities.

Each abstract service is related to a cluster of concrete services. For space restrictions, we consider only one flight reservation operation for each abstract service, as shown in Table 1. For example, if we consider the abstract service FlightBookingInEurope, it can be expressed using DL notation in the following way:

\[
\text{FlightBookingInEurope} \sqsubseteq \text{AirTransfer} \sqcap \exists \text{hasOperation}.(\text{flightBooking} \sqcap \exists \text{hasInput}.(\exists \text{departureCity}.\text{EuropeanCity} \sqcap \exists \text{arrivalCity}.\text{EuropeanCity} \sqcap \exists \text{departureDate}.\text{Date} \sqcap \exists \text{arrivalDate}.\text{Date}) \sqcap \exists \text{hasOutput}.(\exists \text{ticket}.\text{flightTicket})
\]

**Table 1. Description of abstract services considered in the running example.**

<table>
<thead>
<tr>
<th>Operation name</th>
<th>Parameters</th>
</tr>
</thead>
</table>
| FlightBooking  | - departureCity:City  
|                | - arrivalCity:City   
|                | - departureDate:Date  
|                | - arrivalDate:Date    
|                | - ticket:flightTicket |
| FlightBooking in Europe | - departureCity:EuropeanCity  
|                        | - arrivalCity:EuropeanCity  
|                        | - departureDate:Date  
|                        | - arrivalDate:Date  
|                        | - ticket:flightTicket |
| FlightBooking between USA and Europe | - departureCity:EuropeanCity  
|                                      | - arrivalCity:USCity  
|                                      | - departureDate:Date  
|                                      | - arrivalDate:Date  
|                                      | - ticket:flightTicket |

In our example, Ryan Air, AirBaltic, KLM and Lufthansa provide flight reservation services only for European cities, while we suppose that the Delta Airlines and TransOcean Airlines services provide only flight booking between US and European cities; finally, Alitalia and British Airways provide worldwide flight reservation services. The portion of the DomONT ontology with the definitions of the concepts used for the considered services is the following:

\[
\text{DomONT} = \text{EuropeanCity} \sqsubseteq \text{City}  
\text{USCity} \sqsubseteq \text{City}  
\text{ItalianCity} \sqsubseteq \text{EuropeanCity}  
\text{TransportationService} \sqsubseteq \text{TravelService}  
\text{LandTransfer} \sqsubseteq \text{TransportationService}  
\text{AirTransfer} \sqsubseteq \text{TransportationService}
\]

A domain expert, supported by the Compatible Service Manager, can assert that Worldwide Flight Booking is a more specific service than Flight Booking in Europe and Flight Booking between USA and Europe in the sense that it offers at least the functionalities provided by the last ones. The domain expert can assess this fact in
by means of two generalization relationships as shown in Figure 2.

4 Exploiting service ontology for semantic discovery and selection

Our focus in this section is to show how the Compatible Service Manager exploits service ontology relationships to improve service discovery and selection. A service request \( R \) is described as a service description, where the service categories are optional and can be omitted. Once a description \( R \) is submitted to the COMPAT system through the API, searching for available services that match with the required one can be performed, starting from the abstract layer of the service ontology, by running and combining two kinds of comparisons between descriptions of abstract services and the request \( R \): (i) a deductive approach based on Description Logics, where DL-based representations of services are compared by querying a DL reasoner; (ii) a similarity-based approach, where properly defined coefficients are computed to evaluate the degree of similarity between services on the basis of the knowledge provided by the domain ontology [3, 6]. Furthermore, we exploit the organization of services in the service ontology by means of semantic relationships.

A Description Logics based approach is applied with the support of a DL reasoner to obtain a classification of matches between advertisements and request according to five kinds of matches [7]. Given a requested capability representation \( R \) and an advertisement capability representation \( S \) expressed using DL notation, we can consider the following kinds of matches:

- **Pre-filtering**, if one or more categories are specified in \( R \), then a pre-filtering process is performed to select for the following matches (Exact, Plug-in, Subsume and Intersection) only that advertisements \( S \) such that the category of \( S \) is the same or is related in any generalization hierarchy in \( DomONT \) with one of the categories of \( R \);

- **Exact match**, to denote that \( S \) and \( R \) have the same capabilities, that is, they have names of operations, input parameters and output parameters that are equivalent in \( DomONT \);

- **Plug-in match**, to denote that \( S \) offers at least the capabilities of \( R \), that is, the operations in \( R \) can be mapped into operations of \( S \) and, in particular, their names, input parameters and output parameters are related in any generalization hierarchy in \( DomONT \);

- **Subsume match**, to denote that \( R \) offers at least the capabilities of \( S \), like in plug-in match, but with the roles of \( R \) and \( S \) exchanged;

- **Intersection match**, to denote that \( S \) and \( R \) have some common operations and some common I/O parameters, that is, some pairs of operations and some pairs of parameters respectively are related in any generalization hierarchy in \( DomONT \);

**Mismatch**, otherwise.

An order can be established among these categories of match: exact > plug-in > subsume > intersection > mismatch. Similarity analysis is applied to further refine ranking of service matches. In particular, when exact match and plug-in match occur, similarity between advertisement and request is set to 1 (full similarity, the advertisement offers all functionalities required). Otherwise, when subsume and intersection match occur, similarity coefficients are computed to further refine in a quantitative way the ranking of returned services (for more details, see [6]). Finally, when mismatch occurs, then the similarity value is set to zero. Only the abstract services for which the similarity value is equal or greater than a given threshold \( \phi \) are returned as candidate services, ranked w.r.t. the kind of match and the degree of similarity, respectively.

Once the system verified that an abstract service description \( S_{ai} \) is a candidate offer for the request \( R \), ontological organization of abstract services is used to speed up the selection of other abstract services, related to \( S_{ai} \) by means of the semantic relationships in the Abstract Layer. This is done by exploiting the intended semantics of these relationships, considering another abstract service \( S_{aj} \) such that \( S_{ai} \) is a generalization of \( S_{aj} \):

- if \( S_{ai} \) presents an exact match with \( R \), then \( S_{aj} \) can present a plug-in match with \( R \), since it offers at least the functionalities of \( S_{ai} \), so no further application of matchmaking algorithm must be done and similarity\((S_{ai}, R)=similarity(S_{aj}, R)=1\);

- if \( S_{ai} \) presents a plug-in match with \( R \), then \( S_{aj} \) presents a plug-in match with \( R \), no further application of matchmaking algorithm must be done and similarity\((S_{ai}, R)=similarity(S_{aj}, R)=1\);

- if \( S_{ai} \) presents a subsume match with \( R \), then \( S_{aj} \) presents a subsume, a plug-in or an exact match with \( R \); matchmaking algorithm must be applied between \( S_{aj} \) and \( R \);

- if \( S_{ai} \) presents an intersection match with \( R \), then \( S_{aj} \) can present a subsume, a plug-in, an exact or an intersection match with \( R \); matchmaking algorithm must be again applied;

- if \( S_{ai} \) presents a mismatch with \( R \), no further applications of matchmaking algorithm is done.

The same procedure applies when \( S_{ai} \) is a composite service and \( S_{aj} \) is the union of its component ones. We note that the application of the matchmaking procedure to \( S_{aj} \) is required only when \( S_{ai} \) presents an intersection or a subsume match with \( R \), in the other cases the semantic relationships are exploited to include (or exclude) \( S_{aj} \) among the candidate abstract services. Once the set of candidate concrete services has been selected from the functional
viewpoint, further refinements can be performed, for example, to adapt the results to different quality requirements, to take into consideration different contextual conditions or to recover from situations in which availability of concrete services changes. This approach is useful in particular in an adaptive scenario, where the selected abstract service (that is, the desired functionalities) are maintained, while underlying concrete services can be changed according to different quality or contextual conditions in a transparent manner with respect to the user. Moreover, invocation of services can be made through abstract services, while initialization and configuration of concrete service parameters can be performed by means of mapping rules.

**Example.** We consider an user that searches for a travel service to book a flight from Milan to London. The following assertions are added to the Domain Ontology considered in the running example:

\[
\text{DomOnt} \sqsubseteq \text{Milan:ItalianCity} \sqcap \text{London:EuropeanCity}
\]

The service request \( R \) can be formalized using Description Logics as follows:

\[
R \sqsubseteq \text{TravelService} \sqcap \text{hasOperation.(flightBooking \sqcap \text{hasInput.} \{\text{departureCity.\{Milan\}} \sqcap \text{arrivalCity.\{London\}}\})}
\]

If we apply the DL-based matching phase between \( R \) and the Flight Booking in Europe service, we have a plug-in match. So the system must consider as candidate concrete services the set of services Ryan Air, AirBaltic, KLM and Lufthansa. Moreover, since there exists a generalization relationship between the Worldwide Flight Booking and Flight Booking in Europe, also Alitalia and British Airways are included into the set of candidate concrete services (with a similarity value also set to 1). Finally, selection of the best concrete services can then be made according to quality and/or contextual issues. For example, in [2] service descriptions are enriched with both general purpose and domain specific quality parameters, that generate QoS categories in which services are classified. User quality requirements individuate one or more QoS categories that contain the desired services to be presented to the user.

5 Conclusion

The proposed approach fully exploits domain and service ontologies for similarity-based and logic-based matching following semantic organization of service descriptions at different levels of abstractions. The approach is flexible, since it considers five kinds of matches (exact, plug-in, subsume, mismatch) and enhances flexibility with the evaluation of degree of similarity. Finally, the procedure is efficient, since it combines logic-based approach, similarity-based approach and semantic organization of services in the service ontology to reduce the number of comparisons between service descriptions.

**References**


