Ontology-based Semantic Interoperability Tools for Service Dynamic Discovery *

D. Bianchini and V. De Antonellis

University of Brescia
Dept. of Electronic for Automation
Via Branze, 38
25123 Brescia - Italy
bianchin|deantone@ing.unibs.it

Abstract. Enterprises and networked enterprises need today effective communication and exchange of distributed data and services under dynamic and context-dependent requirements. Semantic interoperability is considered a key issue to enforce dynamic discovery and composition of distributed data and services. In this paper we specifically address the problem of service discovery and we present an ontology-based approach for dynamic discovery in a highly variable environment when cooperative enterprises interoperate to dynamically combine available services selecting the best possible offers in a given moment. The ontology structure and its deployment are discussed.

1 Introduction

Enterprises and networked enterprises require advanced semantic interoperability methods and tools to enable cooperation and communication at application level. In particular, semantic interoperability techniques are being proposed to support data and service discovery and sharing [14]. Current approaches for service discovery address the treatment of dynamical aspects both with respect to the continuous addition and removal of services in a highly variable environment and with respect to different contexts in which a service could be invoked [5, 6]. Other works stressed the possibility of using Description Logics to implement matching algorithms and reasoning procedures to enhance service discovery [7, 13]. Ontologies are considered as an enabling technology for the Semantic Web and methods and tools for ontology definition are being studied for interoperability purposes. The use of ontology during service search allows for scalability of the systems when a large number of services is considered. In [2] a service retrieval approach based on the use of ontologies is presented. In [15] a service ontology specifies a domain, a set of synonyms to allow a flexible search for the

* This work has been partially supported by the MAIS (Multichannel Adaptive Information Systems [10]) FIRB Project funded by the Italian Ministry of Education, University and Research, and by NoE INTEROP [9] IST Project n. 508011 - 6th EU Framework Program.

domain and a set of service classes to define the properties of services, further specified by its attributes and operations. The service ontology also specifies a service quality model that is used to describe non functional aspects. In [4] a new technique for Web service discovery which features a flexible matchmaking by exploiting DAML-S ontologies is proposed.

In this paper we specifically address the problem of service discovery and we present an ontology-based approach for dynamic discovery in a highly variable environment when cooperative enterprises interoperate to dynamically combine available services selecting the best possible offers in a given moment. With respect to existing approaches to ontology-based service discovery, original contribution of our approach regards: (i) the ability of abstracting service characteristics from their operating environment to be able to dynamically select services on the basis of contextual features; in fact, our ontology has a three-layer architecture with different abstraction levels properly exploited to scale service discovery; (ii) the possibility of selecting services through semantic-based matching algorithms according to a scored mechanism taking into account semantic relationships among services. The aim of the present work is to present how the use of an ontology-based approach can support service dynamic discovery and selection. We propose an approach in which an ontology-based service description is used as a basis for providing service retrieval in an enhanced UDDI Registry. Instead of associating semantic information directly to services, semantic information is extracted from published services on the basis of a domain ontology and used as a basis to provide advanced searching functionalities.

The paper is organized as follows: in Section 2 we propose a three-layer service ontology architecture; in Section 3 we present a logical framework for service ontology representation; in Section 4, service discovery based on the proposed ontological framework is discussed and in Section 5 the architecture underlying our work is presented. Finally, conclusions are discussed in Section 6.

2 The Service Ontology Model

The proposed service ontology architecture organizes services at different layers of abstraction according to proper semantic relationships [3]. Semantic relationships, as explained in the next section, are the basis for a semantic searching engine, to improve traditional service discovery mechanisms (mainly based on keyword-driven search) with more sophisticated retrieval modalities based on reasoning services. Basic components in our approach are:

- a set of concrete services, described by means of their *WSDL interfaces*, one or more *concrete bindings* (for example, SOAP or HTTP binding) and one or more *endpoints* (that is, the physical localization of the concrete service); the service WSDL interface is described by means of the name of the service itself, the names of its operations and the names of inputs and outputs for each operation;
- an *UDDI Registry*, where concrete services are classified in terms of standard available service classifications (for example, UNSPSC or NAICS); UDDI

Registry offers searching utilities that are mainly keyword-based; one of the aims of our work is to maintain backward compatibility with these existing technologies and their searching functionalities;
- a *domain ontology*, where concepts used for operation and I/O entity names are organized by means of semantic relationships, i.e., *subsumption* and *equivalence*; a weight is assigned to each kind of semantic relationship in the domain ontology.

The conceptual model of the ontology is shown in Figure 1. The service ontology contains concrete services, abstract services and subject categories, organized into three layers (called *Concrete*, *Abstract* and *Category* layer, respectively).

- *Concrete services* are directly invocable services and they are featured by their public WSDL interface, which is used to group them on the basis of their functional similarity (as explained in the following), and by bindings to specific implementations.
- *Abstract services* are not directly invocable services, but represent the capabilities of sets of similar concrete services; abstract service capabilities are also described by means of a WSDL interface and are obtained from the concrete service operations by means of an integration process; mapping rules are maintained among the abstract capabilities and the original concrete operations; abstract services are related to each other by two kinds of semantic relationships: (i) *specialization/generalization*, when an abstract service offers at least the same functionalities of another one; (ii) *composition*, when the functionalities offered by a single abstract service can be provided also by a group of other abstract services, considered in their totality; in this case, the first services is often called the *composite service*, while the other ones are called the *component services*.
- *Subject categories* organize abstract services into a standard available taxonomy (such as UNSPSC or NAICS) to provide a topic-driven access to the underlying abstract and concrete services.

The three-layer service ontology is intended to enhance finding of generic services (*abstract services*) describing the required capabilities that can be actually provided by several specific existing services (*concrete services*). Thus, abstract services are intended to shorten the way towards a variety of alternative concrete services that can be invoked. Both abstract and concrete services are described by means of a *functional description*, that is, a set of capabilities (for the abstract service) or operations (for the concrete ones) with required inputs and provided outputs. Context information and quality requirements can be further considered to refine and filter the set of candidate concrete services. In addition, subject categories give the user a mechanism for an easy access to the underlying levels on the basis of standard topics.

Example 1 - Figure 2 shows a portion of three-layer service ontology in tourism domain. □

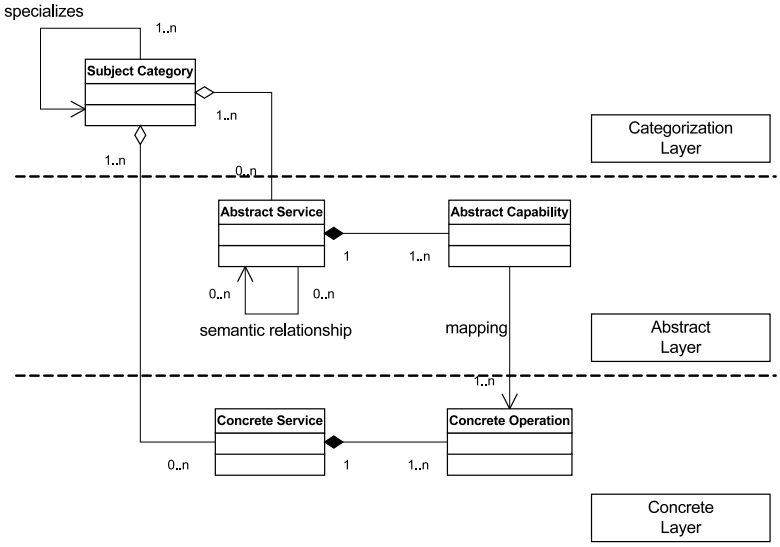

Fig. 1. Conceptual model of three layer service ontology.

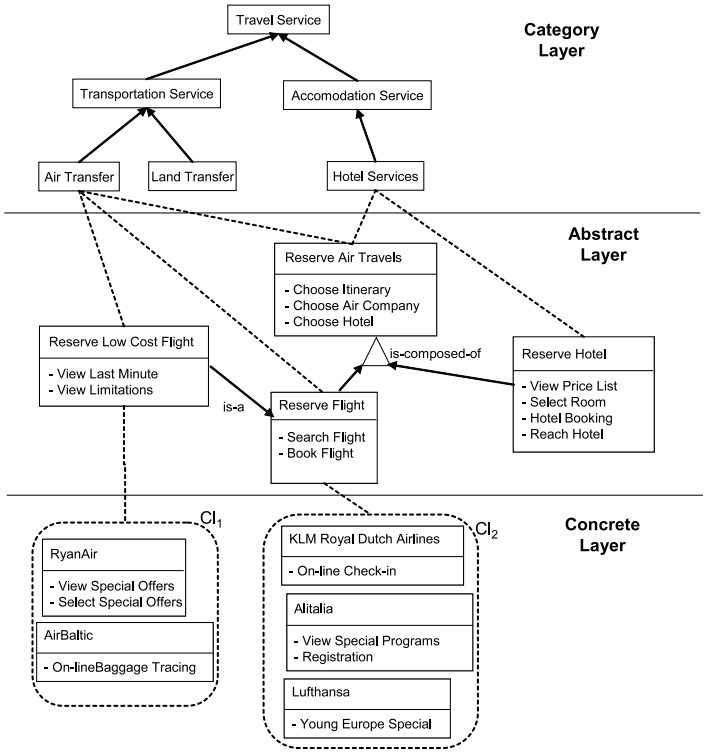

Fig. 2. A portion of three-layer service ontology in tourism domain.

3 A Description Logic for representing services in the ontology

In order to enhance semantic interoperability, Description Logics have been adopted for representing services in the ontology and for reasoning in service discovery. Description Logics are knowledge representation formalisms with enough expressiveness and inference procedures that are sound, that is, the termination of the reasoning algorithms is guaranteed. We consider the $\mathcal{SHOIN}(\mathcal{D})$ Description Logic, that is the logical formalism to which the OWL DL ontology language (a sub-language of OWL [11]) can be mapped directly [8]. We choose OWL DL as the representation language for the service and domain ontologies because of the expressiveness of its constructs and for the readability of its abstract syntax influenced by frame-based languages. $\mathcal{SHOIN}(\mathcal{D})$ allows for the definition of concepts and roles, defined on a *domain* $\mathcal{D}$ [1].

We start from a set of concept names (that are names of services and operations), each of them denoted by the letter A, a set of individual names, each of them denoted by the letter i and a set of role names, each of them denoted by the letter R; a concept C can be defined recursively as follows:

- a concept name A is a concept (called *atomic concept*);
- an enumeration of individuals $\{i_1, i_2, \ldots i_n\}$ is a concept;
- given two concepts C_1 and C_2, $C_1 \sqcap C_2$ (*conjunction*), $C_1 \sqcup C_2$ (*disjunction*), $\neg C$ and (C) are concepts;
- $\exists R.C$ (*existential role restriction*) and $\forall R.C$ (*universal role restriction*) are concepts.

Moreover, the *universal* ($\top$) and *empty concept* ($\bot$) are defined. In $\mathcal{SHOIN}(\mathcal{D})$ it is possible to define:

- *terminological equivalence* ($A_1 \equiv A_2$) between atomic concepts;
- transitivity of the role R ($Tr(R)$);
- *inclusion* ($C_1 \sqsubseteq C_2$) and *disjointness* ($C_1 \sqsubseteq \neg C_2$) between concepts to express intentional knowledge about them;
- the inverse role R^-;
- not qualified cardinality constraints, that is, $\leq nR$ (which stands for $\exists_{\leq n}\ y\ R(x,y)$), $\geq nR$ (which stands for $\exists_{\geq n}\ y\ R(x,y)$) and $= nR$ (which stands for $\leq nR \cap \geq nR$).

We define an ontology $\mathcal{ONT}$ as a set of assertions on concepts used to formalize abstract and concrete services (as explained in the next section).

The semantics of concepts is defined by an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \bullet^{\mathcal{I}})$, consisting of an *abstract domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\bullet^{\mathcal{I}}$; given an atomic concept A, a role name R with arity n, a set of individuals $\{i_1, i_2, \ldots i_n\}$, two generic concepts C_1 and C_2, we have:

$$
\begin{aligned}
A^{\mathcal{I}} &\subseteq \Delta^{\mathcal{I}} \\
R^{\mathcal{I}} &\subseteq (\Delta^{\mathcal{I}})^n \\
i^{\mathcal{I}} &\in \Delta^{\mathcal{I}} \\
(\{i_1, i_2, \ldots i_n\})^{\mathcal{I}} &= \{(i_1)^{\mathcal{I}}, (i_2)^{\mathcal{I}}, \ldots (i_n)^{\mathcal{I}}\} \\
(C_1 \sqcap C_2)^{\mathcal{I}} &= (C_1)^{\mathcal{I}} \cap (C_2)^{\mathcal{I}} \\
(C_1 \sqcup C_2)^{\mathcal{I}} &= (C_1)^{\mathcal{I}} \cup (C_2)^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} - C^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &= \{c \in \Delta^{\mathcal{I}} \mid \exists d \in \Delta^{\mathcal{I}} s.t. (c,d) \in R^{\mathcal{I}} \wedge d \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &= \{c \in \Delta^{\mathcal{I}} \mid \forall d \in \Delta^{\mathcal{I}} s.t. (c,d) \in R^{\mathcal{I}} \Rightarrow d \in C^{\mathcal{I}}\} \\
(\leq nR)^{\mathcal{I}} &= \{x \; s.t. \; |\{y.\langle x,y\rangle \in R^{\mathcal{I}}\}| \leq n\} \\
(\geq nR)^{\mathcal{I}} &= \{x \; s.t. \; |\{y.\langle x,y\rangle \in R^{\mathcal{I}}\}| \geq n\} \\
(= nR)^{\mathcal{I}} &= \{x \; s.t. \; |\{y.\langle x,y\rangle \in R^{\mathcal{I}}\}| = n\} \\
(R^-)^{\mathcal{I}} &= (R^{\mathcal{I}})^- \\
(Tr(R))^{\mathcal{I}} &= (R^{\mathcal{I}})^+
\end{aligned}
$$

The empty concept is mapped to the empty set, while the universal concept is mapped to $\Delta^{\mathcal{I}}$. An assertion $A_1 \equiv A_2$ is satisfied by an interpretation $\mathcal{I}$ if $(A_1)^{\mathcal{I}} = (A_2)^{\mathcal{I}}$; an assertion $C_1 \sqsubseteq C_2$ is satisfied by $\mathcal{I}$ if $(C_1)^{\mathcal{I}} \subseteq (C_2)^{\mathcal{I}}$; an assertion $C_1 \sqsubseteq \neg C_2$ is satisfied by $\mathcal{I}$ if $(C_1)^{\mathcal{I}} \cap (C_2)^{\mathcal{I}} = \emptyset$. An interpretation $\mathcal{I}$ that satisfies all the assertion in $\mathcal{ONT}$ is called a *model* for $\mathcal{ONT}$; a generic concept C is *satisfiable* in $\mathcal{ONT}$ if $\mathcal{ONT}$ allows for a model $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$; $\mathcal{ONT}$ *logically implies* an assertion between generic concepts if, for each model $\mathcal{I}$ of $\mathcal{ONT}$, the application of $\mathcal{I}$ to the assertion is satisfiable.

3.1 Service description

We consider the chosen Description Logic to represent the functional description of abstract and concrete services. To do this, we do not exploit the full expressiveness of $\mathcal{SHOIN}(\mathcal{D})$, but we make a limited use of its constructs. We consider subsumption, conjunction, disjunction and negation of atomic and complex concepts and existential restriction; in particular, we use four specific roles: the role `hasCategory` to express the association link of a service with a subject category, the role `hasOperation` to express the relationship of a service with its operations or capabilities and the role `hasInput` (resp., `hasOutput`) to specify that a given concept is the input (resp., the output) of an operation.

We describe the functional description of a service S_k as a conjunction of the following parts:

- an atomic concept, to represent the service name we denote it as *Name(S_k)*;
- a conjunction of concepts of the form $\exists$`hasCategory`.C, where C is a conjunction of concept names, to represent the associated subject categories; we denote such conjunction as *Categories(S_k)*;
- a conjunction of concepts of the form $\exists$`hasOperation`.OP, where OP is a concept representing an operation of the current service; we denote such conjunction as *Operations(S_k)*; each operation OP is described as the conjunction of:

- an atomic concept, to represent the operation name; we denote it as *Name(OP)*;
- a concept of the form $\exists$`hasInput.`I, where I is a conjunction of atomic concepts representing the inputs of the operation; each concept could also be defined as an enumeration of individuals in the form $\exists$`R.`ℓ, where R is the name of the input and $\ell = \{i_1, i_2, \ldots i_n\}$ is the optional enumeration of individuals; we denote the set of such concepts as *Inputs(OP)*;
- a concept of the form $\exists$`hasOutput.`O, where O is a conjunction of atomic concepts representing the outputs of the operation; each concept could also be defined as an enumeration of individuals; we denote the set of such concepts as *Outputs(OP)*.

Moreover, to distinguish among abstract and concrete services, we add to the conjunction a new concept of the form $\exists$`hasType.`C, where C is an atomic concept `abstractService` or `concreteService`. Finally, for each abstract service we can add a concept of the form $\exists$`hasConcrete.`CS, where CS is a conjunction of concepts corresponding to the names of concrete services associated to the abstract one.

Example 2 - If we consider the abstract service `ReserveFlight` in Figure 2, the following new concept is added to the $\mathcal{ONT}$ knowledge base:

```
ReserveFlight ⊓ ∃hasCategory.AirTransfer ⊓
    ∃hasOperation.(searchFlight ⊓ ∃hasInput.(departureCity ⊓
       arrivalCity ⊓ departureTime ⊓ arrivalTime) ⊓
       ∃hasOutput.flightTicket) ⊓
    ∃hasOperation.(bookFlight ⊓ ∃hasInput.flightTicket ⊓
       ∃hasOutput.bookingConfirmation) ⊓ ∃hasType.abstractService ⊓
    ∃hasConcrete.(KLM ⊓ Alitalia ⊓ Lufthansa)
```

□

Semantic relationships among abstract services (*specialization/generalization* and *composition*) are represented by means of *subsumption* ($\sqsubseteq$) and *disjunction* ($\sqcup$) constructs, that is:

- if an abstract service Sa_1 is a specialization of another one Sa_2, then we add $Sa_1 \sqsubseteq Sa_2$ to $\mathcal{ONT}$;
- if an abstract service Sa_1 is composed by a set of abstract services Sa_2, Sa_3, $\ldots Sa_n$, then we add $Sa_2 \sqcup Sa_3 \sqcup \ldots \sqcup Sa_n \sqsubseteq Sa_1$ to $\mathcal{ONT}$.

Example 3 - In the service ontology shown in Figure 2 we have:

```
ReserveFlight ⊔ ReserveHotel ⊑ ReserveAirTravel
ReserveLowCostFlight ⊑ ReserveFlight
```

□

3.2 Service request description

We represent a functional request $\mathcal{R}$ as well as the functional description of a service, with the only difference that in the request $\mathcal{R}$ the specification of re-

quired subject categories could be optional.

Example 4 - We consider the request of a flight booking service for a trip from Milan or Venice to Rome that allows for payment by credit card. The required capabilities will be two: the first one to book a flight, inserting departure and arrival city (defined by enumeration) and receiving the flight ticket, the second one to pay the flight by credit card, inserting card information and receiving a receipt via e-mail. The request $\mathcal{R}$ will be represented as follows:

```
∃hasCategory.(AirTransfer ⊓ TravelService) ⊓
∃hasOperation.(flightBooking ⊓
   ∃hasInput.(∃departureCity.{Milan,Venice} ⊓
      ∃arrivalCity.{Rome}) ⊓
   ∃hasOutput.flightTicket) ⊓
∃hasOperation.(paymentByCreditCard ⊓
   ∃hasInput.(creditCardHolder ⊓
      creditCardExpiration ⊓ creditCardNumber) ⊓
   ∃hasOutput.(∃paymentReceipt.{email})
```

□

4 An ontological infrastructure for service discovery

In this section we show how to exploit the proposed ontological infrastructure to enhance service discovery by means of a matching and ranking algorithm that finds desired concrete services according to a set of functionalities required by the user (that is, a request $\mathcal{R}$) and ranks them with respect to their degree of similarity with $\mathcal{R}$. First of all, we give the definition of similarity coefficients useful for matching a request $\mathcal{R}$ and an abstract service Sa_i. Then, we present our matching and ranking algorithm for service discovery.

4.1 Functional similarity coefficients

The functional similarity analysis is supported by the domain ontology used to annotate I/O entities and operation names, where concepts are organized by means of weighted semantic relationships (*equivalence* with weight equal to 1, *subsumption* with weight equal to 0.8).

The domain ontology is organized as a graph $\langle \mathcal{N}, \mathcal{E} \rangle$, where $\mathcal{N}$ is the set of nodes (i.e., concepts) and $\mathcal{E}$ the set of edges (i.e., semantic relationships between nodes). Each edge is represented in the form $\langle (n_h, n_k), t, w \rangle$, where $n_h \in \mathcal{N}$ is the source node, $n_k \in \mathcal{N}$ is the destination node, t is the kind of relationship and $w \in (0, 1]$ is the weight associated to that kind of relationship. We call *path* p between two nodes $n, n' \in \mathcal{N}$ a finite ordered sequence $\langle e_1, e_2, \ldots e_n \rangle$, where the source node of e_1 is n and the destination node of e_n is n'. Between two nodes in the ontology there can exist more than one path. We define $\mathcal{P} \subseteq 2^{\mathcal{E}}$ the set of all possible paths in the ontology. The *strength* of a path $p \in \mathcal{P}$ is the value of the function $\mathcal{W} : \mathcal{P} \rightarrow (0, 1]$ that associates to p the product of the weights of all

the relationships belonging to it. We say that two terms n and n' have *affinity* ($n \cong n'$) if:

- there exists at least one path in the ontology between n and n';
- the strength of this path is greater or equal to a given threshold; if there exist more than one path, the one with the highest strength is chosen; the affinity value $A(n, n') \in (0, 1]$ is equal to the strength of the path.

Given a request $\mathcal{R}$ and an abstract service Sa_i, we consider each pair of operations op_R and op_i, one from $\mathcal{R}$ and one from Sa_i. The similarity between op_R and op_i is evaluated considering the domain ontology, that is,

$$OpSim(op_R, op_i) = A(n_R, n_i) \in (0, 1] \tag{1}$$

where n_R and n_i are the concepts associated to the names of op_R and op_i. If a path of relationships does not exist among n_R and n_i names, $OpSim(op_R, op_i) = 0$. If $OpSim(op_R, op_i) \neq 0$, then also similarity among I/O entities is verified exploiting the domain ontology through the formula

$$\begin{aligned} ESim(op_R, op_i) = & \frac{2 \cdot A_{tot}(IN(op_R), IN(op_i))}{\mid IN(op_R) \mid + \mid IN(op_i) \mid} \\ & + \frac{2 \cdot A_{tot}(OUT(op_R), OUT(op_i))}{\mid OUT(op_R) \mid + \mid OUT(op_i) \mid} \in [0, 1] \end{aligned} \tag{2}$$

where A_{tot} represents the total value of affinity between the pairs of I/O entities, one from op_R and one from op_i (obtained by summing up the affinity values of all the pairs of I/O entities with affinity) and $\mid \ \mid$ denotes the cardinality of a given set of I/O entities; the higher the number of pairs of entities with affinity, the higher the value of $ESim(op_R, op_i)$.

The affinity evaluation between inputs and outputs must consider both associated concept names and optional enumerated values. For example, if we consider two input entities I_1 and I_2 defined as $\exists n_1.\ell_1$ and $\exists n_2.\ell_2$, respectively, where n_1 and n_2 are concept names, ℓ_1 and ℓ_2 are enumerations of individuals, we firstly evaluates $A(n_1, n_2)$, then we verify if for at least one of the two inputs ℓ is defined; in such a situation, if $\ell_1 \equiv \ell_2$ then we put $A(n_1, n_2) := A(n_1, n_2) * 1$, else if $\ell_1 \sqsubseteq \ell_2$, then we put $A(n_1, n_2) := A(n_1, n_2) * 0.8$. If ℓ is defined for only one input, for the other one it is interpreted as *"all possible values accepted"*.

For each pair of operations, a global similarity coefficient $GSim(op_R, op_i)$ is evaluated through a weighted sum of the values of $ESim(op_R, op_i)$ and $OpSim(op_R, op_i)$:

$$GSim(op_R, op_i) = \alpha \cdot ESim(op_R, op_i) + \beta \cdot OpSim(op_R, op_i) \in [0, 1] \tag{3}$$

The weights α and β, with $\alpha, \beta \in [0, 1]$ and $\alpha + \beta = 1$, are properly set to assess the desired relevance of each kind of similarity, depending on flexible

comparison strategies (generally, $\alpha = \beta = 0.5$). Two operations op_R and op_i are *globally similar* if the value of $GSim(op_R, op_i)$ is equal or greater than a given threshold λ.

Finally, the degree of functional similarity between a request $\mathcal{R}$ and an abstract service Sa_i is computed as the ratio among the sum of $GSim$ values for each pair of globally similar operations in $\mathcal{R}$ and Sa_i and the total number of required operations:

$$FSim_R(\mathcal{R}, Sa_i) = \frac{\sum_{s,t} GSim(op_R, op_i)}{\mid operations\ in\ \mathcal{R} \mid} \tag{4}$$

for each op_R of $\mathcal{R}$ and for each op_i of Sa_i such that $GSim(op_R, op_i) \geq \lambda$.

Given a threshold ϕ, if $FSim_R(\mathcal{R}, Sa_i) \geq \phi$ then we can assess *functional compatibility*, that is, $isCompatible(\mathcal{R}, Sa_i)$.

We pose two main assumptions, derived from the meaning of semantic relationships between abstract services; the first one asserts that, if an abstract service Sa_1 is a *specialization* of another one Sa_2 and Sa_2 satisfies the user request $\mathcal{R}$, then also Sa_1 satisfies $\mathcal{R}$, that is

$$\textbf{if } \mathcal{ONT} \models Sa_1 \sqsubseteq Sa_2 \ \wedge\ isCompatible(\mathcal{R}, Sa_2) \textbf{ then } isCompatible(\mathcal{R}, Sa_1)$$

The second assumption asserts that, if an abstract service Sa_1 is composed by a set of abstract services Sa_2, Sa_3, $\ldots Sa_n$ and Sa_1 satisfies the user request $\mathcal{R}$, then $\mathcal{R}$ is also satisfied by the overall set $\{Sa_2, Sa_3, \ldots Sa_n\}$, that is

$$\textbf{if } \mathcal{ONT} \models (Sa_1 \equiv Sa_2 \sqcup Sa_3 \sqcup \ldots \sqcup Sa_n) \ \wedge\ isCompatible(\mathcal{R}, Sa_1) \textbf{ then } isCompatible(\mathcal{R}, Sa_2 \sqcup Sa_3 \sqcup \ldots \sqcup Sa_n)$$

4.2 The FC-MATCH algorithm for service discovery

The algorithm is organized in four main phases as shown in Figure 3. During the first one (rows (9)-(16)) subject categories optionally specified in the request are used to reduce the set of candidate abstract services: only abstract services that are associated to at least one selected subject category ($marked(Sa_i)$ in the algorithm) are further considered for the evaluation of their functional compatibility with the request. If no subject categories are specified, then all abstract services are considered in the next phase.

In the second phase (rows (17)-(23)), candidate abstract services are compared with the request $\mathcal{R}$ to evaluate their degree of functional compatibility with $\mathcal{R}$. Only abstract services Sa_i for which $FSim_R(\mathcal{R}, Sa_i)$ is equal or greater than a given threshold ϕ are further considered.

In the third phase (rows (24)-(29)) semantic relationships between abstract services are exploited to make more efficient selection of candidate abstract services, according to assumptions presented in Section 4. This phase can reduce

(1) **Algorithm** FunctionalCompatibilityMatch
(2) **Input** service ontology $\mathcal{ONT}$, a request $\mathcal{R}$, a threshold ϕ
(3) **Output** a set $\mathcal{CSC}$ of pairs $(Sc_i, score(Sc_i))$, where Sc_i are concrete services returned to the user, ranked with respect to $score(Sc_i)$

(4) $\mathcal{AS}$:= set of available abstract services *//initialization*
(5) **foreach** $Sa_i \in \mathcal{AS}$
(6) $score(Sa_i) := 0$;
(7) $\mathcal{ASC} := \emptyset$ *//set of candidate abstract services*
(8) $\mathcal{CSC} := \emptyset$

(9) **if** $Categories(\mathcal{R}) \neq \emptyset$ *//category level filtering (optional)*
(10) **foreach** $Sa_i \in \mathcal{AS}$
(11) **foreach** $Cat_h \in Categories(\mathcal{R})$
(12) **if** $Cat_h \in Categories(Sa_i)$ **then** mark Sa_i
(13) **if** $marked(Sa_i)$ **then**
(14) add Sa_i to $\mathcal{ASC}$;
(15) remove Sa_i from $\mathcal{AS}$;
(16) **else** $\mathcal{ASC} := \mathcal{AS}$ *//if no categories are specified, all available abstract //services are candidate services*

(17) $\mathcal{AS} := \mathcal{ASC}$; *//maintain only abstract services associated to the //selected categories, if specified*
(18) $\mathcal{ASC} := \emptyset$;
(19) **foreach** $Sa_i \in \mathcal{AS}$ *//functional similarity evaluation*
(20) **if** $FSim_R(\mathcal{R}, Sa_i) \geq \phi$ **then**
(21) $score(Sa_i) := FSim_R(\mathcal{R}, Sa_i)$;
(22) add Sa_i to $\mathcal{ASC}$;
(23) remove Sa_i from $\mathcal{AS}$;
(24) **foreach** $Sa_j \neq Sa_i$ such that $\mathcal{ONT} \models Sa_j \sqsubseteq Sa_i$ **then** *//exploitation of specialization relationships between //abstract services*
(25) $score(Sa_j) := score(Sa_i)$;
(26) add Sa_j to $\mathcal{ASC}$;
(27) remove Sa_j from $\mathcal{AS}$;
(28) **foreach** $\{Sa_j\} \neq Sa_i$ such that $\mathcal{ONT} \models (\sqcup\{Sa_i\} \sqsubseteq Sa_i)$ **then** *//exploitation of composition relationships between //abstract services*
(29) $score\{Sa_j\} := score(Sa_i)$;
(30) add $\{Sa_j\}$ to $\mathcal{ASC}$;
(31) remove $\{Sa_j\}$ from $\mathcal{AS}$;
(32) **else** remove Sa_i from $\mathcal{AS}$;

(33) **foreach** $Sa_i \in \mathcal{ASC}$ **then** *//mapping from the abstract to the concrete world*
(34) **foreach** $Sc_k \in concretes(Sa_i)$
(35) $score(Sc_k) := score(Sa_i)$;
(36) add $(Sc_k, score(Sc_k))$ to $\mathcal{CSC}$;

(37) **return** $\mathcal{CSC}$ ranked with respect to $score(Sc_i)$;

Fig. 3. The FC-MATCH algorithm.

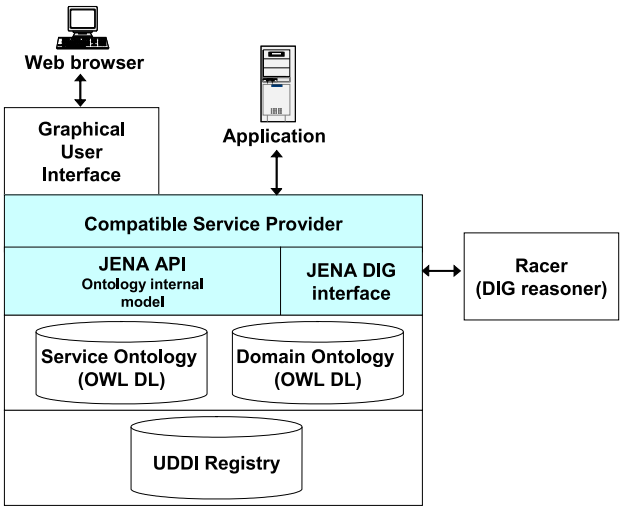

Fig. 4. The COMPAT system architecture.

the set of abstract services for which the evaluation of $FSim_R(\mathcal{R}, Sa_i)$ coefficient is needed, decreasing the complexity of the algorithm.

Finally, concrete services associated to the selected abstract ones are returned to the user, ranked according to the degree of their functional compatibility with the functional requirements (rows (31)-(35)). Future work will address further refinement of the set of concrete services taking into account contextual information and QoS issues. In such a case, the value of $score(Sc_j)$ of each selected concrete service Sc_j will be further modified according to contextual and QoS aspects. Notice that we addressed scalability purposes by exploiting the three-layer structure of the service ontology (we progressively reduce the set of candidate services by considering the organization of services into subject categories and abstract services) and semantic relationships between abstract services.

5 The COMPAT system architecture

Figure 4 shows the proposed architecture for the COMPAT ontology-based system to enhance service discovery according to compatibility degree. The core elements of the architecture are the `Compatible Service Provider`, the `JENA API` and the `JENA DIG Interface`. The `Compatible Service Provider` implements the matching algorithm presented in Section 4 to find available concrete services with respect to the user requests. The `Compatible Service Provider` does not access directly the ontology, but it uses the `JENA API`, a Java interface that includes a subsystem for management of ontologies supporting OWL,

DAML+OIL and RDF. `JENA` uses an internal model to represent ontologies, on which it is possible to add a plug-in to perform reasoning tasks. An automatic reasoner, RACER [12], has been developed to implement reasoning tasks on $\mathcal{SHOIN}(\mathcal{D})$; it can be used by means of the Jena DIG-HTTP interface. The `Compatible Service Provider` satisfies the request of a service both from a human user, that can interact with the system by means of a `Graphical User Interface (GUI)`, implemented using *Java Servlet* and *Java Server Pages* technologies, and directly from a software application to automate service selection and discovery.

6 Conclusions

In this paper we have addressed the problem of semantic interoperability in service discovery to support enterprises in dynamically selecting the best possible offers in a given moment. We have proposed a service ontology architecture and a matching algorithm to exploit it in order to find concrete services according to a given service request. We have also proposed a mechanism to rank the resulting set of concrete services. In this paper we have considered only functional aspects of services, while future works will address also a QoS-based and context-aware selection of concrete services. The basic idea is to take into consideration QoS and contextual aspects to filter concrete services selected by the FC-MATCH algorithm. Negotiation techniques could also be considered in this phase. At the moment, a prototype of the architecture shown in the previous section and the experimentation in the domain of touristic information services are being completed.

References

1. F. Baader and P. Hanschke. A schema for integrating concrete domains into concept languages. In *Proc. of the IJCAI 1991*, pages 452–457, 1991.
2. A. Bernstein and M. Klein. Towards High-Precision Service Retrieval. In *Proc. of the International Semantic Web Conference (ISWC 2002)*, pages 84–101, Sardinia, Italy, June 9-12th 2002.
3. D. Bianchini, V. De Antonellis, B. Pernici, and P. Plebani. Ontology-based Methodology for *e*-Service discovery. *Accepted for publication on Journal of Information Systems, Special Issue on Semantic Web and Web Services*, 2004.
4. A. Brogi, S. Corfini, and R. Popescu. Flexible Matchmaking of Web Services Using DAML-S Ontologies. In *Proc. Forum of the Second Int. Conference on Service Oriented Computing (ICSOC 2004)*, New York City, NY, USA, November 15-19th 2004.
5. F. Casati, M. Castellanos, U. Dayal, and M. Shan. Probabilistic, Context-Sensitive and Goal-Oriented Service Selection. In *Proc. of the Second Int. Conference on Service Oriented Computing (ICSOC 2004)*, pages 316–321, New York City, NY, USA, November 15-19th 2004.
6. I. Elgedawy, Z. Tari, and M. Winikoff. Exact Functional Context Matching for Web Services. In *Proc. of the Second Int. Conference on Service Oriented Computing (ICSOC 2004)*, pages 143–152, New York City, NY, USA, November 15-19th 2004.

7. J. Gonzalez-Castillo, D. Trastour, and C. Bartolini. Description logics for matchmaking of services. Technical report, HPL-2001-265, Hewlett-Packard, 2001.
8. I. Horrocks and P. Patel-Schneider. Reducing OWL entailment to Description Logic satisfiability. In *Proc. of the 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida, USA, October 2003.
9. The INTEROP NoE Portal. `http://www.interop-noe.org/`.
10. The MAIS Project Home Page. `http://black.elet.polimi.it/mais/index.php`.
11. D. L. McGuinness and F. van Harmelen. *OWL Web Ontology Language Overview*. World Wide Web Consortium (W3C) Recommendation, February 10th 2004. `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.
12. The Racer Home Page. `http://www.sts.tu-harburg.de/ r.f.moeller/racer/`.
13. B. Motik S. Grimm and C. Preist. Variance in e-Business Service Discovery. In *Proc. of the Third International Semantic Web Conference (ISWC 2004)*, Hiroshima, Japan, November 8th 2004.
14. HP - Web Services concepts. A technical overview. `http://www.bluestone.com/downloads/pdf/web_services_tech_overview%w.pdf`.
15. L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004.