Appl Intell (2006) 25:223–237
DOI 10.1007/s10489-006-9655-4

Service matching in agent systems

Anton Naumenko · Sergiy Nikitin · Vagan Terziyan

Abstract The problem of service and resource matching is being actively discussed currently as a new challenging task for the next generation of semantic discovery approaches for Web services and Web agents. A significant advantage is expected when using an ontological approach to semantically describe and query services. A matchmaking problem arises when a service is being queried and it includes the distance measure between the required service description and the one from the service registry. We realized the need to analyze the applicability of different matchmaking methods to agent development tools when implemented according to agent technology specifications such as FIPA. We consider three main groups of cases: matchmaking between classes of service profiles in pure taxonomies, matchmaking between classes in faceted taxonomies, and matchmaking between instances of faceted taxonomies.

Keywords Agent technology · Ontology · Service matching · Similarity

1. Introduction

The problem of service and resource matching is being hotly discussed now as a new challenging task for the next generation of annotation approaches to Web services and resources. The most significant outcome in matching was reached using an ontological approach to the description of a domain. The ontologies and ontology description languages are currently being developed by different standardization organizations such as W3C consortium.

We have met the problem of resource (device, expert, service) matchmaking in the SmartResource project [1, 2]. The SmartResource project, led by the Industrial Ontologies Group, is aimed at developing a framework for the adaptation of different heterogeneous resources to the common so-called Global Understanding Environment (GUN) and provisioning, based on this framework, a higher level agent-based interoperability and resource description management.

A similar problem was met in the Adaptive Services Grid project [3], which concentrated on the integration of two IT worlds such as Semantic Web Services and Grid Computing by an open generic software platform for adaptive services discovery, creation, composition, and enactment to offer new applications on Grid Services providing a well-defined range of quality of service. The matchmaking of resources and services is one of the most challenging tasks of the project. The matchmaking problem addresses the question of the distance measure between objects and because the SmartResource project raised the task of enforcing the GUN platform with agents, we realized the need to analyze the applicability of different matchmaking methods to agent development tools that are compliant to agent technology specifications such as FIPA [4].

There are many approaches to defining distance between any two entities (attributes, terms) based on their numerical or semantic closeness. For example, Tailor and Tudhope [5] have presented a hypermedia architecture that is supported by a classification schema. Semantic closeness measures have been developed to measure the closeness of terms in a schema

A. Naumenko · S. Nikitin (✉) · V. Terziyan
Industrial Ontologies Group, MIT Department,
University of Jyvaskyla, P.O. Box 35 (Agora),
FIN-40014 Jyvaskyla, Finland
e-mail: senikiti@cc.jyu.fi

A. Naumenko
e-mail: annaumen@cc.jyu.fi

V. Terziyan
e-mail: vagan@it.jyu.fi

which provides a platform for high-level navigation tools and which can provide flexible access tools to a collection of material. Two higher level navigation tools, navigation via media similarity and best-fit generalization, also have been developed. The similarity coefficients are extended in that similarity is judged on the "semantic closeness" of the sets of classification terms that are attached to the media nodes. The similarity coefficient therefore needs to be able to handle sets of classification terms with varying lengths, with non-exact matches of terms, and where the pairing of terms between media nodes may not be immediately obvious.

Brooks reports two experiments that investigated the semantic distance model (SDM) of relevance assessment [6]. In the first experiment, graduate students of mathematics and economics assessed the relevance relationships between bibliographic records and hierarchies of terms composed of classification headings and help-menu terms. The relevance assessments of the classification headings, but not the help-menu terms, exhibited both a semantic distance effect and a semantic direction effect as predicted by the SDM. Topical subject expertise enhanced both these effects. The second experiment investigated whether the poor performance of the help-menu terms was an experimental design artifact reflecting the comparison of terse help terms with verbose classification headings. In the second experiment, the help menu terms were compared to a hierarchy of single-word terms where they exhibited both a semantic distance and semantic direction effect.

Foo et al. [7] propose and define a modification of Sowa's metric on conceptual graphs. The metric is computed by locating the least subtype which subsumes the two given types, and then adding the distance from each given type to the subsuming type.

The distance metric used by Rada et al. [8] represents the conceptual distance between concepts. Rada et al. use only the path length to determine this conceptual distance, with no consideration of node or link characteristics. Distance is measured as the length of the path representing the traversal from the first classification term to the second. The closeness of terms ranges from 1 (identical terms) to 0 (which represents that terms are not semantically close, although it does not mean that they are disjoint in the classification schema).

Instance-based learning techniques typically handle continuous and linear input values well, but often do not handle nominal input attributes appropriately. The Value Difference Metric (VDM) was designed by Wilson and Martinez [9] to find reasonable distance values between nominal attribute values, but it largely ignores continuous attributes, requiring discretization to map continuous values into nominal values. Wilson and Martinez propose new heterogeneous distance functions, called the Heterogeneous Value Difference Metric (HVDM), the Interpolated Value Difference Metric (IVDM), and the Windowed Value Difference Metric (WVDM). These new distance functions are designed to handle applications with nominal attributes, continuous attributes, or both. As was mentioned in the Wilson and Martinez review [9], there are many learning systems that depend upon a good distance function to be successful. A variety of distance functions are available for such uses, including the Minkowsky, Mahalanobis, Camberra, Chebychev, Quadratic, Correlation, and Chi-square distance metrics; the Context-Similarity measure; the Contrast Model; hyperrectangle distance functions; and others.

The problem of service and resource matching (or measuring distance between service query and registered service profiles) is being actively discussed currently as a new challenging task for the next generation of annotation approaches to Web services and Web agents. A significant advantage is expected when using an ontological approach to semantically describe and query services. A matchmaking problem arises when a service is being queried and the query includes the distance measure between the required service description and the one from the service registry. We realized the need to analyze the applicability of different matchmaking methods to agent development tools implemented according to agent technology specifications such as FIPA (Section 2). We also consider three main groups of cases: matchmaking between classes of service profiles in pure taxonomies (Section 3); matchmaking between classes in faceted taxonomies (Section 4); and matchmaking between instances of faceted taxonomies (Section 5). Some samples of recent related work are given in Section 6. We conclude in Section 7.

2. FIPA matchmaking algorithm

One of the crucial components of a multi-agent system is a Registry to provide support for service registering and discovering. According to FIPA specifications [10], the Service Directory service is in charge of providing such functionality within the Agent System. The Service Directory service stores information of a service as an entry of its Service Description. The Directory Facilitator [11] is a reification of the Service Directory Service to provide a yellow page directory service for agents who have services to advertise. The Directory Facilitator operates with the Service Descriptions, which correspond to the structure of Fig. 1. The Service Description consists of the name of the service, its type, its supported interaction protocols, a list of ontologies, a list of content languages, the owner of the service, and a list of additional descriptive properties.

FIPA specifies a concrete matching criteria. The algorithm has to perform syntactic and structural matching based on a service template and a registered service description in the Directory Facilitator. The service template does not match the registered service description if:

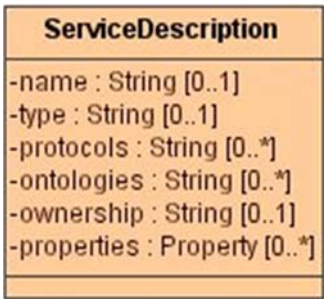

Fig. 1 Structure of Service Description

1. Any parameter of the service template does not exist in the registered service description, or,
2. Any parameter of the service template does not match to a corresponding parameter of the registered service.

A parameter of the service template matches a parameter of registered service if both names are equal and their values match.

For the Service Description it means that

- The name, type and ownership parameters match if their values are equal; and
- Protocols, ontologies and properties parameters match if each element of the set of the service template is matched by an element of the set of the registered service.

A possible service description as part of the Agent Description for Registration:

```
(service-description
          :name BusTicketBookingService1
          :type BusTicketBookingService
          :ontologies(set ServiceOntology)
          :properties(set
           (property
              :name Country
              :value Finland)
           (property
              :name typeOfConnection
              :value local)))
```

Service Template for matching:

```
(service-description
          :type BusTicketBookingService
          :ontologies (set ServiceOntology)
          :properties (set
           (property
              :name Country
              :value Finland)))
```

The main disadvantage is that the algorithm performs only syntactic matching and does not utilize even easy-to-implement and obvious possibilities of semantic matching. For instance, the algorithm checks only the syntactic case-insensitive equality of a string value of the type of the service template with type of the registered service. The algorithm takes into account only existing parameters and values in the service template, and does not take into account any information about semantic relation of the parameters nor their values defined in correspondent ontology. Thus the definition area of the algorithm is restricted by the possible combination of optional parameters, user defined properties, and the ranges of their values. As a result, the algorithm provides a fewer number of possible matching services because of the absence of analysis of the correspondent ontology. Last but not least, the algorithm is a binary function with "yes" and "no" answers instead of a more flexible approach of a distance measure.

Different characteristics of multi-agent systems influence the matching algorithms in the sense of what accessible information to utilize during the matching process. For instance, the Directory Facilitator is a registry of instances of services. Agents can register services in arbitrary Directory Facilitators. Instances of the service have a reference to ontology through the type of service parameter. But ontology does not have references to instances of classes of services. The absence of these references means that ontology cannot provide a number of services of some type. But matching algorithms can utilize numbers of instances to measure distances between classes more precisely.

3. Taxonomy-based distance measure

Having a service type in a search query, according to current FIPA implementations, we can compare a query string to service type[1] strings of available services. Such a comparison may lose efficiency in a large and highly distributed environment. However, new services will require additional new service types, and it will be quite complex to bind all services to a finite number of service types in order to save unambiguity and thus allow for search efficiency.

The simplest solution for a dynamic environment is to provide taxonomy of service types. The taxonomy provides its entities with a class-subclass relationship. This relationship is transitive, so if `classA` has subclass `classB` and `classB` has subclass `classC`, then `classC` is a subclass of `classA` also. Such a hierarchy will break the limitation in the number of service types and will save—and even gain—in search efficiency by providing unambiguous search. The simple example of service type taxonomy (see Fig. 2) can demonstrate the benefits of such approach.

The simplest hierarchy structure allows an arbitrary number of subclasses to be created, while saving the semantics of

[1] Service type refers to a class in an ontology.

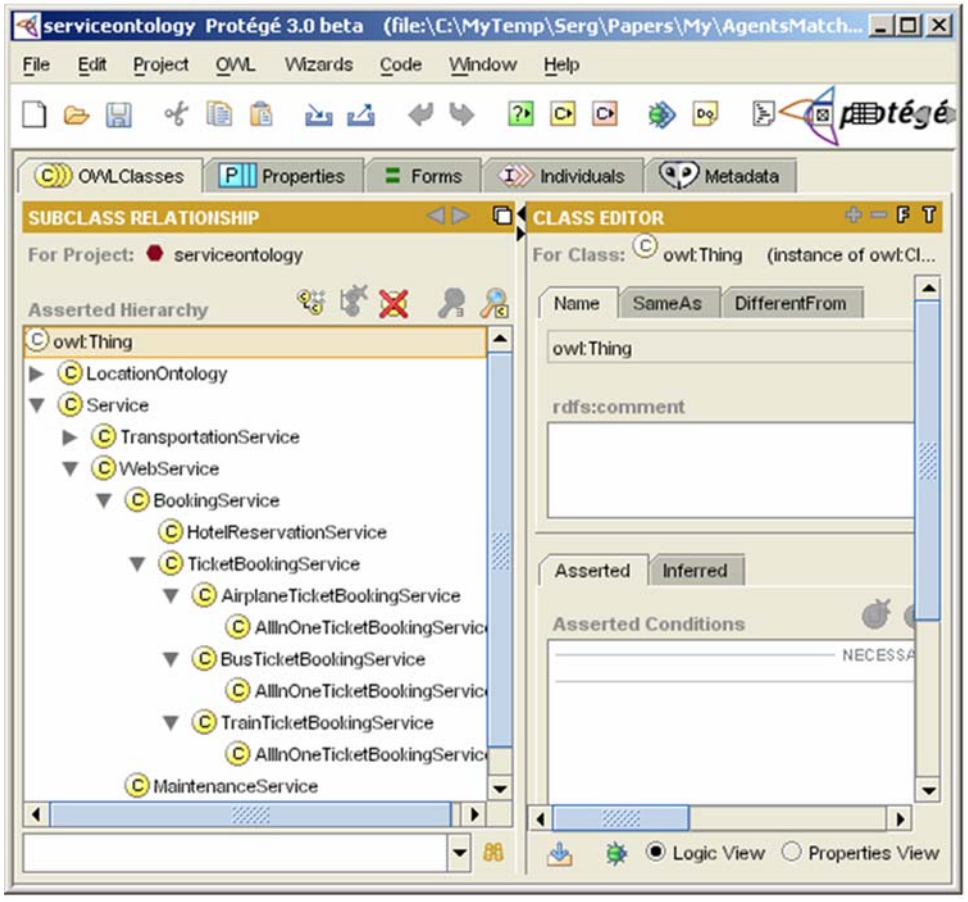

Fig. 2 Simple service hierarchy

the upper ones. For example, if we extend `TicketBookingService` class to four more specific subclasses, they all still belong to `TicketBookingService` yet their instances should be returned upon a search request looking for `TicketBookingService`. Of course it is possible to look for only direct instances of a certain class, but this limitation can be used for a more precise search. For example, if we search for only direct instances of `AirplaneTicketBookingService`, we will not receive instances of `AllInOneTicketBookingService` which could provide us with the facility we look for.

Information that an instance of a class is also an instance of all super classes of the class gives us the possibility to implement the simplest reasoning on taxonomy. For example, a search request,

```
(service-description
:type TicketBookingService
:ontologies (set ServiceOntology))
```

will not provide any response in the case when the FIPA matching algorithm is used and there are no registered services with such type in the Directory Facilitator.

But there are several services registered with types of subclasses (`BusTicketBookingService`, `TrainTicketBookingService`, etc.) of `TicketBookingService`. The Directory Facilitator has to return such services in response, as they are also a type of the requested service that is seen in Fig. 3.

It is easy to implement the functionality of the Directory Facilitator to perform reasoning over a taxonomy utilizing the class-subclass relationship among types of services. There are a number of open source libraries providing reasoning API for ontologies encoded in different languages. For instance, JENA is a JAVA library for the implementation of Semantic Web features. It can operate with RDF- and OWL-based ontologies. Concrete implementation of the matching algorithm has to create the JENA model of the ontology.

Then, instead of comparing syntactical equality of names of direct classes of services, JENA gets all the subclasses of a class of the requested service from the model and compares them further using the FIPA algorithm of syntactic match. This source code below allows the forming of a list of subtypes of a type of the requested service:

```
ArrayList subClasses = new ArrayList();
OntModel model = ModelFactory.createOntologyModel();
model.read("http://sample.domain/ServiceOntology.rdf","
RDF/XML");

OntClass requestedServiceClass =
model.getOntClass("http://sample.domain" +
"/ServiceOntology.rdf#TicketBookingService");

ExtendedIterator iterator =
requestedServiceClass.listSubClasses(false);

while ((iterator!=null)&&(iterator.hasNext())){

     OntClass subClass = (OntClass) iterator.next();

subClasses.add(subClass.getLocalName());}
```

There are two possibilities for a deployment of the subclass relation-based matching algorithm. The most appropriate one is to realize it as a part of the Directory Facilitator, as Fig. 4 illustrates.

Such an architectural solution requires changes in the FIPA specifications. But there is no need to change the protocol of request of agents to the Directory Facilitator. Basically the Directory Facilitator embeds the source code mentioned above to get a list of the types of services that are subclasses of the requested service type. Using the list of subtypes, the Directory Facilitator can perform the regular FIPA algorithm for the syntactic matching of type and subtypes of the requested service and for the types of the services registered in the directory. As a result, the Directory Facilitator responds to all services which have the requested type. Figure 5 shows a sequence diagram of the described interaction between the Agent, Directory Facilitator, and Ontology. The main functional characteristics are

- an agent forms one request with a service type,
- the Directory Facilitator requests a list of subclasses for the specified type of a service from the Ontology, then
- the Directory Facilitator performs internal iterations of the syntactic match over a list of requested types of services.

Main nonfunctional properties are

- an agent operates according to the regular protocol , and
- the response of the Directory Facilitator is now semantically full.

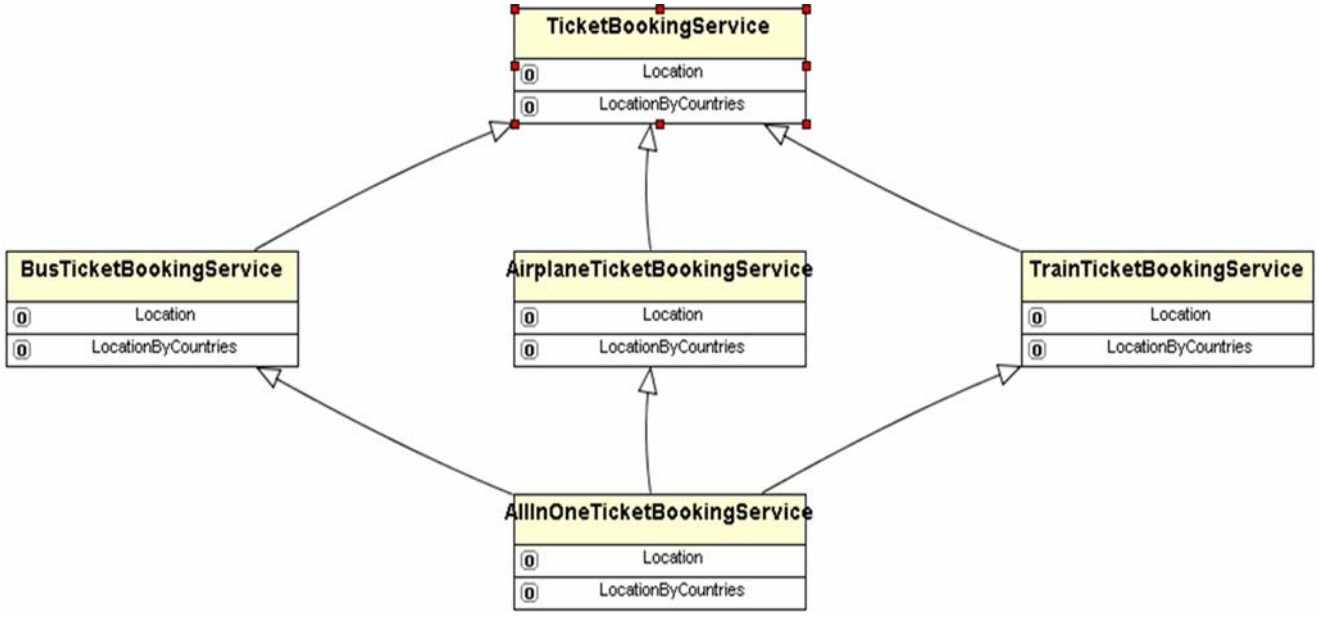

Fig. 3 Subclasses of `TicketBookingService`

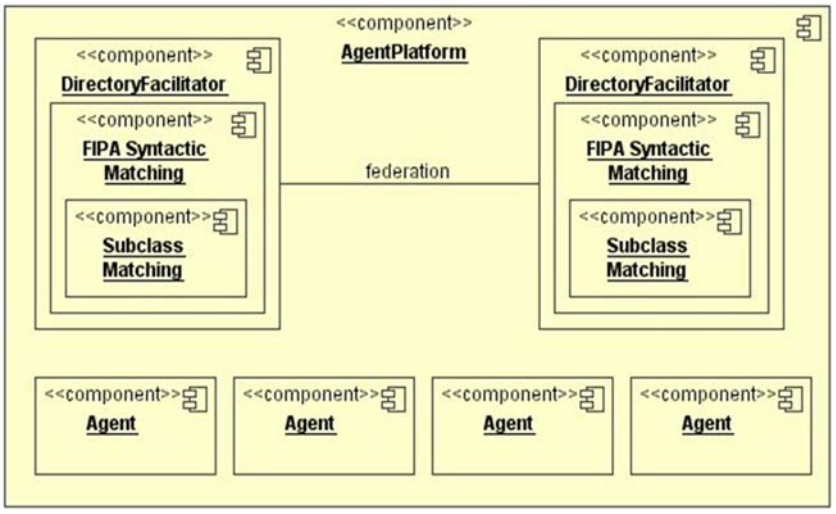

Fig. 4 Component diagram when the subclass matching component belongs to the Directory Facilitator

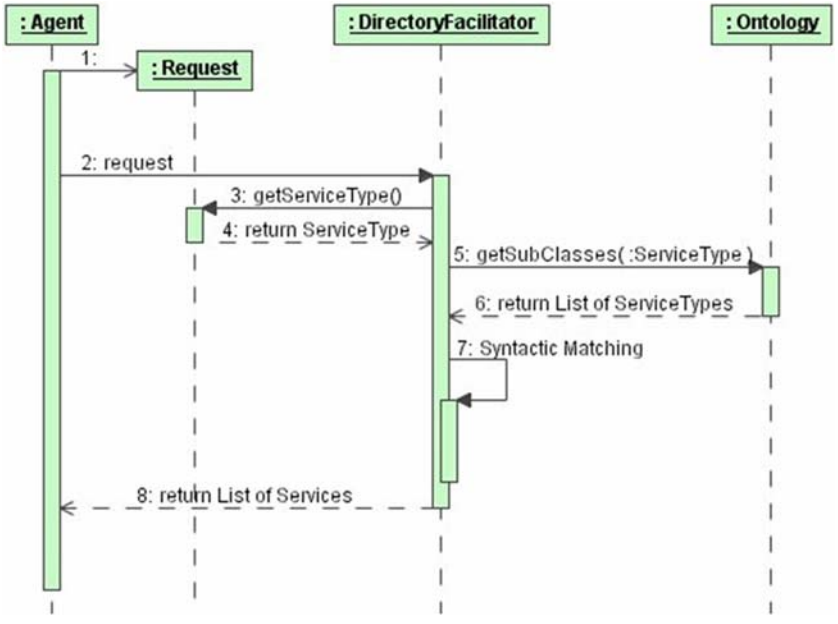

Fig. 5 Sequence diagram of an interoperation of the Agent, Ontology and Directory Facilitator

The solution should be used as a possible elaboration of FIPA specification. Implementing it out of the FIPA specification could cause a misunderstanding and a lack of interoperability of the enhanced Directory Facilitators with the existing FIPA compliant components.

The second approach is to leave the Directory Facilitator as is, while implementing a subclass matching as the Agent's functionality. Figure 6 depicts the architecture of this solution.

There is no need to change the FIPA specifications in this instance. The agents could utilize this functionality with the already existing implementations of the multi-agent systems. Figure 7 illustrates a sequence diagram of an interaction among the Agent, the Ontology and the Directory Facilitator for this architectural solution. The main functional characteristics are

- the agent requests a list of subclasses for the specified type of service from the Ontology,
- the agent iteratively forms requests over the list of requested types of services, then
- the Directory Facilitator provides a syntactic matching service according to the FIPA specification.

The agent collects the responses after the syntactic match. The main nonfunctional properties are

- if needed, the Agent can itself form a semantic matching,
- the Directory Facilitator stays unchanged, and
- inefficient network utilization.

In some cases, it is reasonable to return in a search result not only instances of the class being searched but also instances of classes close to that being searched. For example,

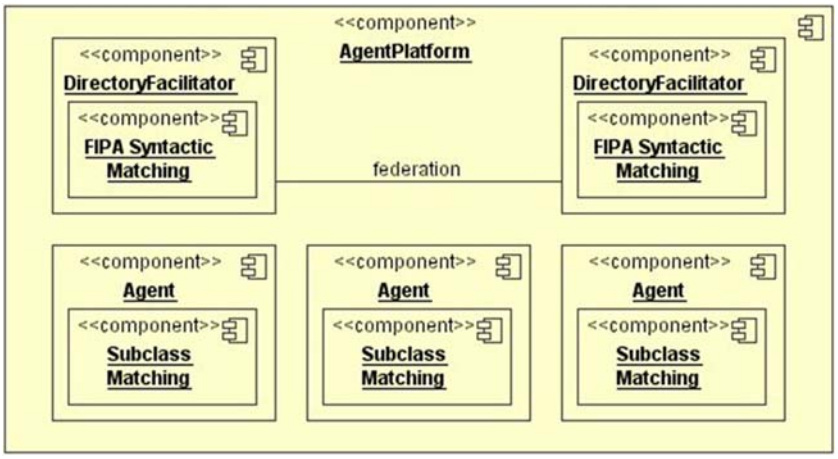

Fig. 6 Component diagram when the subclass matching is part of the Agent functionality

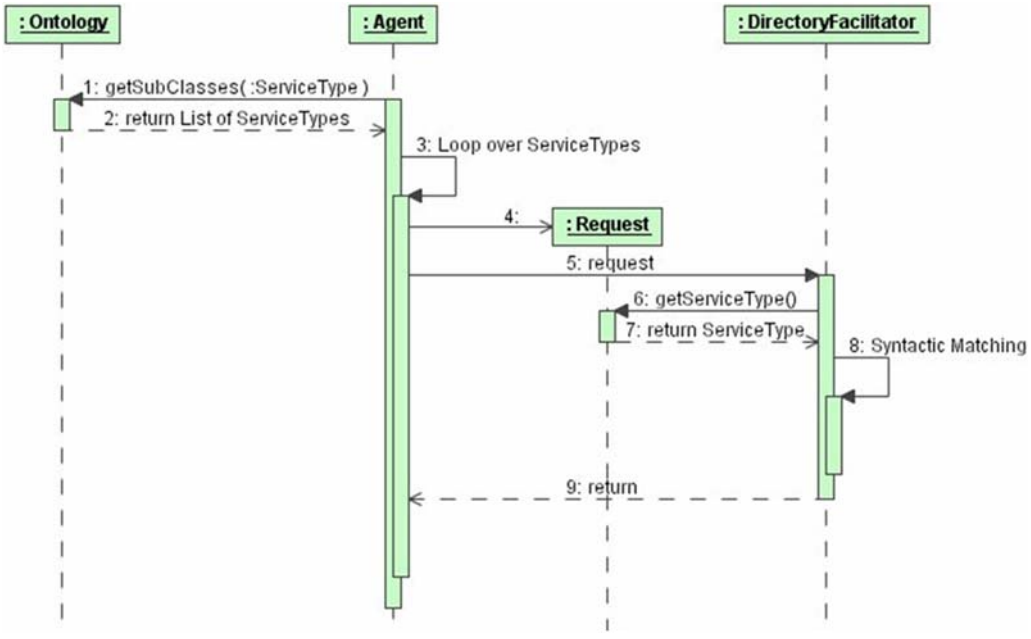

Fig. 7 Sequence diagram of the interoperation of the Agent, Ontology and Directory Facilitator

if we are looking for `BusTicketBookingService` and no instances are available, then, probably, the user wouldn't mind getting instances of a sibling service, `TrainTicketBookingService`, as a result. Considering a general case, we need to provide a distance metric to select the closest classes for response when there are no instances of a requested type.

One of the metrics could reflect generalization of classes. The utilization of this semantic relation depends on the interests of a domain and a task. For a search of some service type, the answer can contain instances of a superclass in the case where there are no instances of the requested service class. It means, for example, that on a request for the `BusTicketBookingService`, the user may get instances of `TicketBookingService`, which are all instances of the other subclasses of `TicketBookingService`. If for some domain or task such functionality is reasonable, then the distance between the classes is 0 in the case of a subclass relation and 1 for each instance of a superclass relation. For instance, Fig. 8 shows that `TicketBookingService` has the distance of 0 to all its subclasses, the distance of 1 to the direct superclass `BookingService`, the distance of 2 to `WebService`, and the distance of 3 to Service. According to the distance measure described above, the implementation of a matching algorithm based on the subclass relation is a particular case of the distance measure based on the subclass and superclass relations.

The implementation of the matching algorithm based on the generalization of classes requires changes of the FIPA specifications. FIPA defines an object of the search constraints to limit the function of searching within a directory. The object consists of three parameters: `max-depth`, the

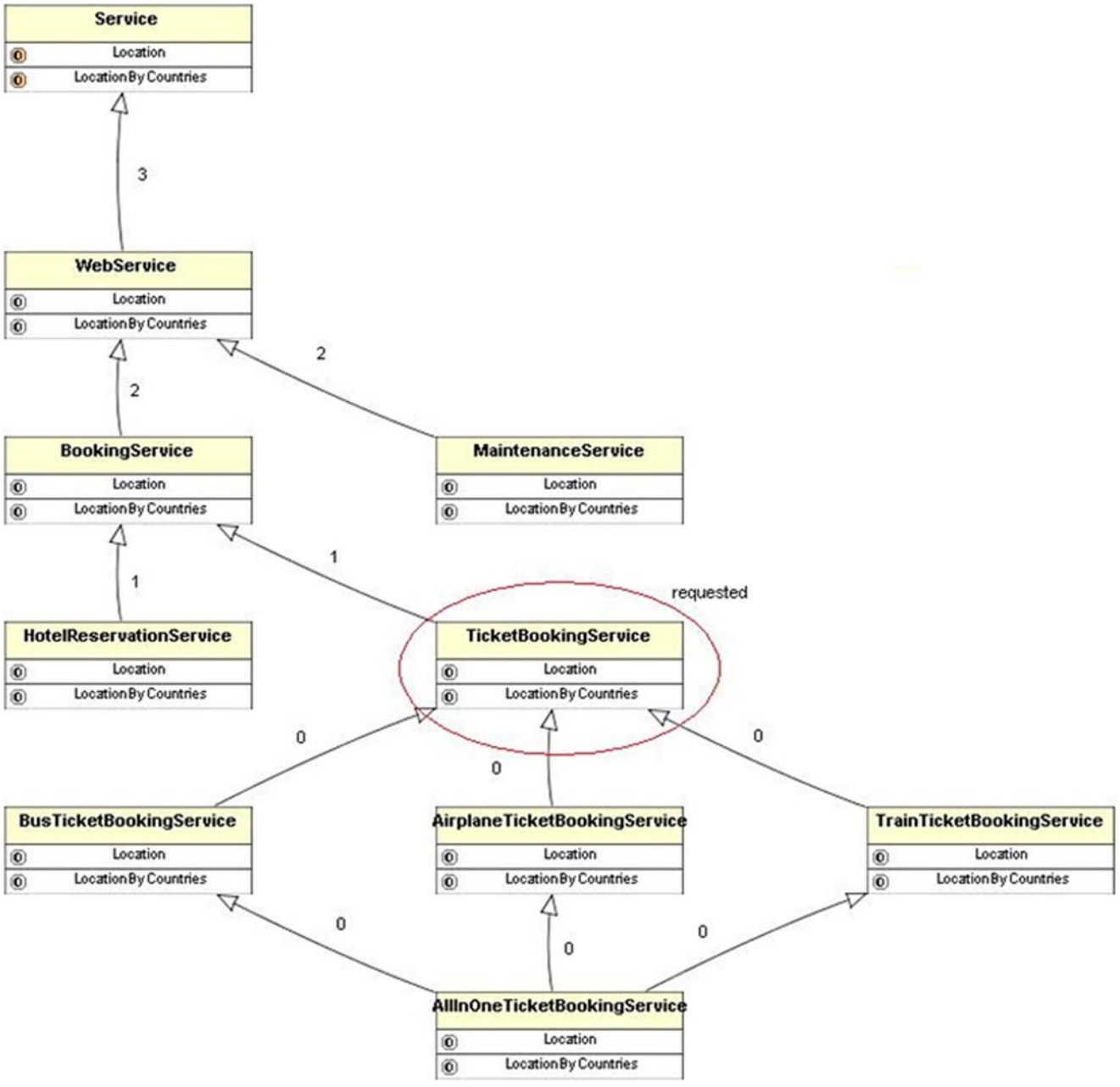

Fig. 8 Distances between `TicketBookingService` and other services

maximum depth of propagation of the search to the federated directories; `max-results`, the maximum number of results to return for the search; and `search-id`, a globally unique identifier for a search. An agent has to pass an additional parameter, `max-distance`, to support the implementation of the generalization-based matching algorithm. `Max-distance` is a positive integer, including zero, that defines an expansion of the search over the ontology. The value 0 requires services which are of the type of the requested class and its subclasses. The value 1 includes the direct superclasses and their subclasses in consideration. Generally, the value of the `max-distance` defines the level of the superclass to which search could be extended if there are no services on lower levels.

The federation of directories adds a complexity to the implementation of this algorithm because the directory has to meet the requirements of the `max-results` parameter of the search constraints. One of the possible solutions is to collect the matching services of the farthest classes in respect to `max-distance` parameter. Then the search is propagated with respect to the `max-depth` parameter while integrating the resulting sets of services from the previous

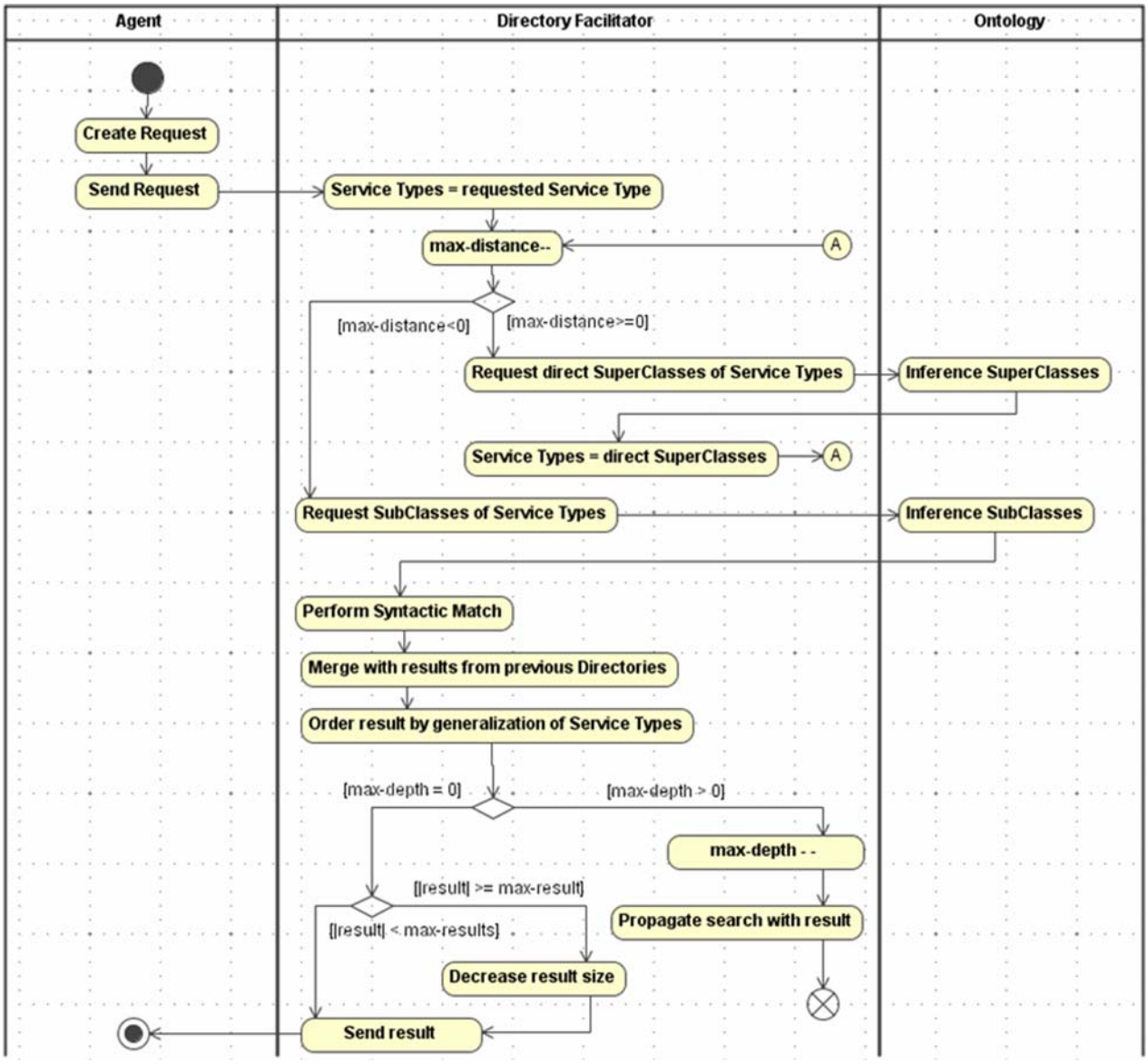

Fig. 9 Activity diagram of a matching algorithm based on generalization distance

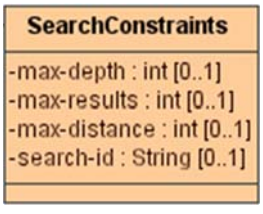

Fig. 10 Extended structure of the Search Constraints

directories. The response consists of a `max-result` or a smaller number of the most specialized services. Figure 9 illustrates the algorithm described above.

Figure 10 shows the extended structure of the object of the search constraints. Architectural considerations are the same as for the matching algorithm based on the subclass relation.

The Object Match algorithm [12] is more sophisticated. It calculates the similarity of two concepts based on a hierarchy of concepts. The algorithm uses the concept of upwards cotopy (UC) defined as follows:

$$UC(O_j, H) = \{O_j \mid H(O_i, O_j) \cup O_j = O_i\} \quad (1)$$

where taxonomy is given by an irreflexive, acyclic, transitive relation class-subclass H and O is a class in a taxonomy.

The intersection of upwards cotopies of two classes is

$$I(O_1, O_2, H) = UC(O_1, H) \cap UC(O_2, H) \quad (2)$$

The union of upwards cotopies of two classes is

$$U(O_1, O_2, H) = UC(O_1, H) \cup UC(O_2, H) \tag{3}$$

And according to the Object Match algorithm, the similarity of two classes equals to:

$$S(O_1, O_2, H) = \frac{|I(O_1, O_2, H)|}{|U(O_1, O_2, H)|} \tag{4}$$

The algorithm reflects the fact that more specialized neighbors are closer to each other than the more general ones. Basically, the algorithm takes into account the quantity of common and different parents with a sensitivity to the depth of classes within the taxonomy.

This algorithm could be enhanced by taking into account the width as well as the depth of the taxonomy. Similarity is defined as follows:

$$S_e(O_1, O_2, H) = \frac{\sum_{O_i \in I(O_1, O_2, H)} |\{O_j | H'(O_i, O_j) \cup O_i = O_j\}|}{\sum_{O_i \in U(O_1, O_2, H)} |\{O_j | H'(O_i, O_j) \cup O_i = O_j\}|} \tag{5}$$

where H' is a relation class—direct subclass.

Let us consider an example of the abilities of the algorithm to reflect the depth and width of taxonomy. Figure 11 shows three hierarchies of services.

We have to calculate the upwards cotopies in order to measure the distance between D and E in the first hierarchy H_1, using formula 1:

$$UC(D, H_1) = \{D, B, A\} \tag{6}$$

$$UC(E, H_1) = \{E, B, A\} \tag{7}$$

The intersection and union of the cotopies are calculated using formulas 2 and 3:

$$I(D, E, H_1) = \{B, A\} \tag{8}$$

$$U(D, E, H_1) = \{D, E, B, A\} \tag{9}$$

The similarity according to formula 4 equals

$$S(D, E, H_1) = \frac{|\{B, A\}|}{|\{D, E, B, A\}|} = \frac{1}{2} \tag{10}$$

The enhanced similarity by formula 5 is equal to

$$S_e(D, E, H_1) = \frac{|\{D, E, F, B\}| + |\{B, C, A\}|}{|\{D, E, F, B\}| + |\{B, C, A\}| + |\{D\}| + |\{E\}|} = \frac{7}{9} \tag{11}$$

The second hierarchy H_2 extends the first hierarchy H_1 by introducing a new subclass of B, thus raising the width of the hierarchy. The upwards cotopies for D and E in such a case are the same as in the first hierarchy. The intersection and union of cotopies are also the same. The value of similarity of D and E by formula 4 stays unchanged. The enhanced similarity shows that D and E are semantically closer in the second hierarchy:

$$\begin{aligned} S_e(D, E, H_2) &= \frac{|\{G, D, E, F, B\}|}{|\{G, D, E, F, B\}| + |\{B, C, A\}| + |\{D\}| + |\{E\}|} \\ &\quad + \frac{|\{B, C, A\}|}{|\{G, D, E, F, B\}| + |\{B, C, A\}| + |\{D\}| + |\{E\}|} \\ &= \frac{8}{10} \end{aligned} \tag{12}$$

The calculation of the similarity of D and E for the third hierarchy H_3 is analogical to the calculation for the first hierarchy. The similarities by formula 4 and the enhanced formula 5 are equal to:

$$S(D, E, H_3) = \frac{|\{B, A, G\}|}{|\{D, E, B, A, G\}|} = \frac{3}{5} \tag{13}$$

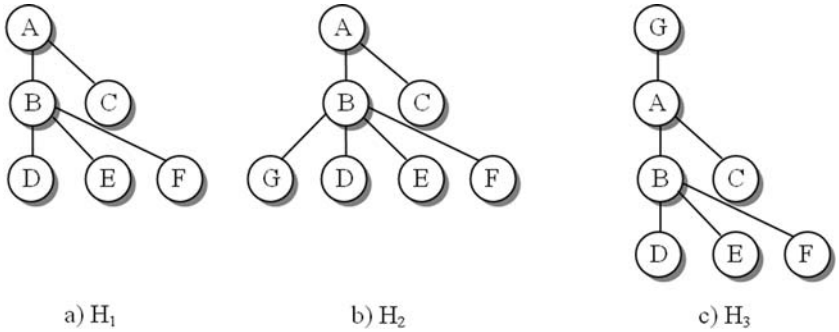

Fig. 11 Example on Hierarchies

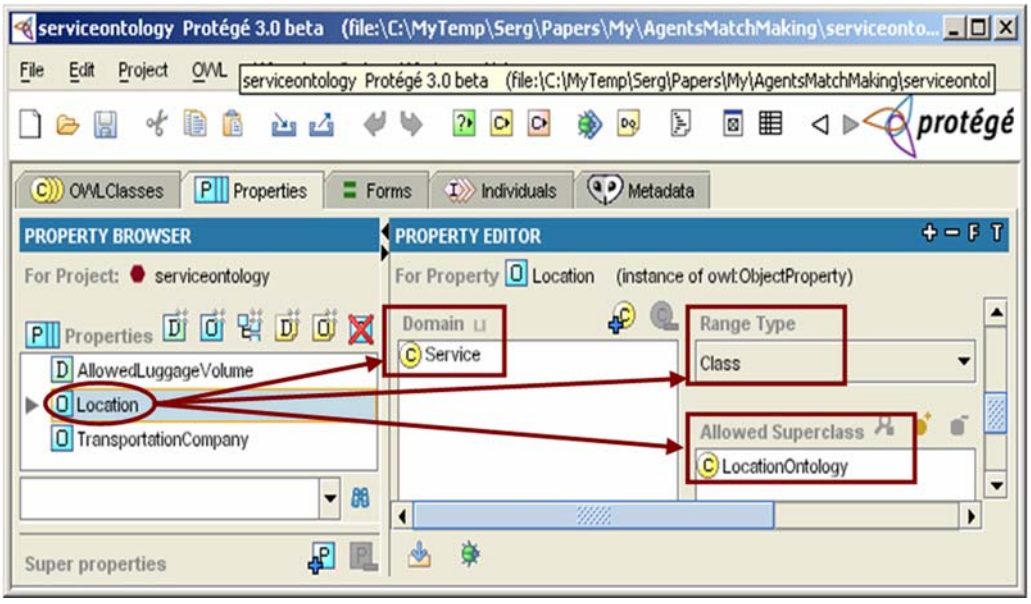

Fig. 12 Property domain and range definition in Protégé

$$S_e(D, E, H_3) = \frac{|\{D, E, F, B\}| + |\{B, C, A\}| + |\{A, G\}|}{|\{D, E, F, B\}| + |\{B, C, A\}| + |\{A, G\}| + |\{D\}| + |\{E\}|} = \frac{9}{11} \quad (14)$$

As we can see, D and E are semantically closer by both formulas 4 and 5 in the third hierarchy as compared to the first one.

4. Distance measure for ontology with facets

Faceted Classification [13] is a sophisticated alternative to the traditional classification schemes and modern web directories, which put one item in only one place. Faceted classifications are based on the Colon Classification scheme of Indian library scientist S.R. Ranganathan developed in the 1930s. Ranganathan created a set of properties or characteristics or attributes of any subject, ideally mutually exclusive (orthogonal) and exhaustive (complete), which means that any object being classified could be assigned one of these descriptions, which he called a facet. The outstanding advantage is that a hierarchy can be built starting with the facet considered as the most important. A facet-based classification has much more extended capabilities in comparison to a taxonomy-based one. In ontological modeling it is quite usual for a certain hierarchy to be selected as a backbone for the whole Ontology, which is then powered by additional properties assigned to the classes. The main hierarchy consists of the objects representing the goal of a description. For example, if we consider services as our description goal, then we can organize them in the main class-subclass relationship by their service type and then augment with additional properties, such as location, type of payment accepted, etc.

For example, we can add the property `Location` to the `BookingService` class (Fig. 12).

We then provide the property `Location` with `domain` and `range` characteristics. The `domain` defines the set of classes allowed to have this property and the `range` defines the allowed values for the property.

In our case, we set the `domain` value to the `Service` class. This means that any instance of a `Service` class or its subclasses may have the property `Location`. As a range value type we set `Class`, and as an allowed superclass for value—`LocationOntology`. In this way we define a controlled vocabulary for the property values. A *controlled vocabulary*, or managed vocabulary, is an attempt to limit the number of terms that will be admitted into a discourse in order to improve communication. However, the applicability of a controlled vocabulary in the quickly changing environments of P2P is still a matter of discussion and research.

When comparing classes with properties, the distance can be measured based on a number of discriminate properties. The more identical properties the classes have, the more similar classes are. The simple distance measure for two classes, A and B with facets n_A and n_B, can be calculated by formula:

$$D = \frac{|n_A \cap n_B|}{|n_A \cup n_B|} \quad (15)$$

So we divide the number of identical facets into the number of facets used for the description of both classes (identical facets of both classes are counted only once).

For example, if class A has 5 facets $\{f1, f2, f3, f4, f5\}$ and class B has 7 facets $\{f1, f2, f3, f6, f7, f8, f9\}$, then distance can be calculated as:

$$\begin{aligned} &|n_A \cap n_B| = |\{f_1, f_2, f_3\}| = 3 \\ &|n_A \cup n_B| = |\{f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9\}| = 9 \\ &D = \frac{3}{9} = \frac{1}{3} \end{aligned} \tag{16}$$

5. Distance measure between instances in ontology

Faceted classification schemes may vary. The limited set of allowed values (controlled vocabulary) for certain facet provides the ability to measure distance between two instances having the same facet. If the allowed values are arranged in taxonomy, it extends the distance measure precision. However, property (attribute) values can be from non-controlled vocabulary, which means that only a Boolean match can be applied for returning 1 if values are equal and 0 if not. If the values are numerical then the distance can be measured using a standard measure (e.g. Euclidean distance) however values obtained should be normalized.

When an instance of a certain class is requested and can not be found (e.g. we ask for the implementation of an online train ticket service in Finland but there is no service currently running), then it makes sense to return a bus ticket service located also in Finland. Hence, we need to deal with instances and assigned property values.

In our opinion, similarity (or distance) between instances can be measured only using common properties and, thus, their comparable values. However, the closest class to class of instance should be found first. After the closest class is found, its instances can be taken and the distance can be measured based on different metrics.

Assume that we have two interpreted profiles with the same set of attributes, which have numerical or nominal values. Assume also that the first profile is taken from the "service requests" database and the second one from the "service offerings" database. The distance between these two profiles can be measured according to [14, 15] as follows:

$$E(X, Y) = \sqrt{\sum_{\forall i, x_i \in X, y_i \in Y} \omega_i \cdot d(x_i, y_i)^2} \tag{17}$$

where X and Y are two vectors of the values of the attributes of the two profiles. The component distance $d(x_i, y_i)$ for every attribute is normalized by the range of the previously known values of the attribute so that it is mostly within the range [0, 1], and weighted by weights ω_i according to the importance of the attribute. The weight ω_i may be the probability that a client, whom a request (interpreted profile) belongs to, will not be satisfied by an offering (interpreted profile of the same structure) if the i-th attribute of these profiles is not taken into account.

The Heterogeneous Euclidean-Overlap Metric is used for both nominal and numerical features. This function defines the component distance between two values of an attribute as:

$$d(x_i, y_i) = \begin{cases} \text{if } i\text{-th attribute is nominal} - \begin{cases} 0, & \text{if } x_i = y_i \\ 1, & \text{otherwise} \end{cases} \\ \text{else} : \dfrac{|x_i - y_i|}{range_i} \end{cases} \tag{18}$$

where $range_i$ is the range of the attribute i.

If the attribute is nominal but has values from the controlled vocabulary organized in taxonomy, then the distance between the two attribute values can be measured using distance measure techniques applicable to taxonomy, as described in Section 3. So the distance can be defined as:

$$d(x_i, y_i) = \begin{cases} \text{if } i\text{-th attribute is nominal} - \text{distance} \\ \quad \text{between ontology concept } x_i \text{ and } y_i \\ \text{else} : \dfrac{|x_i - y_i|}{range_i} \end{cases} \tag{19}$$

The Interpolated Value Difference Metric defines the following component distance between the two values of an attribute:

$$d(x_i, y_i) = \begin{cases} \sqrt{\displaystyle\sum_{j=1}^{k} |P(j \mid i \in [x_i, x_i + \Delta]) - P(j \mid i \in [y_i, y_i + \Delta])|^2}, \\ \quad \text{if } i \text{ numerical (continuous) attribute} \\ \sqrt{\displaystyle\sum_{c_i=1}^{k} |P(j \mid i = x_i) - P(j \mid i = y_i)|^2}, \quad \text{otherwise,} \end{cases} \tag{20}$$

where k is the number of classes of profiles; $P(j \mid i = x_i)$ is the conditional probability that a profile belongs to class j if its attribute i has the value x_i, and $P(j \mid i \in [x_i + \Delta])$ is the interpolated conditional probability that a profile belongs to class j if its discreted attribute i has the value x_i, and Δ is the discretization step.

Table 1 Services separated by facet values

Object	Quantity
Web Services Total	10
`PlaneTicketBookingService`	3
`TrainTicketBookingService`	2
`BusTicketBookingService`	5
Services accepting Web money	4
Services not accepting Web money	6
`PlaneTicketBookingService` *accepting Web money*	1
`TrainTicketBookingService` *accepting Web money*	1
`BusTicketBookingService` *accepting Web money*	2
`PlaneTicketBookingService` *not accepting Web money*	2
`TrainTicketBookingService` *not accepting Web money*	1
`BusTicketBookingService` *not accepting Web money*	3

A more simple interpretation of the previous formula might be:

$$d(x_i, y_i) = \begin{cases} \sqrt{\sum_{c=1}^{C} [P(c|x_i) - P(c|y_i)]^2}, \\ \quad \text{if } i\text{-th attribute is nominal;} \\ \dfrac{|x_i - y_i|}{range_i}, \quad \text{if } i\text{-th attribute is numerical.} \end{cases} \tag{21}$$

A probabilistic approach requires prior knowledge of the total number of instances of a certain class but, in the P2P environment, it may not always be easy to obtain statistics about instances and range values of numerical attributes. But by having this data the calculations gain in accuracy. For example, we have a preference to find a train ticket booking service that accepts Web money (payment via the Internet) and has a service load of not more than 60%. The importance of service load is estimated as 0.2 and importance of transport type (train, bus or plane) is 0.8 respectively. To calculate the distance to the closest service profile we need the following data:

– The total number of ticket booking service instances and the number of service instances of each class;
– The total number of service instances accepting Web money and the number of service instances of each class that accept Web money;
– the service load value.

For example, we have 10 ticket booking service instances, among them:

– three instances of `PlaneTicketBookingService`
– two instances of `TrainTicketBookingService`
– five instances of `BusTicketBookingService`

Among our 10 service instances only 4 accept Web money.

The four instances accepting Web money comprise a `PlaneTicketBooking Service`, a `TrainTicketBookingService` and two instances of `BusTicket BookingService`. The six remaining instances are: two instances of `PlaneTicketBookingService`, one `TrainTicketBookingService` and three instances of `BusTicketBookingService`. The summary is provided in Table 1.

Let's consider two service profiles (Table 2).

In order to calculate the distance from our preference to the Service Profiles, we need conditional probability values calculated according to Eq. (12) (see Table 3).

Table 2 Service profiles attribute values

	Service Profile 1	Service Profile 2
ServiceType	`TrainTicketBookingService`	`BusTicketBookingService`
Load	80%	20%

Table 3 Conditional probability values for the calculation of the distance between `TrainTicket-BookingService` and `BusTicketBooking-Service` values

Probability	Value
P(*Accepts Web money* | *ServiceType* = `TrainTicketBookingService`)	1/2
P(*Accepts Web money* | *ServiceType* = `BusTicketBookingService`)	2/5
P(*Not accepts Web money* | *ServiceType* = `TrainTicketBookingService`)	1/2
P(*Not accepts Web money* | *ServiceType* = `BusTicketBookingService`)	3/5

So the distance can be calculated as:

$$d(\text{“}Train\text{”}, \text{“}Bus\text{”}) = \sqrt{\begin{pmatrix} (P(AcceptsWM \mid ServiceType = Train) \\ - P(AcceptsWM \mid ServiceType = Bus))^2 \\ + (P(NotAccWM \mid ServiceType = Train) \\ - P(NotAccWM \mid ServiceType = Bus))^2 \end{pmatrix}} \tag{22}$$

Substituting the values in Eq. (22), we have:

$$d(\text{“}Train\text{”}, \text{“}Bus\text{”}) = \sqrt{\left(\frac{1}{2} - \frac{2}{5}\right)^2 + \left(\frac{1}{2} - \frac{3}{5}\right)^2} \approx 0,141 \tag{23}$$

Now we can calculate the distance of our preferences to Service Profiles. Here we have to mention that the distance of attribute having numerical value as a marginal value should be calculated with an additional condition. In our case, we assume that “60% load” parameter should be *not more* than 60%, hence the distance can be calculated as:

$$d(x_i, y_i) = \begin{cases} 0, & \text{if } x_i \le y_i \\ \dfrac{|x_i - y_i|}{range_i}, & \text{otherwise,} \end{cases} \tag{24}$$

where $range_i = \max(y_i) - y_i$

The maximum range value for the Service Load attribute is 100 because its value is a percentage.

Now, the distance to the Service Profiles is calculated using Eq. (17).

$$D(UserP, SP1) = \sqrt{0.8 \cdot 0 + 0.2 \cdot \left(\frac{80 - 60}{100 - 60}\right)^2} = 0.1$$
$$D(UserP, SP2) = \sqrt{0.8 \cdot (0.141)^2 + 0.2 \cdot 0} \approx 0.126 \tag{25}$$

where $D(UserP, SP1)$ is a distance between User Preferences and Service Profile 1 and $D(UserP, SP2)$ is a distance between Preferences and Service Profile 2 respectively.

As we can notice from the equations above, it is possible to vary the weights so that the initially preferred `TrainTicketBookingService` might be farther from the `BusTicketBookingService` because of undesirable attribute values.

6. Related work

The application of semantic technology in different distributed environments is an open issue for many research projects. Peer service discovery and matching, based on semantic profiles described in [16], provides an ontology-based matching and a distance measure based on the shortest path between ontology nodes. However, the full power of the ontology is not used and more attention should be paid to overall system architecture and interoperability of nodes.

Another recent activity [17], aimed at service discovery within a grid environment, proposes a similarity metric for measuring the distance between a service request and service descriptions available in the registry. The metric is based on a similarity function which is a weighted sum of matching attributes, description and metadata. The weights are calculated as probabilistic functions.

A Semantic Similarity Measure for Semantic Web Services is proposed in [18]. The formula for semantic similarity is defined as follows: $sim(a, b) = \frac{f_{common}(a,b)}{f_{desc}(a,b)}$, where f_{common} is the common function measuring the information value of the description that is shared between a and b, and f_{desc} is the description function giving the value of the total information content of a and b. But the key feature is how the functions f_{common} and f_{desc} are calculated. Deep analysis is done towards the formation of the description sets. The authors use OWL Lite as a descriptive language and give examples of the applicability on OWL-S descriptions.

7. Conclusions

In this paper, we tried to analyze the applicability of the ontology-based models for the improvement of the searching capabilities in Agent Systems. Firstly, we have demonstrated the drawbacks of the matching algorithm of the Directory Facilitator, compliant with the FIPA specification [11]. The algorithm responds to an incorrect set of services from the ontological point of view because of an instance of a class is also an instance of all superclasses of this class. Secondly, we demonstrated through the examples that matching algorithms based on a distance or similarity measure are more flexible and appropriate in the task of a services search because they provide responses even if an exact match does not exist. With a matching algorithm based on distance measure, there is a possibility for a user to prepare and efficiently execute requests based on uncertain or incomplete information.

The main conclusion is that the Agent, Grid Services, and Web Service technologies can be effectively integrated with each other. However such integration requires correct ontology-based matching tools, which can be considerably improved by similarity measure methods. Analysis shows that some of the algorithms can be easily implemented

without radical changes within the existing tools and standards while giving more sophisticated results of matching.

Although an elaboration of the existing standards requires deeper analysis of the quality characteristics of the distance measure-based matching algorithms, we consider these changes as inevitable.

Acknowledgment This research has been supported by the "Proactive Self-Maintained Resources in Semantic Web" (SmartResource) project, funded by TEKES and the industrial consortium of Metso Automation, TeliaSonera, and TietoEnator.

References

1. Kaikova H, Khriyenko O, Kononenko O, Terziyan V, Zharko A (2004) Proactive self-maintained resources in semantic web. Eastern-European J Enter Technol, 2(1):4–16, ISSN: 1729-3774
2. SmartResource project, http://www.cs.jyu.fi/ai/OntoGroup/Smart–Resource_ details.htm
3. Adaptive Services Grid, Integrated project supported by the European Commission, http://asg-platform.org/
4. FIPA, Foundation for Intellegent Physical Agents, http://www.fipa .org/
5. Tailor C, Tudhope D (1996) Semantic closeness and classification schema based hypermedia access. In: Proceedings of the 3-rd International Conference on Electronic Library and Visual Information Research (ELVIRA'96), Milton, Keynes
6. Brooks T (1995) Topical subject expertise and the semantic distance model of relevance assessment. J Doc, 51(4):370–387
7. Foo N, Garner B, Rao A, Tsui E (1992) Semantic distance in conceptual graphs. In: Gerhotz L (ed) Current directions in conceptual structure research. Ellis Horwood, pp 149–154
8. Rada R, Mili H, Bicknell E, Blettner M (1989) Development and application of a metric on semantic nets. IEEE Tran Sys, Man, and Cybernetics 19(1):17–30
9. Wilson D, Martinez T (1997) Improved heterogeneous distance functions. J Art Intell Rese 6:1–34
10. FIPA Abstract Architecture Specification, http://fipa.org/specs/ fipa00001/
11. FIPA Agent Management Specification, http://fipa.org/specs/fipa 00023/
12. Stojanovic N, Meadche A, Staab S, Studer R, Sure Y (2001) SEAL: A framework for developing semantic PortALs. In: Proceedings of the International Conference on Knowledge Capture, Victoria, British Columbia, Canada, ACM Press. pp 155–162
13. Glossary of Content Management Professionals, http:www. cms-glossary.com/
14. Cost S, Salzberg S (1993) A weighted nearest neighbor algorithm for learning with symbolic features. Mach Learn 10(1):57–78
15. Puuronen S, Tsymbal A, Terziyan V (2000) Distance functions in dynamic integration of data mining techniques. In: Dasarathy BV (ed) data mining and knowledge discovery: Theory, tools and technology II, Proceedings of SPIE, vol. 4057, The Society of Photo-Optical Instrumentation Engineers, USA, pp 22–32
16. Haase P, Agarwal S, Sure Y (2004) Service-oriented semantic peer-to-peer systems, Lecture Notes in Computer Science, vol. 3307, pp 46–57
17. Ludwig SA, Reyhani SMS (2005) Semantic approach to service discovery in a grid environment. J Web Semant, 3(4) Elsevier
18. Hau J, Lee W, Darlington J (2005) A Semantic Similarity Measure for Semantic Web Services, Web Service Semantics Workshop

Vagan Terziyan Professor in Software Engineering since 1994 and the Head of the Artificial Intelligence Department since 1997 in Kharkiv National University of Radioelectronics (Ukraine). Currently he is working in Agora Center (University of Jyvaskyla, Finland) as a project leader of the SmartResource TEKES Project and Head of Industrial Ontologies Group. He is a member of IFIP WG 12.5 ("Artificial Intelligence Applications"), PC Chair of IFIP International Conference on Industrial Applications of Semantic Web. His research and teaching profile is design of Intelligent Web Applications, which utilise and integrate emerging Knowledge-, Agent-, Machine-Learning- and Semantic Web- Based technologies and tools. For more details please refer to his homepage at http://www.cs.jyu.fi/ai/vagan/

Sergiy Nikitin (1982) is currently working as a researcher in Agora Center (University of Jyväskylä, Finland) on the SmartResource TEKES project. Sergiy graduated form the Kharkiv National University of Radioelectronics (KNURE), Ukraine with the Engineer's degree in 2004. At the end of the same year he obtained a Master of Science degree from the University of Jyväskylä, Finland. In 2005 he entered a Ph.D.-program at the Faculty of Information Technology of the University of Jyväskylä. Among other research activities Sergiy has taken part in Adaptive Services Grid (ASG) and SCOMA projects.

His research interests include semantic web services, semantic configuration, agent technology and ontology engineering areas. Sergiy Nikitin has been a member of Industrial Ontologies Group since 2004. For more details please refer to his homepage at http://www.cc.jyu.fi/ senikiti

Anton Naumenko (1980) is a researcher at the Department of Mathematical Information Technology, University of Jyväskylä, Finland (JYU). He has Master of Science degrees in Information Technology from the JYU and in Computer Science from the Kharkov National University of Radioelectronics, Ukraine (KhNURE). He also earned a Bachelor of Economics degree from the KhNURE. In 2004, he has started his postgraduate education in the Faculty of Information Technology, JYU. Anton's professional career started in the KhNURE with a position of technician (2000) in a research project, where he became a researcher and leaded database designers and software developers (2001–2002). He was a leader of a software development team and a researcher in the SmartResource project (2004). After that he participated in the Adaptive Services Grid (2005) and Mobile Design Patterns and Architectures (2005–2006) research projects. His expertise and interests consist of Semantic Web, Agent Technologies, Object-Oriented Design, and Access Control. For more details please visit his homepage at www.cc.jyu.fi/ annaumen.