Abstract

Matching algorithms for Semantic Web Services discovery focus on the functional aspects of the services, which are usually given by the function that transforms inputs into outputs. Current approaches do not take into attention the fact that some services may not be available around the clock and, more restrictively, may require exclusive access. In this paper we present an approach to allow the discovery of Web Services based on properties and temporal restrictions. We present a matching algorithm which leverages the work reported in the literature so as to consider availability matching. Throughout the paper, we illustrate our approach in the context of Web Services related to remote experiments, which may present rigid constraints in terms of both temporal restrictions and availability of resources.

1 Introduction

Semantic Web Services demand the automation of tasks such as service discovery, invocation and composition. In particular with respect to discovery, important research has been carried out toward exploiting information made available by means of the WSDL and UDDI specifications. For instance, the literature reports efforts toward matching parameters in SOAP messages [10], in WSDL and UDDI descriptions using information retrieval and structure matching [17], or matching UDDI descriptions related to distributed directories [1], or matching schema trees [2] and service descriptions [5, 15]. Overall, authors have been exploring the identification of service providers that comply with application requirements by themselves or by composition with other provided services.

The Ontology Web Language for Services (OWL-S) allows the specification of rich semantics related to Web Services, contributing to the definition and to the automation of service provision and usage, and supporting the construction of powerful tools [9].

The service profile defined by OWL-S provides a super-class for a high-level description of services. The profile focuses on two aspects of the service functionality: (a) inputs and outputs, representing information transformation; (b) preconditions and effects, representing state changes produced by the execution of the service. They are known as IOPE (inputs, outputs, preconditions and effects). The profile also allows the description of information that specifies the category of a given service.

Particularly in the context of OWL-S, discovery algorithms focus on functional aspects of the services, which are usually given by the transformation from a set of inputs to a set of outputs. Discovery strategies usually do not take into account that some services may not be available around the clock or may require exclusive access. Such service access restrictions may impose some requirements such as advanced booking, for instance. The information regarding the category of a service, which may be seen as a property of the service in the OWL-S specifications, may be used to identify the services that need to be previously booked.

In this paper we present an approach to allow the discovery of Web Services based not only on service categories or service functionalities, but also on temporal restrictions. We use the Time ontology [3] to formalize temporal concepts and to present a matching algorithm which leverages the work reported in the literature, so as to consider availability matching. Throughout the paper we illustrate our approach in the context of Web Services related to Remote Experiments – a category of applications which is representative of those demanding access restrictions.

The remainder of this paper is structured as follows. In Section 2 we present RALOWS, our platform to make remote experiments available as Semantic Web Services. In Section 3 we formalize properties and classes which, using the Time ontology, allow the association of temporal restrictions to objects in general and to resources in particular. Our matching algorithm of Web Services discovery is detailed in Section 4. Related works are discussed in Section 5. Our final remarks and some directions for future works are presented in Section 6.
2 RALOWS

We have identified Remote Experiments as a category of applications that impose several and rigid constraints with respect to the availability of their resources, and proposed RALOWS (Remote Access Laboratory Ontology and Web Service) as a platform for investigating how Remote Experiments can be deployed as Semantic Web Services [13].

Our RALOWS platform takes advantage of the OWL-S ontology (Ontology Web Language for Services) [9] for describing a Remote Experiment as Web Services. The Time ontology [3] is used to define temporal restrictions associated with resources and services. The Resource ontology [6] is used to describe the resources made available by means of the Remote Experiment.

Figure 1 shows the RALOWS ontology using OWL-S as the upper ontology (classes Service, ServiceModel, Service-Grounding, ServiceProfile, Input) with the extensions from the Resource ontology (class Resource) and Time ontology (classes DateTimeInterval and ProperInterval), as well as RALOWS own classes (InputResource and Instrument).

![Figure 1. RALOWS ontology: Remote Access Laboratory Ontology and Web Service](image)

In RALOWS, a resource or collections of one or more resources (recursively) is anything that may be useful and made available in the context of an experiment: instruments, software, chemicals, live animals, etc. We also refer as a resource to a human being when she or he is a person who supports the execution of an experiment (e.g., a technician) or is an active agent with respect to the experiment (e.g., a tutor who interacts with remote students) [12].

The OWL-S ontology presents a property named provides (see Figure 1) which indicates that a Resource may provide a Service. In the domain of remote experiments we consider two types of resources, shown in Figure 1: InputResource and Instrument. These types of resources are extensions that we made in the Resource ontology to support remote experiments.

An Instrument is a type of resource that may be accessed and controlled via the Web to provide some service (e.g., an oscilloscope). Hence, it seems convenient that an Instrument be wrapped as Semantic Web Service in OWL-S.

The Inputs, Outputs, Preconditions and Effects of an instrument are specified in the ServiceProfile. Each Instrument has its proper IOPE’s which are intrinsic to the instrument.

Wrapping instruments as Semantic Web Services makes it possible the automatic discovery of experiments that are composed by these instruments. The ServiceProfile is used to advertise the experiments for discovering purposes. The ServiceProfile requested (by a user that searches for some experiment) is matched with several ServiceProfiles which are advertised by the providers.

In the ServiceProfile we also specify the experiment category, for example: a Chemistry experiments, a Physics experiment, a Biology experiment, a Psychology experiment, etc. The class ServiceProfile may have a property serviceCategory whose value can be defined by an ontology which describes service categories.

In the work reported in this paper, we defined some concepts to describe experiment categories (ChemistryExperiments, PhysicsExperiments, BiologyExperiments, PsychologyExperiments, etc.) for the purpose of testing the matching algorithm presented in Section 4.

The InputResource subclass of Resource is a type of resource that may appear as an input for a service. Some examples are: a chemical solution demanded in a Chemistry experiment; a technician needed to supervise the execution of an experiment; a rat used in a behavioral experiment, and so on.

As shown in Figure 1, an InputResource is a subclass of Input, which is the class for all inputs in OWL-S. We create this special type of input because some InputResources should be indicated in advance as a resource necessary to the execution of the experiment.

3 Temporal Concepts

The Time ontology defines some basic temporal concepts: Instant, Interval, Instant Event, and Interval Events. Hoobs and Pan describe instants as having no interior points and intervals as things with extent intuitively [3].

The class Interval of the Time ontology has a subclass named ProperInterval, and ProperInterval has a subclass named DateTimeInterval. A ProperInterval is one whose beginning and end are not identical. DateTimeInterval is a ProperInterval in which the beginning and end are a calendar date and time. For example, a ProperInterval may be an interval of one hour and a DateTimeInterval may be an interval of one hour that begins at 15:00 and ends at 16:00 on 9/8/2007.
3.1 Temporal Concepts for Resources

In order to allow the booking of experiments, we have defined two intervals that are associated with resources. The resource intervals are defined by two properties as shown in Figure 1: `availableTime` (defined in Document 1) and `readyTime` (defined in Document 2).

The property `availableTime` (lines 3 to 6 of Document 1) from the `Resource` class defines the time slots that the resource is available to be used in a remote experiment. A resource is declared as having zero or more cardinality with respect to the property `availableTime` (not shown in Document 1). As a result, a resource can have zero or more intervals of availability.

If the resource has zero intervals of availability, it is not available and cannot be booked; as result, all the experiments that need that resource will not be booked either. If the resource has one or more intervals of availability, the resource availability can be achieved by:

\[ \text{ResourceAvailability} \equiv \exists \text{availableTime.Interval1} \cup \exists \text{availableTime.Interval2} \cup \exists \text{availableTime.Interval3} \cup \ldots \cup \exists \text{availableTime.Intervaln} \]

Some resources need some preparation time so as to be ready for usage. For example the time associated with the preparation of a chemical solution demanded in a Chemistry experiment. The property `readyTime` (lines 3 to 6 of Document 2) from the `Resource` class has a zero or one cardinality (not shown in Document 2).

As an example, in order to define a session that may last from 30 to 120 minutes (`Session30-120minutesDuration`), we define its minimum session duration, i.e. 30 minutes, and maximum session duration, i.e. 120 minutes. Since the range of `minSessionDuration` and `maxSessionDuration` is an Interval, intervals with specific durations need to be created first. For `Session30-120minutesDuration`, we need to define `Interval30Minutes` and `Interval120Minutes` first as shown in Document 4.

3.2 Temporal Concepts for Experiments

Given that an interactive experiment may have one or more resources which have to be reserved in advance, the availability of an experiment can be computed by the intersection of the availability of all involved resources, as follows:

\[ \text{ExpAvailability} \equiv \exists \text{resource.1.ResourceAvailability} \cap \exists \text{resource.2.ResourceAvailability} \cap \ldots \cap \exists \text{resource.n.ResourceAvailability} \]

The interval of the reserved slot must be a valid Interval, according to the intervals of availability of the resource, and cannot overlap with an existing reserved interval.

Another temporal concept of remote experiments is the duration of a session, which is the time that a user spends executing an experiment. Some experiments may have a minimum and a maximum session duration.

In Document 3, both `minSessionDuration` (line 6) and `maxSessionDuration` (line 11) are defined as properties of `SessionDuration`. The cardinality of 1 for both properties in the definition of `SessionDuration` indicates that an instance of `SessionDuration` must have one and only one property value for `minSessionDuration` and `maxSessionDuration` respectively.
Finally, an experiment modeled as a Semantic Web Service in OWL-S (by the RALOWS approach in Section 2) has an Input named DateTimeSession (lines 3 to 5 of Document 6) of the type Session30-120minutesDuration. As a result, a session of this experiment must have a duration between 30 and 120 minutes.

In this paper we do not make any assumptions with respect to the service that reserves experiments. We propose the description of resource availability and resource booking by using the Time ontology. A Semantic Web Service for reservation may be implemented that, using the Time ontology, reserves the resources demanded by an experiment.

## 4 Matching Algorithm

The matching algorithms reported in the literature (e.g. [11, 14, 5, 7, 18, 4]) for OWL-S (or its predecessors) are based on the matching of services functionalities. These algorithms have some common characteristics as the use of ranked matching of the results and matching performed using the subsumption reasoning (Description Logic).

Subsumption reasoning is performed over the conceptual schema. The matching is not syntactic, but is based on the concept relations of an OWL ontology. Aspects of an OWL ontology, such as classes, properties, restrictions, intersections, unions and symmetric, transitive and inverse properties are used for subsumption reasoning.

We divide our matching algorithm in five stages: Category Matching, Input Matching, Output Matching, Availability Matching and Service Selection. The first four stages use subsumption reasoning to define the similarity degree of a matching. We use the four similarity degree of Paolucci et al. [11]. Each degree of matching has a ranked value. Moreover, the matching determines the semantic connectivity between a service request (Req) and a service advertisement (Adv).

In the following sections (4.1 to 4.5), we use, as a real scenario of experimentation, the Skinner box experiment described as Semantic Web Service in our previous work [13]. In Section 3, we defined some temporal concepts related with remote experiments to validate our matching algorithm. All examples of inputs and outputs of this section are based on the Skinner box experiment.

The variation of the Skinner box experiment discussed in this section aims at conditioning a rat, placed in a special box called Skinner box, to press a bar in order to obtain water. The rat is deprived of water for some time before being placed in the box, so that it is supposed to be thirsty. The Skinner box, in this case, has a bar and a water dispenser which releases a water drop upon activation. This activation can be made externally, by an experimenter pressing a water-release button, or internally, by the rat pressing the bar inside the box.
4.1 Category Matching

In the first stage of the algorithm, Category Matching, the matching determines the semantic connectivity between the service category of a service request (ReqCategory) and the service category of a service advertisement (AdvCategory).

Figure 2. Hierarchy of experiments categories.

The Figure 2 defines a hierarchy of experiments categories. Our matching algorithm uses ontologies for service classification to realize the subsumption reasoning for the matching. For example, a service request for the Skinner box experiment defines a service category Behaviorism (ReqCategory). The matching algorithm finds a service advertisement (AdvCategory) with an experiment that the service category is PsychologyExperiments. In this case, as shown in Figure 2, the degree of matching is not exact. The matching algorithm recognizes, by subsumption reasoning, that Behaviorism is a PlugIn (see Table 1) of PsychologyExperiments.

Table 1. Ranking for Category Matching

<table>
<thead>
<tr>
<th>Match</th>
<th>Explanation</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>If AdvCategory and ReqCategory are equivalent</td>
<td>0</td>
</tr>
<tr>
<td>PlugIn</td>
<td>If ReqCategory is sub-concept of AdvCategory</td>
<td>1</td>
</tr>
<tr>
<td>SubSume</td>
<td>If ReqCategory is super-concept of AdvCategory</td>
<td>2</td>
</tr>
<tr>
<td>Fail</td>
<td>Otherwise, we call the match fail</td>
<td>3</td>
</tr>
</tbody>
</table>

The degree of matching for the category matching is defined in Table 1. According to Table 1, if the matching by a given category returns a super-concept (PlugIn), we rank 1 – and for SubSume we rank 2. When the matching result is a PlugIn (e.g. PsychologyExperiments is found when Constructivism or Behaviorism is searched), in the remaining steps of the algorithm it will be possible to identify if the exact match is available.

4.2 Input Matching

The second stage of the algorithm, Input Matching, determines how much the input parameters of the advertised service (AdvInput) satisfies the requested service (ReqInput).

Table 2 shows the degree assignment for different matching: assign 0 for exact matching, 1 when the matching is a sub-set, 2 for super-concept and 3 when there is not match at all.

Table 2. Ranking for Input Matching

<table>
<thead>
<tr>
<th>Match</th>
<th>Explanation</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>If AdvInput and ReqInput are equivalent</td>
<td>0</td>
</tr>
<tr>
<td>PlugIn</td>
<td>If ReqInput is sub-concept of AdvInput</td>
<td>1</td>
</tr>
<tr>
<td>SubSume</td>
<td>If ReqInput is super-concept of AdvInput</td>
<td>2</td>
</tr>
<tr>
<td>Fail</td>
<td>Otherwise, we call the match fail</td>
<td>3</td>
</tr>
</tbody>
</table>

For example, in the context of the Skinner box experiment, we may have:

AdvInput ≡ ∃ hasInput.Light_In ∩ ∃ hasInput.Sound_In
ReqInput ≡ ∃ hasInput.Light_In ∩ ∃ hasInput.Sound_In ∩ ∃ hasInput.Water_In

Since AdvInput subsumes ReqInput, the matching result is a PlugIn with rank 1.

4.3 Output Matching

The third stage of the algorithm, Output Matching, determines how much the output parameters of the advertised service (AdvOutput) agrees with the requested service (ReqOutput). Table 3 is similar to Table 2. Zero is assigned for exact matching, 1 when the matching is a sub-set, 2 for super-concept and 3 when there is not match at all.

Table 3. Ranking for Output Matching

<table>
<thead>
<tr>
<th>Match</th>
<th>Explanation</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>If AdvOutput and ReqOutput are equivalent</td>
<td>0</td>
</tr>
<tr>
<td>SubSume</td>
<td>If AdvOutput is sub-concept of ReqOutput</td>
<td>1</td>
</tr>
<tr>
<td>PlugIn</td>
<td>If AdvOutput is super-concept of ReqOutput</td>
<td>2</td>
</tr>
<tr>
<td>Fail</td>
<td>Otherwise, we call the match fail</td>
<td>3</td>
</tr>
</tbody>
</table>

For example, in the context of the Skinner box experiment, we may have:

AdvOutput ≡ ∃ hasOutput.PressureBar_Out ∩ ∃ hasOutput.Results_Out
ReqOutput ≡ ∃ hasOutput.PressureBar_Out

In the above, since ReqOutput subsumes AdvOutput, the matching result is SubSume with ranking equals to 1.

4.4 Availability Matching

In Section 3 we define temporal concepts for remote experiments. In the fourth stage of our matching algorithm, we use these temporal concepts to perform the matching of experiment and resources availability. Concepts as
ResourceAvailability, ExpAvailability, SessionDuration, for example (defined in Section 3), are matched to discover the requested experiment.

The matching algorithm considers the resources and experiment availability (defined in Section 3). The objective of this stage is to discover whether the experiments matched in the previous stages have or not available time (ExpAvailability) for its execution if they may be reserved in the date and time requested (ReqAvailability).

### Table 4. Ranking for Availability Matching

<table>
<thead>
<tr>
<th>Match</th>
<th>Explanation</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>If ReqAvailability and ExpAvailability are equivalent</td>
<td>0</td>
</tr>
<tr>
<td>PlugIn</td>
<td>If ReqAvailability is sub-concept of ExpAvailability</td>
<td>0</td>
</tr>
<tr>
<td>SubSume</td>
<td>If ReqAvailability is super-concept of ExpAvailability</td>
<td>3</td>
</tr>
<tr>
<td>Fail</td>
<td>Otherwise, we call the match fail</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4 shows that, if the requested time is equal to the available time (ExpAvailability), or if the requested time is a sub-concept of the available time, the matching is ranked 0. This is because if the requested time is equal (Exact), or is contained (PlugIn) in the available time, the remote experiment is available for execution or reservation. If more time (SubSume) than the available time is requested, the experiment will not have enough availability for its execution – in at the required date and time.

### 4.5 Service Selection

The fifth and last stage of our matching algorithm, Service Selection, generates an overall ranking of matching (see Table 5). In this stage, the matching algorithm only has Fail as final result if at least one of the four previous stages also has Fail as result.

### Table 5. Final Ranking for Service Selection

<table>
<thead>
<tr>
<th>Match/Explanation</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match</td>
<td>0</td>
</tr>
<tr>
<td>Result of each stage is Exact</td>
<td>0</td>
</tr>
<tr>
<td>Match</td>
<td>1</td>
</tr>
<tr>
<td>At least one result in a stage is ranked 1, and other results are ranked not more than 1</td>
<td>1</td>
</tr>
<tr>
<td>Match</td>
<td>2</td>
</tr>
<tr>
<td>At least one result in a stage is ranked 2, and other results are ranked not more than 2</td>
<td>2</td>
</tr>
<tr>
<td>Fail</td>
<td>3</td>
</tr>
<tr>
<td>In one stage the result is Fail</td>
<td>3</td>
</tr>
</tbody>
</table>

By breaking the matching algorithm into stages, we are able to discover in which of the stages the result Fails, if that is the case – it means that it is possible to identify why it is not possible to reserve a remote experiment.

### 5 Related Work

Paolucci et al. [11] present a semantic matching algorithm based on service capabilities: their algorithm considers inputs and outputs and a Degree of Matching (DoM) calculated as the similarity between outputs and inputs. Their algorithm considers four degrees of matching: Exact, PlugIn, SubSume and Fail. Li and Horrocks [7] add a DoM, called Intersection, to the algorithm of Paolucci et al.

Sycara et al. [15] propose a matching engine that implements an algorithm for service capability matching and adds capability matching to UDDI. Srinivasan et al. [14] present a OWL-S Profile to UDDI mapping mechanism and extend the matching algorithm of Sycara et al. [15] to match service product and service classification properties of OWL-S.

Klusch et al. [5] use semantics and similarity measures to improve the matching process. Oh et al. [10] use the matching of parameters in SOAP message so as to determine relations between different parameter types during the process of service composition.

Wang and Strouliia [17] use traditional information-retrieval methods to identify the most similar service descriptions that they named services candidates. In a second step, they refine these services candidates by a structure-matching method.

Hao and Zhang [2] propose a new schema matching algorithm for supporting web-service operations matching. The algorithm is based not only on the schema structure but also on the semantic information of each schema.

The approaches aforementioned are based on the functionalities of the service. In our case, we also consider temporal restrictions. In [8], temporal reasoning capability for Web Services has been proposed. In that work, the authors have focus on the synchronization between services – in the phase of service composition. Our work has focus in the temporal restrictions for service availability – in the phase of service discovery.

Our matching algorithm is similar to those of Yao et al. [18] and Jaeger et al. [4], who divide the matching algorithm into stages. The first step of our algorithm performs matching based on service categories. The second and third steps perform matching based on capabilities (similar to Li and Horrocks [7]). The fourth step considers the availability of the resources for matching purposes, which is an essential requirement in remote experiments and an important contribution reported in this paper. In the last step we use the ranking technique used by Yao et al. [18].

### 6 Final Remarks

We have proposed an approach to allow the discovery of Web Services based on service parameters, service categories and temporal restrictions.

We use the Time ontology to add temporal concepts to the service and resource descriptions so as to make it possible to provide reservation and to impose restrictions in the use of the resources involved in the service.

Leverages the work reported in the literature, we have presented a matching algorithm so as to consider category
matching and availability matching.

The context of Web Services related to remote experiments – a category of applications that may impose several and rigid constraints with respect to the availability of their resources – was taken as a case study of the proposed matching algorithm. Further potential applicability can be envisioned as in booking for meetings – people with the adequate skills can be automatically discovered and a meeting can be booked.

We plan to extend our approach to consider QoS restrictions of Web Services in the matching algorithm.

In this paper we do not make any assumptions with respect to the service that manage services reservation. We are working on a Semantic Web Service that uses our matching algorithm for such management. Our efforts focus on a general approach for services reservation that can be applied to other domains. In particular, the Brazilian Interactive Digital TV Platform presents interesting problems with respect to scheduling of time slots in programs with different requirements regarding the inputs, outputs, preconditions and effects of user interactions with the interactive programs, which are specified in the XML-based declarative language Ginga-NCL\(^1\).

Acknowledgments. We acknowledge financial support from FAPESP to TIDIA-Ae project, and from FINEP to the Brazilian Digital TV Ginga-NCL project. We also thank: CAPES for financial support to Maria da Graça Pimentel, and CNPq for the financial support to Maria da Graça Pimentel.

References