Interleaving Discovery and Composition for Simple Workflows*

Ora Lassila
Nokia Research Center, Burlington, MA

Sapna Dixit†
Cornell University, Ithaca, NY

Abstract

This paper describes a simple approach for service discovery and composition, where services are described using a subset of the DAML-S ontology. The central idea is to base composition on the *mismatch* between service discovery queries and their results, and iteratively letting this guide subsequent queries. This approach is a result of work aimed at understanding how much the DAML-S model can be simplified and still be viable for semantic annotation of Web services, specifically when the "use case" is that of automatic substitution of services that have become unavailable. A proof-of-concept prototype to demonstrate the viability of the scheme is described.

Introduction

The ability to automatically discover, compose and invoke Web services is an important component (and *benefit*) of the so-called *Semantic Web* (Berners-Lee, Hendler, & Lassila 2001), and a key enabler to the anticipated "serendipity" of agent behavior on the Semantic Web (Lassila 2002a). Upper ontologies for semantic annotation of Web services have started to appear, making it possible to apply Web services in the Semantic Web context. One of these is DAML-S (Ankolekar *et al.* 2002) which in itself is quite complex, and motivates the question whether a simpler ontology could be sufficient for some uses of *semantic Web services*.

Our attempts to simplify the DAML-S model have predominantly been based on a single assumption, namely that the concept of *complex process* could be eliminated. In practice, this means that all described services are in effect "black boxes", i.e., they have input and output parameters, but nothing more is known of their internal workings, and their potential components cannot be invoked separately. For the purposes of this paper, the services are further restricted to only have a single input parameter and a single output parameter (extending to multiple inputs will make the search algorithm a bit more complicated, but will generally not change the principle presented in this paper).

*Work described in this paper was supported in part by Nokia Mobile Phones and the Nokia Research Center.

†This paper describes work conducted during the author's internship at the Nokia Research Center.

In this paper we will propose an approach to Web service composition for simple workflows consisting of the aforementioned services with a single input and a single result – typically these services would be what (McIlraith, Son, & Zeng 2001) refers to as "information-providing services" (conversions, queries, etc.) but we are not excluding the possibility of some type of (real world) side effects. The principal aim is to provide for *service substitution* in the face of changing availability of services used.

It is assumed that the following exist and are available:

- A service discovery mechanism that accepts *partial* DAML-S service descriptions as queries, and returns *complete* descriptions of services discovered. No assumptions are made about how this mechanism actually finds services nor where the descriptions of these services would be located; details of this mechanism are beyond the scope of this paper.
- One or more ontologies for *classifying* services; that is, classification taxonomies describing *what* services actually do.
- Ontologies defining concepts and datatypes in the domains relevant to the services used (for example, if we are interested in services that deal with geographical phenomena – such as various map services – we would assume the existence of an ontology that defines concepts such as *address*, *GPS coordinates*, *region*, *containment of regions*, etc.).

It is further assumed that the ontologies and knowledge representation used could rely on (some form of) logic for simple expression of concept subsumption, specifically because we assume that the "quality" of the discovery results (i.e., the degree of the match) can be classified according to the discrete scale presented in (Paolucci *et al.* 2002): *exact*, *plugin*, *subsumes*, and *fail*.[1] We anticipate that more complex logical relations are not be needed. In this paper, only RDF (Lassila & Swick February 1999) is used for describing services, albeit it has its obvious shortcomings.

[1]The additional match category of *intersects* as described in (Li & Horrocks 2003) is not considered.

"DAML-S Lite": a Subset of DAML-S

For service description we are using a highly simplified subset of DAML-S written in RDF. The rough idea is that the concept of a complex process is eliminated. Each service described is assumed to be a black box, with inputs and outputs which are described in terms of their datatypes (expressed using RDF classes). Effectively this simplification eliminates process models from the ontology. Each service is also classified using some hypothetical service taxonomy; in practice this means that each service profile is an instance of an appropriate service class (here we differ from DAML-S, since service classification now happens by subclassing the `ServiceProfile` class, as opposed to using a separate value for the `serviceCategory` property of service profiles). Furthermore, for the purposes of the prototype described in this paper, we assume that each service only has one input and one output. Compositions of services like this form linear sequences of operations.

A minimal version of the upper "DAML-S Lite" ontology, one that effectively only addresses service profiles, is included as Appendix B.

Service Substitution

The particular "use case" of interest – at least to the authors of this paper – is that of *service substitution*, that is, a situation where a service needs to be replaced with an "equivalent" service (given *some* notion of equivalence). For example, imagine that we have a system that makes use of a service x which then becomes unavailable for one reason or another; we would now like the system to *automatically* be able to discover a set of substituting services $\{y_i\}$ which, when assembled into a linear workflow would provide an equivalent service comparable to x (by "linear workflow" we mean a situation where the output of service y_i is "fed" to the input of service y_{i+1}, for $1 \leq i \leq n-1$). A typical situation is one where many of the services in the workflow perform conversions or other mediation of input and output parameters.

It is the authors' belief that with the emerging mainstream deployment of architectures for *service-oriented computing* (Papazoglou & Georgakopoulos 2003), the *availability* of services will be of critical importance. In scenarios where near-100% availability cannot be guaranteed, the idea of automatic substitution of services (that have become unavailable) becomes very compelling.

There can naturally be several reasons for a service to become unavailable. In addition to a service going "off-line" for one reason or another (server crash, partial network outage, etc.), the preconditions for invoking a particular service may become invalid. For example, a service may be tied to a specific geographical location, and when the invoking terminal moves – it could be a mobile phone – a new service has to be discovered. More generally, in a software framework that supports *context awareness*, services could be tied to a particular *usage context*, and substitutions have to be made whenever there are relevant changes in context. Furthermore, service substitution could be used to opportunistically take advantage of the best available services (given some criteria to determine what "best" means – criteria such as cost, quality, speed, etc. could be used).

Composing Workflows

Given our basic use case of service substitution, a single service may be replaced by a slightly different service that potentially needs input and output parameter conversions. These conversions may be performed using either internal functions or by external services; in either case they constitute additional "workflow" to be performed before and after the invocation of the "main" service. From the viewpoint of DAML-S, whether the functionality is internal or external is only a matter of expressing the proper *grounding* of the particular service.

We compose workflows using a breadth-first search, starting with the description of the service we want to replace. Based on the degree of match[2] in the results of service discovery, the search proceeds as follows:

- For *exact* and *plugin*, the result of the query is accepted.
- For *subsumes*, the service found is accepted, but the mismatch between the original query and its result will be used to construct further queries to convert input and output parameters, in order to compose a service that exactly matches the original. For example, given a discovered service whose output parameter has a type "temperature" and whose unit is "Celsius" when we were looking for results to be returned in "Fahrenheit", this implies that we will have to find a conversion from "Celsius" to "Fahrenheit".
- For *fail*, re-perform the query with relaxed input and output parameter descriptions, implying that a resulting successful match will have to be further augmented with input and/or output parameter conversions (a relaxed parameter description is one that is more general in the classification sense).

This analysis is performed for the results of each query – also when these queries are conducted as a result of previous analyses – resulting in a recursive descent tree traversal. The intermediate results of the algorithm consist of workflows expressed in terms of services and queries (the reader is here reminded that queries and concrete services – that is, results of queries – are expressed the same way). As long as an intermediate result contains "unexpanded" queries, the algorithm will keep "rewriting" the result. Any number of parallel intermediate results (= "hypotheses") may be pursued; a null result from a query causes a hypothesis to be eliminated from the search.

In case of large fan-out of the search – that is, when a large number of services match a query – some type of heuristic pruning could be applied on the intermediate hypotheses. The heuristic applied could be based on similar criteria as when the algorithm is used for opportunistically finding the best available services.

The algorithm is described formally in Appendix A.

[2] As in (Paolucci *et al.* 2002).

Practical Implementation Using RDF

We have constructed a proof-of-concept prototype of the composition algorithm that uses RDF as its description language. Because of the use of RDF, the actual *matching* of service descriptions is partially based on procedurally expressed semantics: we look at service classification, input and output separately[3] – two service descriptions match if all three components, correspondingly, are "compatible", where compatibility is defined in terms of the degree of match. As matching ultimately comes down to the comparison of RDF datatypes, we use the following logic:

$$match(a,b) = \begin{cases} \textit{exact} & \text{if } a = b \\ \textit{subsumes} & \text{if } b \subset a \\ \textit{plugin} & \text{if } a \subset b \\ \textit{fail} & \text{otherwise} \end{cases}$$

Relaxation, as described in the appendix, is in our implementation performed by "walking" up the RDF class tree; it can also be done in a single step by replacing any type with the root of the class tree `rdfs:Resource`.

Given that sequences of conversions can result in loops, the prototype also performs simple pattern matching of hypotheses and eliminates those where loops are seen forming.

The prototype implementation is based on the Wilbur toolkit for Semantic Web programming (Lassila 2001; 2002b) and written in Common Lisp.

Conclusions

The ability to automatically replace services in the event of service failure (due to server or connectivity outage, or change in service invocation preconditions) is an important aspect of building robust and autonomous agents for the Semantic Web. The approach we have described achieves this for limited services and linear workflows. It is our belief that useful compositions can be created without sophisticated planning technologies given the limitations described earlier (including the use of a subset of the DAML-S ontology).

Future work will address some of the shortcomings of our current implementation, including the use of RDF as the representation language, and the lack of heuristic pruning during search. We will also explore whether this algorithm can be extended for more complex situations, and will conduct some more realistic testing of the idea. Naturally, the key to the utility of this mechanism is in the availability of semantically annotated Web services.[4]

References

Ankolekar, A.; Burstein, M.; Hobbs, J. R.; Lassila, O.; McDermott, D.; Martin, D.; McIllraith, S. A.; Narayanan, S.; Paolucci, M.; Payne, T.; and Sycara, K. 2002. DAML-S: Web service description for the Semantic Web. In Horrocks, I., and Hendler, J., eds., *The Semantic Web - ISWC 2002*, Lecture Notes in Computer Science 2342. Springer Verlag. 348–363.

Berners-Lee, T.; Hendler, J.; and Lassila, O. 2001. The Semantic Web. *Scientific American* 284(5):34–43.

Erol, K.; Hendler, J.; and Nau, D. 1994a. Semantics for hierarchical task network planning. Technical report, Department of Computer Science, University of Maryland - College Park.

Erol, K.; Hendler, J.; and Nau, D. S. 1994b. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, 1123–1128. Seattle, Washington, USA: AAAI Press/MIT Press.

Lassila, O., and Swick, R. R. February 1999. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, World Wide Web Consortium.

Lassila, O. 2001. Enabling Semantic Web Programming by Integrating RDF and Common Lisp. In *Proceedings of the First Semantic Web Working Symposium*. Stanford University.

Lassila, O. 2002a. Serendipitous interoperability. In Eero Hyvönen., ed., *The Semantic Web Kick-off in Finland – Vision, Technologies, Research, and Applications*, HIIT Publications 2002-001. University of Helsinki.

Lassila, O. 2002b. Taking the RDF Model Theory Out for a Spin. In Horrocks, I., and Hendler, J., eds., *The Semantic Web - ISWC 2002*, Lecture Notes in Computer Science 2342. Springer Verlag. 307–317.

Li, L., and Horrocks, I. 2003. A software framework for matchmaking based on semantic web technology. In *Proceedings of the 12th International Conference on the World Wide Web*.

McIlraith, S.; Son, T.; and Zeng, H. 2001. Mobilizing the web with DAML-enabled web services. In *The Second International Workshop on the Semantic Web (SemWeb'2001) at WWW-10*.

Paolucci, M.; Kawamura, T.; Payne, T.; and Sycara, K. 2002. Semantic matching of web services capabilities. In Horrocks, I., and Hendler, J., eds., *The Semantic Web - ISWC 2002*, Lecture Notes in Computer Science 2342. Springer Verlag.

Papazoglou, M., and Georgakopoulos, D. 2003. Service-oriented computing. *Communications of the ACM* 46(10):25–28.

Acknowledgements

The authors wish to thank Franklin Davis and Heikki Saikkonen for support during this project. The work was funded in part by Nokia Mobile Phones and the Nokia Research Center.

[3] The use of a more expressive language (say, OWL DL) would allow us to express service descriptions (including parameters) as single class expressions, in turn allowing us to rely more on the reasoning engine for matching.

[4] The real-world availability of such services is beyond the scope of this article.

Appendix A: Composition Algorithm

The following is a formal description of the composition algorithm. Effectively, search happens in a breadth-first manner, and resembles some approaches to HTN-planning (Erol, Hendler, & Nau 1994b; 1994a).

$$RewriteAll(\{[q_{1_1},\ldots,q_{m_1}],\ldots,[q_{1_k},\ldots,q_{m_k}]\}) \tag{1}$$
$$= \begin{cases} \{[q_{1_1},\ldots,q_{m_1}],\ldots,[q_{1_k},\ldots,q_{m_k}]\} & \text{if } \forall i=1\ldots m, \forall j=1\ldots k, complete(q_{i_j}) \\ RewriteAll(Rewrite(\{[q_{1_1},\ldots,q_{m_1}],\ldots,[q_{1_k},\ldots,q_{m_k}]\})) & \text{otherwise} \end{cases}$$

$$Rewrite(\{[q_{1_1},\ldots,q_{m_1}],\ldots,[q_{1_k},\ldots,q_{m_k}]\}) \tag{2}$$
$$= Concatenate(RewriteOne([q_{1_1},\ldots,q_{m_1}]), Rewrite(\{[q_{1_2},\ldots,q_{m_2}],\ldots,[q_{1_k},\ldots,q_{m_k}]\}))$$

$$Rewrite(\{\}) = \{\} \tag{3}$$

$$RewriteOne([q_1,\ldots,q_m]) \tag{4}$$
$$= \begin{cases} SubstituteAlternatives(q_j, expand(q_j), [q_1,\ldots,q_m]) & \text{where } \exists j, \neg complete(q_j) \\ \{[q_1,\ldots,q_m]\} & \text{otherwise} \end{cases}$$

$$Concatenate([x_1,\ldots,x_i],[x_j,\ldots,x_k]) = [x_1,\ldots,x_i,x_j,\ldots,x_k] \tag{5}$$

$$SubstituteAlternatives(q_j,\{[s_{1_1},\ldots,s_{m_1}],\ldots,[s_{1_k},\ldots,s_{m_k}]\},[q_1,\ldots,q_n]) \tag{6}$$
$$= \{Substitute(q_j,[s_{1_1},\ldots,s_{m_1}],[q_1,\ldots,q_n]),\ldots,Substitute(q_j,\ldots,[s_{1_k},\ldots,s_{m_k}],[q_1,\ldots,q_n])\}$$

$$SubstituteAlternatives(q_j,\{\},[q_1,\ldots,q_n]) = \{\} \tag{7}$$

$$Substitute(q_j,[s_i,\ldots,s_m],[q_1,\ldots,q_n]) = [q_1,\ldots,q_{j-1},s_1,\ldots,s_m,q_{j+1},\ldots,q_n] \tag{8}$$

Note that in the above description there are two functions that have not been described in formal terms:

The function $complete(q)$ is a boolean test indicating whether a service description q is "complete", that is, whether it is an actual concrete service as opposed to a query expression (think of a complete description as a leaf that cannot be elaborated further in an HTN plan tree).

The function $expand(q)$ will perform the actual querying, and will return a set of results (the resulting set may be empty). It will perform the relaxation of queries based on a failure to discover new services, and will construct compositions based on the mismatch between the original query and the results of the relaxed query. For example, given a query q that can only be satisfied after relaxation, we have

$$expand(q) = \{[q_{input_1}, s_{q_1}, q_{output_1}],\ldots,[q_{input_n}, s_{q_n}, q_{output_n}]\} \tag{9}$$

where s_{q_i} is the concrete results of querying for a relaxed form of q, and where q_{input_i} is a query for a service that will convert the input of q to the input of s_{q_i} (correspondingly q_{output_i} will convert the output of s_{q_i} to the output of q). These conversions are called "mismatches": based on the degree of match the conversion may not be needed.

Appendix B: Minimal "DAML-S Lite" Upper Ontology

The following is a minimal version of the DAML-S "subset". Note how the new classes and properties are subclasses or subproperties of the corresponding OWL-S concepts.

```
<?xml version="1.0"?>

<!DOCTYPE uridef [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY dsl "http://www.nokia.com/research/semanticweb/schemata/daml-s-lite#">
  <!ENTITY owl-s-profile "http://www.daml.org/services/owl-s/0.9/Profile.owl#">
  <!ENTITY owl-s-service "http://www.daml.org/services/owl-s/0.9/Service.owl#">
]>

<rdf:RDF xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;"
         xmlns:dsl="&dsl;"
         xmlns:owl-s-profile="&owl-s-profile;"
         xmlns:owl-s-service="&owl-s-service;">

<rdfs:Class rdf:about="&dsl;ServiceProfile">
  <rdfs:subClassOf rdf:resource="&owl-s-service;ServiceProfile"/>
</rdfs:Class>

<rdf:Property rdf:about="&dsl;input">
  <rdfs:subPropertyOf rdf:resource="&owl-s-profile;input"/>
  <rdfs:domain rdf:resource="&dsl;ServiceProfile"/>
  <rdfs:range rdf:resource="&dsl;ParameterDescription"/>
</rdf:Property>

<rdf:Property rdf:about="&dsl;output">
  <rdfs:subPropertyOf rdf:resource="&owl-s-profile;output"/>
  <rdfs:domain rdf:resource="&dsl;ServiceProfile"/>
  <rdfs:range rdf:resource="&dsl;ParameterDescription"/>
</rdf:Property>

<rdfs:Class rdf:about="&dsl;ParameterDescription">
  <rdfs:subClassOf rdf:resource="&owl-s-profile;ParameterDescription"/>
</rdfs:Class>

<rdf:Property rdf:about="&dsl;parameterName">
  <rdfs:subPropertyOf rdf:resource="&owl-s-profile;restrictedTo"/>
  <rdfs:domain rdf:resource="&dsl;ParameterDescription"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&dsl;restrictedTo">
  <rdfs:subPropertyOf rdf:resource="&owl-s-profile;restrictedTo"/>
  <rdfs:domain rdf:resource="&dsl;ParameterDescription"/>
  <rdfs:range rdf:resource="&rdfs;Class"/>
</rdf:Property>

</rdf:RDF>
```