Matchmaking of Web Services Based on the DAML-S Service Model

Sharad Bansal
University of South Carolina
Computer Science and Engineering
Columbia, SC, 29208
bansal@cse.sc.edu

José M. Vidal
University of South Carolina
Computer Science and Engineering
Columbia, SC, 29208
vidal@sc.edu

ABSTRACT

DAML-S provides the means for a web service to advertise its functionality to potential users of the service. This brings to fore the issue of discovering an advertisement that best matches a request for a particular service—a process referred to as matchmaking. The algorithms that have thus far been proposed for matchmaking are based on comparisons of the requested and offered inputs and outputs. In this project, we extend these algorithms by taking into account the detailed process description of the service, thus leading to more accurate matchmaking.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Artificial Intelligence—*multiagent systems*

General Terms

Algorithms

Keywords

DAML-S, UDDI, Service Discovery

1. INTRODUCTION

The semantic web vision calls for a transformation of the world wide web from a provider of information to a purveyor of services. Matchmaking agents would, upon receiving a request from a consumer of a web service, search their database of advertisements to come up with a set of advertisements that best meet the requested requirements. This process is referred to as matchmaking.

The DARPA Agent Markup Language for Services (DAML-S) has been developed to serve as the medium of expression for web service capabilities. A DAML-S advertisement consists of three parts: the Service Profile, the Service Model, and the Service Grounding. The matchmaking algorithms proposed thus far have been based on the Service Profile, and operate by comparing the requested inputs and outputs against the advertised inputs and outputs. It is our contention, however, that exclusive use of the Service Profile does not allow users to fully exploit the information available in a DAML-S advertisement.

AAMAS'03, July 14–18, 2003, Melbourne, Australia.
ACM 1-58113-683-8/03/0007.

The limitations of Service Profile based matchmaking arise due to the logical relationships underlying the inputs and outputs of a process. To put this in perspective, one can envision a simple choice process which produces two outputs, say o_1 and o_2. If a request for a service providing both these outputs were to be matched by simply comparing the Service Profile outputs, the result would be a positive match. In reality however, the process is not capable of providing both these outputs. This simple example illustrates the need for taking into account the logical nature of processes comprising a service in order to arrive at an accurate match. We have developed and implemented algorithms for such matchmaking based on the process model and present these algorithms in this paper.

2. SERVICE MODEL MATCHMAKING

The Process Model, the main subclass of the Service Model, decomposes the service into its constituent processes. A process can, in turn, consist of other processes, in which case it is said to be a Composite Process. Various types of composite processes are: Split, Sequence, Unordered, Split+Join, Choice, If-then-else, Iterate and Repeat-Until. Processes that are not decomposable any further are known as Atomic Processes. The algorithms that follow employ a tree data structure to represent the DAML-S Process Model advertisement.

The algorithms developed are recursive in nature. Each type of composite node as well as atomic node has a corresponding matchmaking algorithm. The matchmaking process commences by initiating the matchmaking algorithm for the root node of the advertisement, which in turn invokes the matchmaking process for its child nodes, and so on until the process bottoms out at the leaf nodes of the advertisement tree.

The algorithm for matching outputs for the Split-Sequence node, which corresponds to a Split or a Sequence composite process is described below. We will be using the following data structure to represent a DAML-S node:

```
struct Node {
  Node[] children;
  set matchSet;
  list outputs;
  list inputs; }
```

where children is an array of child nodes, matchSet is a set of outputs that are currently matched against this node,

outputs is the list of outputs provided by the node and inputs is the list of inputs of the node. The outputs and inputs are specified only for Atomic Nodes, that is, nodes with no children. The matchSet of all the nodes in the tree are initially empty.

The query consists of the inputs provided by the user and the outputs expected by her. Formally, we define a query as a pair of lists: $Q = (I, O)$, where $I = i_1, i_2, \ldots, i_m$ is the list of inputs to be matched against the node and $O = o_1, o_2, \ldots, o_n$ is the list of outputs to be matched against the outputs of the node.

2.1 Split Node/Sequence Node

The components of a Split process are a bag of process components to be executed concurrently. A Sequence process consists of a list of processes which are to be done in order. The algorithm to match outputs of either a split or a sequence node is as follows:

```
matchOutputs(List I, List O, split-seq-Node N)
if O is empty then
  return true
end if
o1 ← head(O)
for all k ∈ N.children do
  k.matchSet ← k.matchSet ∪{o1}
  if matchOutputs(I, k.matchSet, k) then
    if matchOutputs(I, tail (O), N) then
      return true
    end if
  end if
  k.matchSet ← k.matchSet - o1
end for
return false
```

The above algorithm works by distributing the outputs required from a sequence/split node over its children. If the desired outputs can be satisfied by all the children collectively, the match is a success, otherwise a failure. The algorithm employs backtracking to find such a distribution of outputs over the children. It thus tries all possible distributions of outputs before returning a failed match.

Since this algorithm examines the possible placements of the n outputs to be matched among the c process nodes in the advertisement, in the worst case all the c^n possible placements will be tried. Thus the worst case asymptotic time complexity of the algorithm is $O(c^n)$. This time assumes that all the calls to matchOutputs are recursive calls to this algorithm and not calls to the other algorithms presented in the next sections, except for the last call, which is assumed to be to an atomic node. That is, the time complexity is based on the assumption that the whole advertisement consists only of split or sequence nodes. The basic algorithm presented above can be modified to take into account negated outputs. This would enable the request to specify that the matched advertisement should not produce a particular output.

2.2 Choice Node

The algorithm for a choice node matching is:

```
matchOutputs(List I, List O, choice-Node N)
for all k ∈ N.children do
  if matchOutputs(I,O,k) then
    return true
  end if
end for
return false
```

To determine the worst case time complexity of this algorithm, we consider a tree of Choice nodes of height h with each node having c children. Let $T(h)$ be the time complexity of the algorithm for such a tree. Then, $T(h) = c \cdot T(h-1)$. This equation has a solution for $T(h) = c^h$. Hence, the worst case time is $O(c^h)$, assuming that all the calls to matchOutputs are to this algorithm.

This basic algorithm has also been extended to identify the type of match based on the hierarchy of outputs, as well as to account for negative terms in the user's query. The matchmaking algorithms for other process nodes have also been formulated, but are not described here for sake of brevity. The interested reader should contact the authors for the complete set of algorithms.

3. IMPLEMENTATION AND TESTING

We have implemented the above algorithms in Java and performed a series of tests to determine the feasibility of our approach. In our implementation the DAML-S advertisement was read using the DAMLJessKB package, which converts a DAML file into a set of equivalent Subject-Verb-Object (SVO) triples. These triples are then asserted into the JESS knowledge base and the rules of the DAML language are then applied by JESS. Our matchmaking agent then queries JESS to obtain the information necessary for building the advertisement tree. Finally, the request query is parsed and the matchmaking process is executed on the basis of the above algorithms. The use of the JESS and DAMLJessKB reduced our development time and facilitated the matching based on the subsumption hierarchy.

4. CONCLUSION

The Service Model provides a far richer description of a service than the Service Profile. The algorithms presented in this paper exploit the additional information available in the Service Model to provide matchmaking capabilities in situations where the conventional IOPE-based matchmaking algorithms are unable to determine suitable matches for a request. Our use of the Service Model also affords more flexibility since changes in the underlying Service Model do not necessarily have to be reflected in the Service Profile to support matchmaking. While these algorithms have a worst case timing analysis of exponential order, the average case performance is actually much better since most of the possible distributions of outputs over the advertisement process nodes are not explicitly examined.

While the algorithms presented above are applicable to the simplest case, we have extended them to incorporate queries with negative terms and also matchmaking based on the subsumption hierarchy of outputs, thus making it possible to distinguish the quality of match as well. Hence our algorithms use the subsumption hierarchy to distinguish among an exact, plug-in, subsumes or failed match.

We believe that matchmaking based on the Service Model is a prerequisite for the automatic composition of web services as envisioned by the Semantic Web vision. Our algorithms form a first step towards the automatic construction of sophisticated aggregate web services.