MATCHMAKING OF WEB SERVICES BASED ON THE DAML-S SERVICE MODEL

by

Sharad Bansal

Bachelor of Engineering

Bangalore University, 1994

Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in the Department of Computer Science and Engineering College of Engineering and Information Technology University of South Carolina 2002

Department of Computer Science and Engineering Director of Thesis

Department of Computer Science and Engineering 2nd Reader

Department of Computer Science and Engineering 3rd Reader

Dean of the Graduate School
ABSTRACT

DAML-S provides the means for a Web service to advertise its functionality to potential users of the service. This brings to the 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. That is, we present an algorithm that will allow users to find services based on their Service Model Description. The query language we introduce supports both positive and negative terms. The algorithm runs in worst time of $O(c^n)$, where $c$ is the number of process nodes in the advertisement and $n$ is the number of outputs to be matched. We also show results of tests performed against a simple database.
# Table of Contents

1. Introduction ........................................... 1

2. Overview of DAML-S .................................. 4

3. Related Work .......................................... 9

4. Shortcomings of Matchmaking Based on Service Profile .... 12

5. Matchmaking Based on the Service Model .............. 14

6. Implementation and Testing ......................... 27

7. Conclusion .......................................... 32

8. References ........................................... 34

9. Appendix 1 - Test Cases .................................. 35

10. Appendix 2 - Code Listing ........................... 42

11. Appendix 3 - DAML-S Advertisement ............... 66
List of Figures

1. Schematic of a Choice Process 12

2. Node Inheritance Hierarchy 16


4. Hierarchical Decomposition of Advertisement used in Tests 31
Chapter 1

Introduction

The Semantic Web vision [1] calls for a transformation of the World Wide Web from a provider of information to a purveyor of services. The agents providing these services would register a description of their capabilities, called an advertisement, with mediator agents known as matchmaking agents. These middle agents, upon receiving a request from a consumer of a Web service, would 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 [1].

The DARPA Agent Markup Language for Services (DAML-S) has been developed to serve as the medium of expression for Web service capabilities [2]. A DAML-S advertisement consists of three parts: the Service Profile, the Service Model, and the Service Grounding. The Service Profile specifies the capabilities of the Web service by stating the inputs that are expected by the Web service as well as the outputs produced by it. The Service Model, on the other hand, gives a description of how the service works by decomposing the service into its constituent sub-processes.

The matchmaking algorithms proposed thus far, which we review in Chapter 3, 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. Specifically, it does not allow the user to make queries based on the manner in which the inputs are transformed into outputs, which is specified in the Process Model. We have developed and implemented algorithms for such matchmaking and present these algorithms in this paper.

For example, imagine a DAML-S advertisement that represents an online computer retailer. The detailed description of the DAML-S advertisement for such a hypothetical Web service is given in Chapter 6. Briefly, such a service is considered to be a sequence of processes corresponding to the customer logging in, configuring a computer system to suit her requirements, selecting the transaction type, specifying delivery details, and finalizing the transaction. As part of this sequence the customer has the option to specify whether she wants to enter into a hire-purchase transaction, a lease transaction, or a direct sale. The inputs expected from the user and the outputs returned will be different in each case. Under the traditional matchmaking based on the Service Profile, the match would be successful only if the user provided the inputs for all three possible transaction options. However, by using the Process Model, we can differentiate among the possible transaction options and obtain a successful match by accepting an input corresponding to only one of these options.

This paper is organized as follows: in Chapter 2 we briefly review the basic structure of DAML-S. Chapter 3 goes on to discuss the current algorithms proposed for matchmaking
based on the Service Profile. The limitations of these algorithms are discussed in Chapter 4. In Chapter 5, we present our algorithms based on the Service Model. The implementation and testing of these algorithms is outlined in Chapter 6. Finally, we conclude this paper in Chapter 7.
Chapter 2

Overview of DAML-S

The Darpa Agent Markup Language for Web Services (DAML-S) is an ontology description language for Web services, i.e it provides a mechanism to describe the capabilities and properties of a Web service in a machine interpretable form. To put DAML-S in perspective, it is necessary to understand its foundations viz, the Extensible Markup Language (XML) and the Resource Description Framework (RDF).

Raw data by itself is meaningless unless put in a particular context. For example, Arthur C. Doyle may be any name. However, putting it in the context of “author” allows us to interpret that Arthur C. Doyle writes books. XML provides a means of specifying this context by marking up the data with tags. Thus the above information can be encoded as:

<author>
    Arthur C. Doyle
</author>

While XML provides us with a syntax for describing metadata, RDF provides us with a model or framework for specifying this metadata. RDF by itself is independent of the
means used to actually encode the information, but this role is generally fulfilled by XML. All objects being described by RDF are viewed as resources, which are identified by URLs. Thus, a resource may be a Web page, a person or any other object that we wish to describe. A resource is associated with one or more attributes, which are basically its characteristics. RDF specifies that information be represented in the form of ‘triples’ of the form `<Resource, Attribute, Attribute-value>`. The attribute value can be another resource or a literal value. Hence, if we want to state that Arthur C. Doyle is the author of ‘The Hound of Baskervilles’, we would construct the following triple: `<‘The Hound of Baskervilles’, Author, ‘Arthur C. Doyle’`

RDF Schema (RDF-S) provides a type system for RDF by defining a supporting vocabulary. This vocabulary consists of modeling primitives such as ‘class’, ‘subclass’, ‘property’, ‘subproperty’, ‘domain’ and ‘range’. This allows us to create a hierarchical taxonomy of a particular domain and specify properties of classes of objects. Such a conceptualization of a domain is termed an ‘ontology’.

While RDF-S provides us with a formalism of describing the structure of data, its descriptive power is limited for it to be used to realize the Semantic Web vision. The Darpa Agent Markup Language and Ontology Inference Layer (DAML + OIL) builds upon RDF-S by providing a richer set of constructs to describe information about a particular domain which is machine understandable.
Turning to Web Services now, the Semantic Web envisions the Web to be a repository of not just information content but also of services. These services are proposed to be capable of being discovered, invoked, composed and monitored automatically by software agents. This capability calls for a machine interpretable description of a Web service, which is where DAML-S comes into play. The basic goal of DAML-S is to provide a set of constructs for modeling a Web service. These constructs are in turn defined in the DAML language. DAML-S is expected to provide the following facilities:

- **Automatic Web Service Discovery**: This implies the automatic location of Web services that provide specific services. For example, a user may desire a service that sells books and accepts a particular credit card for payment. Rather than have the user conduct a manual search for such a service, the information describing the service could be expressed in an ontology description language and a search agent could then read and interpret various such descriptions (which we call ‘advertisements’) to find a suitable match.

- **Automatic Web Service Invocation**: This involves the automatic execution of a particular Web service. To do so, the software agent must be able to understand the inputs required of the service, the outputs provided and how to execute the service automatically.

- **Automatic Web Service Composition and Interoperation**: This involves combining several primitive services to achieve a higher level objective. This process is to be automated, which implies that the pre-requisites and effects of the primitive services must be available to the composing agent.
Automatic Web Service Execution Monitoring: This involves ascertaining the status of execution of a Web service at various times after it has been invoked.

DAML-S is an ontology for Web services written in DAML-OIL [3]. The impetus for the specification of DAML-S was provided by the need for automated Web service discovery, composition, execution and monitoring by intelligent agents on the Web. At the root of the ontology is the generic class Service. Instead of classifying the types of services possible, DAML-S specifies the types of information that must be provided for any service in order to fulfill the goals outlined above. Accordingly, the Service class has three sub-classes. The Service Profile class outlines the functionality of the service in terms of the inputs expected by it and the outputs it provides. The Service Model models the service as a process and thus specifies how the service works. The Service Grounding sub-class specifies how the service can be accessed by describing the communications protocol and means of exchanging data with the service.

The Service Profile presents three types of information about the service: information regarding the provider of the service, the functionality of the service and the functional attributes. The functionality of the service is, in turn, represented by specifying the Inputs, Outputs, Preconditions, and Effects of the service (referred to as the IOPEs). The inputs are the information that must be provided to the service and the preconditions are the logical conditions that must hold for the service to be executed. The outputs represent the results of the service. The effects are the events resulting from the execution of the service. For example, an online bookstore could have as inputs the customer details and
credit card information, with the precondition being the validity of the credit card. The
output of the service could be an electronic receipt and the effect would be a transfer of
ownership of the purchased item. Finally, the functional attributes provide certain
descriptive attributes of the service such as the degree of quality and geographic
applicability.

The Process Model is a subclass of the Service Model and provides an exploded view of
the service in terms of its constituent sub-processes. Each sub-process has its own
independent set of IOPEs. DAML-S defines three kinds of processes. Atomic
processes, as the name suggests have no sub-processes and are directly invocable.
They must provide a service grounding to enable the service requester to execute the
process. Simple processes are not directly invocable and are not associated with a
service grounding. They are used to abstract a set of processes by providing a view that is
executable in a single step from the requester’s viewpoint. Hence, a simple process can
be said to provide a ‘black box’ view of the process. Composite processes consist of sub-
processes and the composition is specified in terms of control constructs such as If-then-
else, Iteration, Split, Sequence, Split and Join and Choice. As compared to simple
processes, composite processes provide a ‘glass box’ view of the process.
Chapter 3

Related Work

Paolucci et al [4] have identified 5 essential characteristics of a matching engine.

- The matching engine should support flexible semantic matching on the basis of shared ontologies.
- The requesting service should have some control over the flexibility allowed to the matching process.
- The matching engine should encourage advertisers and requesters to be honest with their descriptions of the service.
- The matching process should be efficient
- The matching engine should minimize the number of false negatives and false positives.

The matching algorithms proposed in [4] are based on DAML ontologies, thereby enabling the matchmaking agent to recognize semantic similarities between the request and the advertisement despite syntactic differences. The algorithm attempts to identify advertisements that can be of some use to the requester. This is achieved by comparing the IOPEs specified in the service model with those specified in the request. To accommodate flexible semantic matching, the request is matched on the basis of the subsumption hierarchy provided by the ontology relating to the concept being matched, rather than syntactical similarity between the request and the advertisement. As such, the
ontology provides the context in which the request and advertisement are interpreted. Hence, they declare that a request for ‘cars’ matches an advertisement for ‘vehicles’, since cars is a category subsumed by the vehicle category.

A match between an advertisement and a request occurs when all the outputs of the request are matched against the outputs of the advertisement and all the inputs of the advertisement are matched against all the inputs of the request i.e. when the service is capable of satisfying the needs of the requester and the requester provides all the inputs to the matched service needed for its operation. Hence, even if one of the request’s outputs is not matched against the outputs of the advertisement, the match fails.

Based on the semantic equivalence of the request and the advertisement, the following categories of matches have been identified [4][5]:

- **Exact**: The match is said to be exact when the requested outputs are the same as the advertised outputs. In such cases, the advertised output can be used to completely satisfy the requested output.

- **Plug-In**: A plug-in match results when the requested output is subsumed by the advertised output. Hence, the advertised service can be substituted or plugged-in in place of the requested service. Such a match is less accurate but is also capable of satisfying the request. As can be seen, an exact match is a special case of the plug-in match. For example, a service that provides all types of vehicles is a plug-in match for a request for cars.
- **Subsumes**: when the requested output subsumes the advertised output. In such a case, the advertised service does not fully satisfy the request. For example, a service that provides cars is a subsumes match for a request that expects vehicles, and the service will only partially satisfy the request.

- **Fail**: In all other cases, the matchmaking results in failure.
Chapter 4

Shortcomings of Matchmaking Based on Service Profile

Matchmaking based on comparing the IOPEs of the request with those specified in the Service Profile of the advertisement is limited in its functionality. Using the Service Model as the basis for matchmaking, we are able to process queries where matchmaking based on the Service Profile fails. The limitations of Service Profile based matchmaking arise due to the nature of the processes comprising the service and the logical relationships underlying the Inputs and Outputs of a process. For instance, consider a service $S$ that is composed of two sub-processes related by the $Choice$ construct, as depicted in Figure 1.

Figure 1: Schematic of a Choice process
In this case, a request specifying output $O_1$ and output $O_2$ would result in a positive match against the service. However, the *Choice* construct mandates that the service can produce either $O_1$ or $O_2$, but not necessarily both the outputs. Hence, the given service would actually fail to satisfy the request. Similarly, a request that specifies only $I_1$ as an input would result in a failed match against the above service, since the service model requires both inputs to be present. In reality however, input $I_1$ is sufficient for the service to execute since only $P_1$ or $P_2$ need to execute. Extending this further, we see that a request specifying input $I_1$ and output $O_2$ should result in a failed match. In other words, the logical relationship amongst the individual inputs/outputs of a process node, as well as the relationship that the inputs of a node have to its outputs, has to be taken into account while determining a match.

Another case where IOPE based matchmaking is not able to provide a satisfactory result is when the service comprises an *if-then-else* construct. The outputs corresponding to the *if-then-else* construct are produced only in the event that the associated condition is satisfied. The conditionality of the outputs is not represented in the service profile at all.

As can be seen, not all the required information for accurate matchmaking is embodied in the Service Profile. In the following sections, we present algorithms for performing matchmaking based on the Process Model.
Chapter 5

Matchmaking Based on the Service Model

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
- Repeat-Until

The processes derive their name from the control structure governing their constituent sub-processes. Processes that are not decomposable any further are known as Atomic Processes. This hierarchical decomposition of processes lends itself naturally to a tree representation where the root corresponds to the composite process representing the...
entire Web service, and the leaves correspond to 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.

In the next few sub-sections, we show the algorithms for matching outputs for various nodes. In these algorithms we will be using the following data structure to represent a DAML-S node:

```c
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.

We establish an inheritance hierarchy of DAML-S nodes, as depicted in Figure 2. Each node type has its own algorithms for matching inputs and outputs.

![Node Inheritance Hierarchy](image)

**Figure 2 : Node Inheritance Hierarchy**

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 $I = i_1, i_2, ..., i_m$ is the list of inputs to be matched against the node and $O = o_1, o_2, ..., o_n$ is the list of outputs to be matched against the outputs of the node.
5.1 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:

Algorithm `matchOutputs(List I, List O, split-seq-Node N)`

```plaintext
if (O is empty)
    return true;

ol ← head (O);

foreach k ∈ N.children
{
    k.matchSet ← k.matchSet ∪ {ol};
    if(matchOutputs(I, k.matchSet, k))
    {
        if (matchOutputs(I, tail (O), N))
            return true;
    }
    k.matchSet ← k.matchSet – {ol};
}

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. The detailed working of the algorithm is as follows:

a) Start with the list of outputs that has to be matched.

b) If the list is empty, return true to imply that all outputs in the list have been matched successfully.

c) Place the first output of the list in the matchSet of the first child.

d) Invoke matchOutputs( ) recursively to match the outputs in the matchSet against the first child.

e) If the match in step (d) fails, remove the output from the current child’s matchSet and try to match this output against the next child of the node in the subsequent iteration of the loop.

f) If the match in step (d) succeeds, try to match the remaining outputs in the list against the node through a recursive call to matchOutputs( ).

g) If the remaining outputs are matched against the node successfully in step (f), return true to indicate that all outputs from the current output onwards have been matched successfully.
h) If, however, it is not possible to match the remaining outputs against the entire node then we backtrack by removing the output from the current child’s matchSet and trying to match it against the next child in the sequence.

i) This process continues until an attempt has been made to match the current output against all the children of the current node.

j) If no match is obtained for the current output or for the remaining outputs of the list, we backtrack to the previous output by returning false.

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. The modified algorithm is given below:
Algorithm matchOutputs(List I, List O, split-seq-Node N)
{
    if (O is empty)
        return true;

    o1 ← head (O).

    foreach k ∈ N.children
    {
        k.matchSet ← k.matchSet ∪ {o1};

        if (matchOutputs(I, k.matchSet, k))
        {
            if ((o1 is not negated) or (o1 is negated and k = tail (N.children))
                if (matchOutputs(I, tail(O), N))
                    return true;
        }
        else {
            k.matchSet ← k.matchSet – {o1};

            if (o1 is negated){
                foreach k we have seen
                
                    k.matchSet ← k.matchSet-{o1};

                return false;
            }
        }
    }
}
return false;

}

5.2 Choice Node

A Choice node consists of a set of processes of which one or more processes are selected for execution.

Algorithm matchOutputs(List I, List O, choice-Node N)
{
    foreach k ∈ N.children
    {
        if (matchOutputs(I, O, k))
            return true;
    }
    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,
\[ T(h) = c T(h-1) \]

The above equation has a solution for \( T(h) = c^h \). Hence, the worst case asymptotic time complexity is \( O(c^h) \), assuming that the tree rooted at \( N \) has height \( h \) and, once again, all the calls to \texttt{matchOutputs} are to this algorithm.

### 5.3 If–Then – Else node

The \textit{If-Then-Else} node selects a particular process from two given process depending on whether an associated condition evaluates to true or false.

In the following algorithm, \texttt{thenPart} is the child node corresponding to the process that is executed if the node condition evaluates to true, and \texttt{elsePart} is the child node corresponding to the process that is executed if the node condition evaluates to false.

**Algorithm** \texttt{matchOutputs(List I, List O, if-then-else-Node N)}

{
   if (N.condition)
      if (matchOutputs(I, O, N.thenPart ))
         return true;
   else
      if (matchOutputs(I, O, N.elsePart) )
         return true;

}
The worst case asymptotic time complexity of this algorithm operating on a tree of If-Then-Else nodes is $O(h)$, where $h$ is the height of the tree.

**5.4 Atomic node**

An **Atomic Node** is a node that has no sub-processes, is directly invocable, and executes in one step from the perspective of the user.

In the algorithm below, we borrow the subset operation from set terminology and apply it to lists. The subset operator determines if all the inputs/outputs belonging to one list are members of the other list.

**Algorithm** matchOutputs (List I, List O, atomic-Node N)

```java
public boolean matchOutputs(List I, List O, atomic-Node N) {
    if (O ⊂ N.outputs and N.inputs ⊂ I )
        return true;
    return false;
}
```
We assume that the subset operation executes in constant time, thus giving us an asymptotic time complexity of this algorithm as $O(1)$.

### 5.5 Matchmaking based on subsumption hierarchy

The algorithms presented thus far have not taken into account the subsumption hierarchies of the outputs. Incorporating the subsumption hierarchies will enable us to determine the degree of match and differentiate among exact, plug-in, and subsumes matches. In this section we modify the basic algorithm for the split-sequence node to incorporate subsumption hierarchies. Similar modifications can be made to the algorithms for other nodes.

We modify the node data structure that represents a DAML-S node by adding a variable `matchDegree`, which represents the degree of match achieved for the outputs matched at any point in time. This variable can take one of the four possible values: exact, plug-in, subsumes, and fails. These values have an ordering imposed on them such that exact > plug-in > subsumes > fails.

In order to achieve this type of matching we had to re-write the `matchOutputs` function for the atomic node. Specifically, the `subset` operation defined in the algorithm is extended to determine the actual degree of match among the two sets of outputs. That is, instead of returning only true or false based on exact matches it uses our induction engine to determine if the match is either exact, plug-in, subsumed, or there is no match.
possible. This match is achieved with the use of the Jess knowledge base and the
inference rules defined by the DAML semantics. The DAML inference rules are asserted
into the Jess database along with the facts from the DAML-S advertisement and the
ontology for the outputs. The Atomic match algorithm asks Jess whether each output in
the query is either a plug-in, subsumes, or exact match of each output in the node. In this
way we let Jess perform all the inferencing required to prove a match between outputs.

For a sequence node, the overall degree of match for a set of outputs corresponds to the
lowest degree of match obtained among its child nodes. Accordingly, the algorithm
presented in Section 5.1 is modified as below:

Algorithm matchOutputs(List I, List O, spli-seq-Node N)
{
    N.matchDegree ← exact;
    if (O is empty)
        return true;
    o1 ← head (O)
    foreach k ∈ N.children
    {
        k.matchSet ← k.matchSet ∪ {o1};
        if(matchOutputs(I, k.matchSet, k))
        {
            if(k.matchDegree < N.matchDegree)
        }
    }
}
N.matchDegree ← k.matchDegree;

if((o1 is not negated) or
   (o1 is negated and
    k = tail(N.children))
if(matchOutputs(I, tail(O), N))
    return true;
}
k.matchSet ← k.matchSet – {o1};
if (o1 is negated)
{
    foreach k we have seen
    k.matchSet ← k.matchSet – {o1};
    N.matchDegree ← fails;
    return false;
}
N.matchDegree ← fails;
return false;}
Chapter 6

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 [6] package, which converts a DAML file into a set of equivalent Subject-Verb-Object (SVO) triples. These triples are then asserted into the JESS [7] 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 schematic flow diagram of the process is given in Figure 3. The use of the JESS and DAMLJessKB reduced our development time and facilitated the matching based on the subsumption hierarchy.

A tree representation of the advertisement used for testing is given in Figure 4. The advertisement we used is a hypothetical description of an online PC retailer that requires users to have an account in order to buy their PC. The user can choose which parts to use, whether to buy or lease, and which kind of shipping to use. The root node represents the entire advertisement, which is composed of a number of processes executed in sequence.
The leaf nodes correspond to the atomic processes of the advertisement. The inputs and outputs of these atomic nodes are listed in Table 1.

<table>
<thead>
<tr>
<th>Atomic Process</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Account</td>
<td>Name, Password, Credit Card No., Credit Card Type</td>
<td></td>
</tr>
<tr>
<td>Load Account</td>
<td>Name, Password</td>
<td></td>
</tr>
<tr>
<td>Select Vendor</td>
<td>Customer’s choice of a vendor (Intel, AMD…) for a sub-component</td>
<td></td>
</tr>
<tr>
<td>Add Component</td>
<td>Component to be added and its vendor</td>
<td></td>
</tr>
<tr>
<td>Lease</td>
<td>Lease Period</td>
<td>Lease Rental</td>
</tr>
<tr>
<td>Hire Purchase</td>
<td>HP Period</td>
<td>Installment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Amount</td>
</tr>
<tr>
<td>Direct Sale</td>
<td></td>
<td>Sale Price</td>
</tr>
<tr>
<td>Waive Charges</td>
<td>Purchase amount</td>
<td></td>
</tr>
<tr>
<td>Levy Charges</td>
<td>Purchase amount</td>
<td>Shipping Charges</td>
</tr>
<tr>
<td>Delivery Type</td>
<td>Delivery Type</td>
<td></td>
</tr>
<tr>
<td>Finalize Buy</td>
<td></td>
<td>E-Receipt</td>
</tr>
</tbody>
</table>

**Table 1 : Inputs and Outputs for the Sample Advertisement**

A sample query posed to this advertisement and expressed in our Query Language is presented below:

*find service with*

*inputs =*

(http://www.cse.sc.edu/~bansal/PCStore1.daml.txt#Name)
and http://www.cse.sc.edu/~bansal/PCStore1.daml.txt#Password
and http://www.cse.sc.edu/~bansal/PCStore1.daml.txt #Input ComponentType and
http://www.cse.sc.edu/~bansal/PCStore1.daml.txt #InputVendorName and
http://www.cse.sc.edu/~bansal/PCStore1.daml.txt#LeasePeriod )
outputs =
(http://www.cse.sc.edu/~bansal/PCStore1.daml.txt#E-Receipt)
condition =
http://www.cse.sc.edu/~bansal/PCStore1.daml.txt#ShippingDiscountPrice.ge.5000

The above query requests a service that accepts, among other inputs, the lease period and
gives an electronic receipt as the output. The query results in a positive match even
though we are not specifying all the inputs possible, since it is possible to execute the
service with the inputs provided and produce the desired output.

The software for this implementation and the DAML-S file for the advertisement can be
downloaded from our project homepage at http://jmvidal.cse.sc.edu/matchmaking/.
- Rectangles represent documents
- Hexagons represent processes

Figure 3: Schematic Flow Diagram of Implementation of Matchmaking Algorithms
The DAML-S source code is available from our project homepage http://jmvidal.cse.sc.edu/ matchmaking/
Chapter 7

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. We extended our basic algorithms to incorporate subsumption hierarchies of the inputs and outputs, thus enabling us to distinguish between exact, plug-in, and subsumes matches. We have also implemented and tested these algorithms using a DAML-S advertisement comprising various types of process nodes, and queries corresponding to varying user requirements. The results obtained on running these tests show that matchmaking based on the Service Model is capable of determining matches in cases where IOPE-based matchmaking is unable to do so. 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. This is possible because once a particular output fails to match against a particular node all possible output configurations having
this node-output combination are discarded, thus pruning the search space. We are currently looking at ways to improve the time complexity of the algorithm. Matchmaking based on the Service Model therefore overcomes the inherent limitations of Service Profile based matchmaking, and is capable of supporting more detailed queries.

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. That is, in order for agents to compose aggregate Web services they will first need to be able to find these services by identifying certain restrictions on how these services are composed, such as which other sub-services a particular service uses. Our algorithms form a first step towards the automatic construction of sophisticated aggregate Web services.
References


http://www.daml.org/services/daml-s/2001/05/daml-s.html


[6] DAMLJessKB.

http://plan.mcs.drexel.edu/projects/legorobots/design/software/DAMLJessKB/


http://herzberg.ca.sandia.gov/jess/
Appendix 1

Test Cases

1.  
Query:
find service with
inputs = not name1
outputs = http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#E-Receipt
condition =
   http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#ShippingDiscountPrice.ge.5000

Comments: Negation of an input not required by the advertisement

Results:
   Inputs : match
   Outputs: match
   MatchDegree: exact

2.  
Query:
find service with
inputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Name
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Password
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LoginID
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#CreditCardNumber
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#CreditCardType
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputVendorName
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputComponentType
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeasePeriod
and not http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#HPPeriod
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputTransactionType)
outputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#E-Receipt
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#SalePrice)
condition =
   http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#ShippingDiscountPrice.ge.5000

Comments : Input of one branch of an Choice node with negation of the input of the other branch of the Choice node
3.
Query: find service with
inputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Name
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Password
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LoginID
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputVendorName
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputComponentType
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeasePeriod
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputTransactionType)
outputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#E-Receipt
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#SalePrice)
condition =
   http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#ShippingDiscountPrice.ge.5000

Comments: Input of only one branch of the Choice node specified

Results:
   Inputs : match
   Outputs: match
   MatchDegree: exact

4.
Query: find service with
inputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Name
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Password
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LoginID
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#CreditcardType
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#CreditCardNumber
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputVendorName
and not http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeasePeriod
and not http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#HPInput
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputTransactionType)
outputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#E-Receipt
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#SalePrice)
condition =
   http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#ShippingDiscountPrice.ge.5000

Comments: Negation of inputs on two of the branches of a Choice node
Results:
Inputs : match
Outputs: match
MatchDegree: exact

5.
Query:
find service with
inputs = (not http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Name
and not http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Password
and not http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LoginID
and not http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#CreditcardType
and not http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#CreditCardNumber
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputComponentType
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputVendorName
and not http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeasePeriod
and not http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#HPInput
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputTransactionType)
outputs = (and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#E-Receipt
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#SalePrice)
condition =
http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#ShippingDiscountPrice.ge.5000

Comments: Negation of inputs for all branches of a Choice node

Results:
Inputs : no match
Outputs: no match
MatchDegree: fails

6.
Query:
find service with
inputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Name
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Password
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputComponentType
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputVendorName
and not http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeasePeriod
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputTransactionType)
outputs = (not http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#E-Receipt
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#SalePrice)
condition =
http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#ShippingDiscountPrice.ge.5000

Comments: Negation of one of the outputs of a Sequence Node
7.  
**Query:**
find service with 
inputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Name 
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Password 
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputComponentType 
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputVendorName 
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeasePeriod 
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputTransactionType) 
outputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#E-Receipt 
and not http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#SalePrice) 
condition = 
   http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#ShippingDiscountPrice.ge.5000

**Comments:** Negation of one of the outputs of a Choice node, with inputs to another branch of the same Choice node specified.

**Results:**
- **Inputs:** match
- **Outputs:** match
- **MatchDegree:** exact

8.  
**Query:**
find service with 
inputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Name 
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Password 
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputComponentType 
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputVendorName 
and not http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeasePeriod 
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputTransactionType) 
outputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#E-Receipt 
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeaseRental) 
condition = 
   http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#ShippingDiscountPrice.ge.5000

**Comments:** Output of a Choice node with the corresponding inputs negated.

**Results:**
- **Inputs:** match
9. 
Query:
find service with
inputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Name
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Password
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputComponentType
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputVendorName
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeasePeriod
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputTransactionType)
outputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeaseReceipt
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeaseRental)
condition =
http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#ShippingDiscountPrice.ge.5000

Comments: Plug-in Match

Results:
Inputs : match
Outputs: match
MatchDegree: plug-in

10. 
Query:
find service with
inputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Name
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Password
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputComponentType
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputVendorName
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeasePeriod
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputTransactionType)
outputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#E-Receipt
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InstalmentAmount)
condition =
http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#ShippingDiscountPrice.ge.5000

Comments: Input of one branch of a Choice node, with the output of the other branch

Results:
Inputs : match
Outputs: no match
MatchDegree: fails
11.
Query:
find service with
inputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Name
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Password
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputComponentType
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputVendorName
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeasePeriod
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputTransactionType)
outputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#E-Receipt
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Rental)
condition =
http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#ShippingDiscountPrice.ge.5000

Comments: Subsumption match

Results:
Inputs : match
Outputs: match
MatchDegree: subsumes

12.
Query:
find service with
inputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Name
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Password
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputComponentType
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputVendorName
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeasePeriod
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputTransactionType)
outputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#E-Receipt
and not http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#PCConfiguration)
condition =
http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#ShippingDiscountPrice.ge.5000

Comments: Negated output of the sequence node and plug-in output of choice node

Results:
Inputs : match
Outputs: no match
MatchDegree: fails

13.
Query:
find service with
inputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Name
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Password
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputComponentType
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputVendorName
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeasePeriod
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputTransactionType)
outputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#E-Receipt
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeaseRental
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InstalmentAmount)
condition =
    http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#ShippingDiscountPrice.ge.5000

Comments: Outputs for two branches of the same Choice node

Results:
    Inputs : match
    Outputs: no match
    MatchDegree: fails

14.
Query:
find service with
inputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Name
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#Password
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputComponentType
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputVendorName
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeasePeriod
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#InputTransactionType)
outputs = (http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#E-Receipt
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#LeaseRental
and http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#ShippingCharges)
condition =
    http://www.cse.sc.edu/_bansal/PCStore1.daml.txt#ShippingDiscountPrice.ge.5000

Comments: Condition for If clause satisfied and output corresponding to the else clause

Results:
    Inputs : match
    Outputs: no match
    MatchDegree: fails
Appendix 2

Code Listing

/*
 * MatchAgent.java
 * This class represents a Match Making agent
 * Created on July 18, 2002, 1:31 PM
 *
 * @author Sharad Bansal
 * @version
 */

import java.util.*;
import edu.drexel.itcsl.daml.*;
import jess.*;
import jess.awt.*;
import jess.factory.*;
import java.io.*;

public class MatchAgent extends Object
{
    //table for storing the nodes of the advertisement
    private HashMap table = new HashMap();
    //instance of DAML to Jess engine for the advertisement
    private DAMLJessKB damlJess;
    //instance of DAML to Jess engine for ontology of outputs
    public static DAMLJessKB ontoKB;

    //constants for the degree of match
    public static final int exact = 4;
    public static final int plugIn = 3;
    public static final int subsumes = 2;
    public static final int fails = 1;

    /** Creates new MatchAgent */
    public MatchAgent()
    {
        try
        {
            damlJess = new DAMLJessKB();
            ontoKB = new DAMLJessKB();
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }

    //checks whether the given process is a composite process
    private boolean isCompositeProcess(String pName)
    {
        boolean answer = false;
        try
        {
            String defineQuery = "(defquery search(PropertyValue
http://www.daml.org/services/daml-s/2001/10/Process.daml#composedOf " + pName
            damlJess.evaluate(defineQuery);
            if (ontoKB.match(defineQuery))
                answer = true;
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
        return answer;
    }

    //Method to add advertisement to the table
    public void addAdvertisement(String adName, String adDesc)
    {
        table.put(adName, adDesc);
    }

    //Method to search for advertisements
    public String searchAd(String query)
    {
        return (String) table.get(query);
    }
}

damlJess.rete.executeCommand(defineQuery);
damlJess.rete.executeCommand("(store RESULT (run-query search))");
Iterator i = (Iterator)
damlJess.rete.fetch("RESULT").externalAddressValue(null);
if (i.hasNext())
    answer = true;
else
    answer = false;
}
catch (Exception e)
{
    System.out.println(e);
}
return answer;
}

executeQuery(String query, int n)
{
    ArrayList v = null;
    try
    {
        // Define and run the query
        String defineQuery = "(defquery search " + query + ")";
        damlJess.rete.executeCommand(defineQuery);
        damlJess.rete.executeCommand("(store RESULT (run-query search))");

        // Fetch the result (an Iterator).
        Iterator e = (Iterator)
damlJess.rete.fetch("RESULT").externalAddressValue(null);
        v = new ArrayList();

        // Pick each element of the Iterator apart and store the
        // desired part in the ArrayList v.
        while (e.hasNext())
        {
            Token t = (Token) e.next();

            // We want the second fact in the token - the first is the
            // query trigger
            Fact f = t.fact(1);

            // The first and only slot of this fact is the __data
            multislot.
            ValueVector multislot = f.get(0).listValue(null);

            // The n-th elements of this slot is the datum we're interested
            in.
            v.add(multislot.get(n).stringValue(null));
        }
    }
    catch (Exception e)
    {
        System.out.println(e);
    }

    return v;
}
//determines the type of a composite process
private String findNodeType(String pName) {
    String nodeType = null;
    try {
        damlJess.executeCommand("(defquery search5 (PropertyValue
            http://www.daml.org/services/daml-s/2001/10/Process.daml#composedOf ?x ?y))");
        //run query to find the control construct for the composite process
        damlJess.rete.executeCommand("(defquery search3 (PropertyValue
            http://www.daml.org/services/daml-s/2001/10/Process.daml#composedOf " + pName
            + " ?y))");
        damlJess.rete.executeCommand("(store RESULT3 (run-query search3))");
        Iterator e3 = (Iterator)damlJess.rete.fetch("RESULT3").externalAddressValue(null);
        String composedOfNode = null;
        Token t3 = (Token)e3.next();
        Fact f3 = t3.fact(1);
        ValueVector multislot3 = f3.get(0).listValue(null);
        composedOfNode = multislot3.get(2).stringValue(null);

        //Determine node type
        damlJess.executeCommand("(defquery search4 (PropertyValue
            http://www.w3.org/2000/01/rdf-schema#subClassOf "
            + composedOfNode.toString() + " ?y))");
        damlJess.rete.executeCommand("(store RESULT4 (run-query search4))");
        Iterator e4 = (Iterator)damlJess.rete.fetch("RESULT4").externalAddressValue(null);
        Token t4 = (Token)e4.next();
        Fact f4 = t4.fact(1);
        ValueVector multislot4 = f4.get(0).listValue(null);
        nodeType = multislot4.get(2).stringValue(null);
    } catch (Exception e) {
        System.out.println(e);
    }
    return nodeType;
}

//creates a node depending on the type of the node
private BTNode createNode(String nodeName, String nodeType) {
    BTNode node = null;
    //create appropriate node
    if (nodeType.equals("http://www.daml.org/services/daml-s/2001/10/Process.daml#Sequence"))
        node = new SequenceNode(nodeName);
    if (nodeType.equals("http://www.daml.org/services/daml-s/2001/10/Process.daml#Choice"))
        node = new ChoiceNode(nodeName);
    if (nodeType.equals("http://www.daml.org/services/daml-s/2001/10/Process.daml#Alternative"))
        node = new ChoiceNode(nodeName);
    if (nodeType.equals("http://www.daml.org/services/daml-s/2001/10/Process.daml#If-Then-Else"))
        node = new If_Then_ElseNode(nodeName, this);
    setIfCondition(nodeName, node);
    if (nodeType.equals("http://www.daml.org/services/daml-s/2001/10/Process.daml#Repeat-Until"))
        node = new RepeatUntilNode(nodeName);
    if (nodeType.equals("http://www.daml.org/services/daml-s/2001/10/Process.daml#Split"))
        node = new SplitNode(nodeName, this);
    return node;
}
node = new SplitNode(nodeName);
table.put(nodeName, node);
return node;
}

private void setIfCondition(String nodeName, BTNode node)
{
    String query = "(PropertyValue http://www.daml.org/services/daml-s/2001/10/Process.daml#ifCondition "
            + nodeName
            + " ?y)");
    ArrayList v = executeQuery(query, 2);
    Iterator iter = v.iterator();
    String conditionClass = (String) iter.next();
    query = "(PropertyValue http://www.w3.org/1999/02/22-rdf-syntax-ns#type "
            + "?x "
            + conditionClass
            + ")";
    v = executeQuery(query, 1);
    iter = v.iterator();
    String condition = (String) iter.next();
    If_Then_ElseNode ifNode = (If_Then_ElseNode) node;
    ifNode.setIfCondition(condition);
}

//main method
public static void main(String argv[])
{
    try
    {
        MatchAgent myAgent = new MatchAgent();
        String fnurl = "http://www.cse.sc.edu/~bansal/PCStore1.daml.txt";
        System.out.println("Parsing advertisement " + fnurl + " ...
");
        myAgent.damlJess.processKB();
        myAgent.damlJess.loadDAMLFilenameOrURL(fnurl);
        MatchAgent.ontoKB.processKB();

        MatchAgent.ontoKB.loadDAMLFilenameOrURL("http://www.cse.sc.edu/~bansal/RentalOntology.txt");
        BTNode node = null;

        //run query to find all composite processes
        System.out.println("Building advertisement tree...");
        myAgent.damlJess.rete.executeCommand("(store RESULT1 (run-query search1))");

        // Fetch the result (an Iterator).
        Iterator el = (Iterator) myAgent.damlJess.rete.fetch("RESULT1").externalAddressValue(null);
        ArrayList vl = new ArrayList();

        while (el.hasNext())
        {
            Token t1 = (Token) el.next();
            Fact f1 = t1.fact(1);
            ValueVector multislot1 = f1.get(0).listValue(null);

            //create a tree node for the composite process, if one does not already exist
        }
    }
}
String nodeName = (multislot1.get(1).stringValue(null));
String composedOfNode = (multislot1.get(2).stringValue(null));

// determine node type
String nodeType = myAgent.findNodeType(nodeName);

// create a node if one does not already exist
if (myAgent.table.get(nodeName) == null)
    node = myAgent.createNode(nodeName, nodeType);
else node = (BTNode)myAgent.table.get(nodeName);

// find all child nodes of this composite node - these children
// can be composite nodes or atomic nodes
String query2 = "(PropertyValue
http://www.daml.org/services/daml-s/2001/10/Process.daml#components "
    + composedOfNode
    + " ?y)";
ArrayList v2 = myAgent.executeQuery(query2, 2);

for (Iterator answers2 = v2.iterator(); answers2.hasNext();)
{
    BTNode childNode;
    String childName = (answers2.next()).toString();
    if(myAgent.isCompositeProcess(childName))
    {
        if (myAgent.table.get(childName) == null)
        {
            String childNodeType =
myAgent.findNodeType(childName);
            childNode = myAgent.createNode(childName, childNodeType);
            // set the link
            node.setLink(childNode);
            childNode.setChild(true);
        }
        else
        {
            childNode =
            (BTNode)myAgent.table.get(childName);
            node.setLink(childNode);
            childNode.setChild(true);
        }
    }
    else
    {
        childNode = new AtomicNode(childName);
        myAgent.table.put(childName, childNode);
        // set the link
        node.setLink(childNode);
        childNode.setChild(true);
        // obtain outputs of the node
    }
}

// find all child nodes of this composite node - these children
// can be composite nodes or atomic nodes
String query5 = "(PropertyValue
http://www.w3.org/2000/01/rdf-schema#domain "
    + " ?x "
    + childName
    + ")";
ArrayList a = myAgent.executeQuery(query5, 1);

for (Iterator answers5 = a.iterator(); answers5.hasNext();)
{
    String property = (String)answers5.next();
    String query6 = "(PropertyValue
http://www.w3.org/2000/01/rdf-schema#subPropertyOf "
    + " ?y "
    + property
    + 
```
myAgent.executeQuery(query6, 2);
answers6.hasNext();
{
    String propertyType =
    AtomicNode atomicNode =
    if (propertyType.equals("http://www.daml.org/services/daml-s/2001/10/Process.daml#output"))
    {
        atomicNode.outputs.add(property);
    }
    else
    {
        atomicNode.inputs.add(property);
    }
}
boolean isMatch = false;
while (!outputStack.isEmpty())
{
    Set outset = (Set)outputStack.pop();
    String op = null;
    Iterator iter = outset.iterator();
    if (iter.hasNext())
    {
        Object obj = (Object)iter.next();
        if (obj instanceof String)
        {
            op = (String) obj;
            System.out.println(op);
        }
    }
    if (op != null && op.equals("any"))
    {
        isMatch = true;
        matchDegree = MatchAgent.exact;
    }
    else
    {
        isMatch = root.matchOutputs(inset, outset);
        if (root.getMatchDegree() > matchDegree)
        {
            matchDegree = root.getMatchDegree();
        }
    }
}
if(isMatch)
    System.out.println("Outputs : Match");
else System.out.println("Outputs : No match");

//give degree of match
switch(matchDegree)
{
    case exact :
    {
        System.out.println("Degree of match : Exact");
        break;
    }
    case plugIn :
    {
        System.out.println("Degree of match : PlugIn");
        break;
    }
    case subsumes :
    {
        System.out.println("Degree of match : Subsumes");
        break;
    }
    case fails :
    {
        System.out.println("Degree of match : Fails");
        break;
    }
}
}
catch (Exception e)
{
    System.out.println(e);
    e.printStackTrace();
}
import java.util.*;

public class SequenceNode extends BTNode {
    //the degree of match for outputs
    private int matchDegree = MatchAgent.exact;
    /** Creates new SequenceNode */
    public SequenceNode(String name) {
        super(name);
        nodeType = "Sequence Node";
    }

    /**
     * matchInputs : matche the inputs of the sequnce node against the query inputs
     * @param in The set of query inputs
     */
    public boolean matchInputs (Set in) {
        //get the number of children
        int numChildren = this.getNumChildren();

        //get the children of the sequence node
        BTNode[] children = this.getLinks();

        //match inputs for all the children
        for (int i = 0; i < numChildren; i++) {
            //if any of the children fails to match on inputs, then the entire sequence fails
            if (!children[i].matchInputs(in))
                return false;
        }
        return true;
    }

    /**
     * match the outputs for this node
     * @param in the set of inputs to be matched
     * @param p the set of outputs to be matched
     */
    public boolean matchOutputs (Set in, Set p) {
        System.out.println("Now in node: " + getData());
    }
}
System.out.println("Matching outputs : " + p);
if (p.size() == 0)
    return true;

Iterator iter = p.iterator();
IOPair firstOutput = (IOPair)iter.next();

// set the degree of match to the maximum possible
matchDegree = MatchAgent.exact;

// get the number of children of this current sequence node
int numChildren = this.getNumChildren();

// get the array of children of the sequence node
BTNode[] children = this.getLinks();

System.out.println ("children are : ");
for (int i = 0; i < numChildren; i++)
{
    System.out.println(children[i].getData());
}

for (int k = 0; k < numChildren; k++)
{
    System.out.println("K = " + k);
    // add the output being matched to the output set of the current
    children[k].set.add(firstOutput);
    BTNode child = children[k];
    boolean val = child.matchOutputs(in, child.set);
    if (val)
    {
        if (matchDegree > children[k].getMatchDegree())
            matchDegree = children[k].getMatchDegree();
        if ((firstOutput.matchFlag) || (!firstOutput.matchFlag && (k == numChildren - 1)))
        {
            p.remove(firstOutput);
            if (matchOutputs(in, p))
                return true;
        }
    }

    else
    {
        // remove the output from the current node's set
        children[k].set.remove(firstOutput);
        if (!firstOutput.matchFlag)
        {
            for (int m = 0; m < k; m++)
            {
                children[k].set.remove(firstOutput);
                matchDegree = MatchAgent.fails;
                return false;
            }
        }
    }
}

matchDegree = MatchAgent.fails;
return false;
public String toString()
{
    return ("Sequence node: " + this.getData());
}

public int getMatchDegree()
{
    return matchDegree;
}

/*
* ChoiceNode.java
* This Class represents a Choice Process node
* Created on July 17, 2002, 7:06 PM
* 
* @author Sharad Bansal
* @version
*/
import java.util.*;
public class ChoiceNode extends BTNode
{
    //the degree of match for outputs of this node
    private int matchDegree;

    /** Creates new ChoiceNode */
    public ChoiceNode(String name)
    {
        super(name);
    }

    /*
    * matches the inputs of this node against the query inputs
    * @param in The set of query inputs
    */
    public boolean matchInputs(Set in)
    {
        //get the children of this node
        BTNodel[] children = this.getLinks();

        //get the number of children of this node
        int numChildren = this.getNumChildren();

        //match inputs for each child
        for (int k = 0; k < numChildren; k++)
        {
            BTNodel child = children[k];
            //if any child satisfies the inputs, then the entire node matches
            the inputs
            if (child.matchInputs(in))
            {
                return true;
            }
        }
    }
}
/**
* match the outputs for this node
* @param in the set of inputs to be matched
* @param p the set of outputs to be matched
*/
public boolean matchOutputs(Set in, Set p)
{
    //set degree of match to 'fails'
    matchDegree = MatchAgent.fails;

    //get the children of this node
    BTNode[] children = this.getLinks();

    //get the number of children of this node
    int numChildren = this.getNumChildren();

    //do for each child
    for (int k = 0; k < numChildren; k++)
    {
        BTNode child = children[k];

        //match outputs for each child; if the outputs for any one child
        //match, then the
        //match for the entire Choice node succeeds
        if (child.matchOutputs(in, p))
        {
            //set the match degree to the greatest degree amongst all
            children
            if (child.getMatchDegree() > matchDegree)
            {
                matchDegree = child.getMatchDegree();
            }
        }
    }

    return false;
}

//get the degree of match for this node
public int getMatchDegree()
{
    return matchDegree;
}

//return the name of this node
public String toString()
{
    return ("Choice node: " + this.getData());
}
import java.util.*;

public class If_Then_ElseNode extends BTNode {
    private int matchDegree;
    private String condition;
    private BTNode thenNode;
    private BTNode elseNode;
    private MatchAgent myAgent;

    /** Creates new If_Then_ElseNode */
    public If_Then_ElseNode(String name, MatchAgent matchAgent) {
        super(name);
        myAgent = matchAgent;
    }

    //sets the condition
    public void setIfCondition(String condition) {
        this.condition = condition;
    }

    //returns the match degree of the node
    public int getMatchDegree() {
        return matchDegree;
    }

    //sets the 'then' and 'else' nodes
    public void setNodes() {
        String query = "(PropertyValue http://www.daml.org/services/daml-
        s/2001/10/Process.daml#then " + getData() + " ?y)";
        ArrayList v = myAgent.executeQuery(query, 2);
        Iterator iter = v.iterator();
        String thenNodeName = null, elseNodeName = null;
        while (iter.hasNext())
            thenNodeName = (String) iter.next();
        query = "(PropertyValue http://www.daml.org/services/daml-
        s/2001/10/Process.daml#else " + getData() + " ?y)";
        v = myAgent.executeQuery(query, 2);
        iter = v.iterator();
        while (iter.hasNext())
            elseNodeName = (String) iter.next();
        for (int i = 0; i < getNumChildren(); i++) {
            if (thenNodeName.equals(links[i].getData()))
                thenNode = links[i];
            if (elseNodeName.equals(links[i].getData()))
                elseNode = links[i];
        }
    }
}
elseNode = links[i];
}
}

// matches the given inputs against the inputs for this node
public boolean matchInputs(Set in)
{
    setNodes();
    boolean flag;
    if (QueryParser.condition.equals(this.condition))
        flag = thenNode.matchInputs(in);
    else
        flag = elseNode.matchInputs(in);

    return flag;
}

// matches the given outputs against the outputs for this node
public boolean matchOutputs(Set in, Set p)
{
    setNodes();
    boolean flag;
    if (QueryParser.condition.equals(this.condition))
        {
            flag = thenNode.matchOutputs(in, p);
            matchDegree = thenNode.getMatchDegree();
        } else
        {
            flag = elseNode.matchOutputs(in, p);
            matchDegree = elseNode.getMatchDegree();
        }

    return flag;
}

/*
 * AtomicNode.java
 * This Class represents an atomic process node
 * Created on July 23, 2002, 10:18 PM
 *
 * @author Sharad Bansal
 * @version */
import java.util.*;
import edu.drexel.itcsl.*;
import jess.*;

public class AtomicNode extends BTNode
{
    public Set outputs; // the outputs of the atomic node
    public Set inputs; // the inputs of the atomic node
    private int matchDegree; // the degree of match for this node

    /** Creates new AtomicNode object */
    public AtomicNode(String name)
super(name);
nodeType = "Atomic Node";
outputs = new HashSet();
inputs = new HashSet();

// gets the degree of match
public int getMatchDegree()
{
    return matchDegree;
}

public boolean matchInputs(Set in)
{
    Iterator i = inputs.iterator();
    boolean flag = true;
    boolean matchFlag = false;
    while (i.hasNext() && flag)
    {
        flag = false;
        String nodeIp = (String) i.next();
        Iterator j = in.iterator();
        while(j.hasNext())
        {
            IOPair ioPair = (IOPair) j.next();
            String ip = (String) ioPair.url;
            matchFlag = ioPair.matchFlag;
            if (ip.equals(nodeIp))
            {
                flag = matchFlag;
            }
        }
    }
    return flag;
}

/**
* Method for matching a set of outputs against the given
* inputs and outputs
*/
public boolean matchOutputs(Set in, Set p)
{
    if (subsetOf(p, outputs) && matchInputs(in))
    {
        return true;
    }
    return false;
}

// method to execute a jess query
private HashSet executeQuery(String query, int n)
{
    HashSet result = null;
    try
    {
        String defineQuery = "(defquery search " + query + ")";
        MatchAgent.ontoKB.rete.executeCommand(defineQuery);
        MatchAgent.ontoKB.executeCommand("(store RESULT (run-query search))");

        // Fetch the result (an Iterator).
        Iterator e = (Iterator)
            MatchAgent.ontoKB.rete.fetch("RESULT").externalAddressValue(null);
        result = new HashSet();
    }
    catch (Exception e)
    {
        result = null;
    }
    return result;
}
// Pick each element of the Iterator apart and store the
// interesting part in the ArrayList v.
while (e.hasNext())
{
    Token t = (Token) e.next();
    // We want the second fact in the token - the first is the
query trigger
    Fact f = t.fact(1);
    // The first and only slot of this fact is the __data
multislot.
    ValueVector multislot = f.get(0).listValue(null);
    // The n-th elements of this slot is the datum we're
interested in.
    result.add(multislot.get(n).stringValue(null));
}
}
catch (Exception e)
{
    System.out.println(e);
}
return result;

//returns true if the intersection of 2 sets is not null
private boolean intersectionOf(Set a, Set b)
{
    Iterator iter = b.iterator();
    while (iter.hasNext())
    
        if (a.contains(changeNameSpace((String)iter.next())))
            return true;
    return false;
}

/*
 //determines whether a set of inputs is a subset of another set of inputs
 private boolean inputSubsetOf(Set a, Set b)
 {
     Iterator i = a.iterator();
     boolean flag = true;
     boolean matchFlag = false;
     while (i.hasNext() && flag)
     {
         flag = false;
         String nodeIp = (String) i.next();
         Iterator j = b.iterator();
         while(j.hasNext())
         {
             IOPair ioPair = (IOPair) j.next();
             String ip = (String) ioPair.url;
             matchFlag = ioPair.matchFlag;
             if (ip.equals(nodeIp))
             {
                 flag = matchFlag;
             }
         }
     }
     return flag;
 }
*/

//checks whether a set of outputs is a subset of another set of outputs
//Also determines the degree of match
private boolean subsetOf(Set a, Set b)
{
    Iterator i = a.iterator();
    boolean flag = true;
    while (i.hasNext())
    {
        IOPair ioPair = (IOPair) i.next();
        String ip = (String) ioPair.url;
        matchFlag = ioPair.matchFlag;
        if (ip.equals(nodeIp))
        {
            flag = matchFlag;
        }
    }
    return flag;
}
boolean matchFlag = false;
matchDegree = MatchAgent.exact;
while (i.hasNext() && flag)
{
    IOPair ioPair = (IOPair) i.next();
    String op = (String) ioPair.url;
    matchFlag = ioPair.matchFlag;
    //exact match
    if (b.contains(op))
    {
        flag = matchFlag;
        if (!flag)
            matchDegree = MatchAgent.fails;
    }
    else
    {
        String op1 = changeNameSpace(op);
        Iterator k = b.iterator();
        String query = "(PropertyValue http://www.w3.org/2000/01/rdf-schema#subclassOf "
                    + op1
                    + " ?y")";
        HashSet result = executeQuery(query, 2);
        if (intersectionOf(result, b))
        {
            flag = matchFlag;
            //set the matchDegree to the minimum degree so far
            if (!flag)
            {
                if (matchDegree > MatchAgent.plugIn)
                    matchDegree = MatchAgent.plugIn;
            }
            else matchDegree = MatchAgent.fails;
        }
        else //subsumption match
        {
            query = "(PropertyValue http://www.w3.org/2000/01/rdf-schema#subclassOf "
                    + op1
                    + " ?y "
                    + op1
                    + ")";
            result = executeQuery(query, 1);
            if (intersectionOf(result, b))
            {
                flag = matchFlag;
                if (!flag)
                {
                    if (matchDegree > MatchAgent.subsumes)
                        matchDegree = MatchAgent.subsumes;
                }
                else
                {
                    flag = !matchFlag;
                    if (!flag)
                    {  
                        matchDegree = MatchAgent.fails;
                    }
                }
            }
            else
            {
                flag = !matchFlag;
                if (!flag)
                {  
                    matchDegree = MatchAgent.fails;
                }
            }
        }
    }
}
return flag;
private String changeNameSpace(String s)
{
    return (http://www.cse.sc.edu/_bansal/RentalOntology.txt#" + s.substring(s.indexOf('#') + 1));
}

public String toString()
{
    return ("Atomic node: " + this.getData());
}

import java.util.*;
public abstract class BTNode
{
    private Object data;
    protected BTNode[] links;
    protected String nodeType;
    private int numChildren = 0;
    private BTNode left, right;
    private boolean child = false;
    public HashSet set;
    public BTNode()
    {
    }
    public BTNode (Object initialData)
    {
        data = initialData;
        links = new BTNode[10];
        set = new HashSet();
    }
    public abstract boolean matchOutputs(Set in, Set p);
    public abstract boolean matchInputs(Set in);
    public abstract int getMatchDegree();
}
public BTNode[] getLinks()
{
    return links;
}

public String getNodeType()
{
    return nodeType;
}

public int getNumChildren()
{
    return numChildren;
}

public void setLink(BTNode node)
{
    links[numChildren] = node;
    numChildren++;
}

public void setChild(boolean value)
{
    child = value;
}

public boolean getChild()
{
    return child;
}

public String toString()
{
    return (String)getData();
}

public BTNode(Object initialData, BTNode initialLeft, BTNode initialRight)
{
    data = initialData;
    left = initialLeft;
    right = initialRight;
}

/**
 * Accessor method to get the data from this node.
 * @param - none
 * @return
 * the data from this node
 **/
public Object getData()
{
    return data;
}

/**
 * Accessor method to get a reference to the left child of this node.
 * @param - none
 * @return
 * a reference to the left child of this node (or the null reference if
 * there
 * is no left child)
 **/
public BTNode getLeft()
{
return left;
}

/**
 * Accessor method to get the data from the leftmost node of the tree below
 * this node.
 * @param - none
 * @return
 * the data from the deepest node that can be reached from this node by
 * following left links.
 **/
public Object getLeftmostData( )
{
    if (left == null)
        return data;
    else
        return left.getLeftmostData( );
}

/**
 * Accessor method to get the data from the rightmost node of the tree below
 * this node.
 * @param - none
 * @return
 * the data from the deepest node that can be reached from this node by
 * following right links.
 **/
public Object getRightmostData( )
{
    if (right == null)
        return data;
    else
        return right.getRightmostData( );
}

/**
 * Accessor method to get a reference to the right child of this node.
 * @param - none
 * @return
 * a reference to the right child of this node (or the null reference if
 * there
 * is no right child)
 **/
public BTNode getRight( )
{
    return right;
}

/*
 * IOPair.java
 * A class representing an input or output of a process.
 * The class contains the name of the input or output
 * and a flag indicating whether the input or output is
 * negated or not.
 * Created on August 1, 2002, 12:05 AM
 */
* @author Sharad Bansal
* @version
*
public class IOPair extends Object
{
    public String url; //the name of the input or output
    public boolean matchFlag; //flag indicating whether the IO is negated

    /** Creates new IOPair */
    public IOPair() {
    }

    /**constructs a string representation of the object**/
    public String toString()
    {
        String flag;
        if (matchFlag)
            flag = "true";
        else flag = "false";
        return (url + ":" + flag);
    }
}

/**
 * @author Sharad Bansal
 * @version
 */
public class Pair extends Object {

    /** Creates new ClassName */
    public Pair() {
    }

    public int j;
    public int k;

    public Pair(int j, int k)
    {
        this.j = j;
        this.k = k;
    }
}

/**
 * @author Sharad Bansal
 * @version
 */
import java.io.*;
import java.util.*;
public class QueryParser extends Object {

    private String query;
    private String token;
    private String currentToken;
    public static String condition;
    private int index = 0;
    private ObjectStack inputStack, outputStack;
    private Set set;
    private IOPair pair;

/** Creates new ClassName */
public QueryParser()
{
    inputStack = new ObjectStack();
    outputStack = new ObjectStack();
}

private String scan()
{
    token = "";
    char nextChar = query.charAt(index);
    //consume blank spaces
    while (nextChar == ' ' || nextChar == '
' || nextChar == '')
    {
        index++;
        nextChar = query.charAt(index);
    }
    //get the next token
    while (nextChar != '(' && nextChar != ')' && nextChar != '' && nextChar != '
' && nextChar != '' && index < (query.length()-1))
    {
        //System.out.print(nextChar);
        token = token + nextChar;
        index++;
        nextChar = query.charAt(index);
    }
    //check for '(' and ')' 
    if (token.equals("") && (nextChar == ')' || nextChar == '('))
    {
        token = token + nextChar;
        index++;
    }
    //return token
    //System.out.println("next token is : "+token);
    return token;
}

//consumes a given token from the input stream 
private void accept(String str)
{
    if (currentToken.equals(str))
    currentToken = scan();
    else
    {
        System.out.println("Parse error :" + str + " expected");
        System.out.println("found : " + currentToken);
        System.exit(1);
    }
}

//consumes the next token unconditionally
private String acceptIt()
{
    String temp = currentToken;
    currentToken = scan();
    return temp;
}
//parses a query
private void parseQuery()
{
    accept("find");
    accept("service");
    accept("with");
    parseInputsClause();
    parseOutputsClause();
    parseCondition();
}

//parses an input clause
private void parseInputsClause()
{
    accept("inputs");
    accept("=");
    if (currentToken.equals("any"))
        acceptIt();
    else
        parseInputsList();
}

//parses an Input List
private void parseInputsList()
{
    set = new HashSet();
    if (currentToken.equals("="))
    {
        acceptIt();
        parseInputsProduct();
        accept("=");
    }
    else
    {
        parseInputName();
    }
    inputStack.push(set);
    if (currentToken.equals("or"))
    {
        acceptIt();
        parseInputsList();
    }
}

private void parseCondition()
{
    accept("condition");
    accept("=");
    condition = acceptIt();
}

public String getCondition()
{
    return condition;
}

//parses a conjunction of input terms
private void parseInputsProduct()
{
    parseInputName();
    if (currentToken.equals("and"))
private void parseInputName() {
    pair = new IOPair();
    if (currentToken.equals("not")) {
        acceptIt();
        pair.matchFlag = false;
    } else {
        pair.matchFlag = true;
    }
    pair.url = acceptIt();
    set.add(pair);
}

private void parseOutputsClause() {
    accept("outputs");
    accept("=");
    if (currentToken.equals("any")) {
        acceptIt();
        set = new HashSet();
        set.add("any");
        outputStack.push(set);
    } else {
        parseOutputsList();
    }
}

private void parseOutputsList() {
    set = new HashSet();
    if (currentToken.equals("(")) {
        acceptIt();
        parseOutputsProduct();
        accept(")");
    } else {
        parseOutputName();
    }
    outputStack.push(set);
    if (currentToken.equals("or")) {
        acceptIt();
        parseOutputsList();
    }
}

//parses a conjunction of Output terms
private void parseOutputsProduct()
{
    parseOutputName();
    if (currentToken.equals("and"))
    {
        acceptIt();
        parseOutputsProduct();
    }
}

//accepts an Output name
private void parseOutputName()
{
    pair = new IOPair();
    if (currentToken.equals("not"))
    {
        acceptIt();
        pair.matchFlag = false;
    }
    else
    {
        pair.matchFlag = true;
        pair.url = acceptIt();
        set.add(pair);
    }
}

public ObjectStack getInputStack()
{
    return inputStack;
}

public ObjectStack getOutputStack()
{
    return outputStack;
}

public void parse()
{
    StringBuffer buf = new StringBuffer();
    try
    {
        BufferedReader reader = new BufferedReader( new FileReader("DAMLQuery.txt"));
        int c = reader.read();
        while (c != -1)
        {
            buf.append((char) c);
            c = reader.read();
        }
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
    query = buf.toString() + '$';
    //System.out.println(query);
    currentToken = scan();
    parseQuery();
}
Appendix 3

DAML-S Advertisement

<?xml version="1.0" encoding="ISO-8859-1" ?>

<daml:Ontology rdf:about="">
</daml:Ontology>

<--
  DATA TYPES
  -->

<--
<daml:Class rdf:ID="CreditCardType">
  <daml:oneOf rdf:parseType="daml:collection">
    <CreditCardType rdf:Id="MasterCard"/>
    <CreditCardType rdf:Id="Visa"/>
    <CreditCardType rdf:Id="AmericanExpress"/>
    <CreditCardType rdf:Id="DiscoverCard"/>
  </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="ValidityType">
  <daml:oneOf rdf:parseType="daml:collection">
    <ValidityType rdf:Id="Valid"/>
    <ValidityType rdf:Id="Expired"/>
    <ValidityType rdf:Id="InvalidCCNumber"/>
    <ValidityType rdf:Id="InvalidCCType"/>
  </daml:oneOf>
</daml:Class>
<ValidityType rdf:Id="AuthorizationRefused"/>
</daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="DeliveryType">
    <daml:oneOf rdf:parseType="daml:collection">
        <DeliveryType rdf:Id="FedExOneDay"/>
        <DeliveryType rdf:Id="FedEx2-3Day"/>
        <DeliveryType rdf:Id="UPS"/>
        <DeliveryType rdf:Id="OrdinaryMail"/>
    </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="VendorName">
    <daml:oneOf rdf:parseType="daml:collection">
        <VendorType rdf:ID="Intel"/>
        <VendorType rdf:ID="Motorola"/>
        <VendorType rdf:ID="SeaGate"/>
        <VendorType rdf:ID="Samsung"/>
    </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="ComponentType">
    <daml:oneOf rdf:parseType="daml:collection">
        <ComponentType rdf:ID="Processor"/>
        <ComponentType rdf:ID="RAM"/>
        <ComponentType rdf:ID="HardDisk"/>
        <ComponentType rdf:ID="Monitor"/>
        <ComponentType rdf:ID="NetworkCard"/>
    </daml:oneOf>
</daml:Class>

<daml:Class rdf:ID="TransactionType">
    <daml:oneOf rdf:parseType="daml:collection">
        <TransactionType rdf:ID="Lease"/>
        <TransactionType rdf:ID="HirePurchase"/>
        <TransactionType rdf:ID="DirectSale"/>
    </daml:oneOf>
</daml:Class>

-->

<daml:Class rdf:ID="OnLinePCstore">
    <rdfs:subClassOf>
        <daml:Restriction daml:cardinality="1">
            <daml:onProperty rdf:resource="#Name"/>
        </daml:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <daml:Restriction daml:cardinality="1">
            <daml:onProperty rdf:resource="#Password"/>
        </daml:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <daml:Restriction daml:cardinality="1">
            <daml:onProperty rdf:resource="#CreditCardNumber"/>
        </daml:Restriction>
    </rdfs:subClassOf>
</daml:Class>
<process:composedOf rdf:resource="#SelectTransactionType1" />
    <rdfs:subClassOf>
        <daml:Restriction daml:cardinality="1">
            <daml:onProperty rdf:resource="#InputTransactionType" />
        </daml:Restriction>
    </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:ID="#SelectTransactionType1">
    <rdfs:subClassOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#Choice" />
</daml:Class>

<daml:Class rdf:about="#SelectTransactionType1">
    <process:components rdf:resource="#LeaseTransaction" />
    <process:components rdf:resource="#DirectSale" />
    <process:components rdf:resource="#HirePurchaseTransaction" />
</daml:Class>

当即 ShippingDiscount is a composite process consisting of an If-Then_Else control construct. If the total cost of the order exceeds a specified amount, shipping charges are waived, else a shipping charge is levied.
-->

<daml:Class rdf:ID="ShippingDiscount">
    <process:composedOf rdf:resource="#ShippingDiscount1" />
    <rdfs:subClassOf>
        <daml:Restriction daml:cardinality="1">
            <daml:onProperty rdf:Resource="#ShipIfCondition" />
        </daml:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <daml:Restriction daml:cardinality="1">
            <daml:onProperty rdf:Resource="#ShipThenProcess" />
        </daml:Restriction>
    </rdfs:subClassOf>
</daml:Class>
<daml:Restriction daml:cardinality="1">
  <daml:onProperty rdf:resource="#NewAccountEffect"/>
</daml:Restriction>
</rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:ID="LoadAccount">
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#Name"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#Password"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#AccountExistsPreCondition"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:ID="SelectVendor">
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#InputVendorName"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#InputComponentType"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:ID="AddComponent">
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#InputVendorName"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#InputComponentType"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <daml:Restriction daml:cardinality="1">
  </daml:Restriction>
</daml:Class>
<daml:Class rdf:ID="LeaseTransaction">
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#LeaseRental="/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:ID="HirePurchase">
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#InstalmentAmount="/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:ID="DirectSale">
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#SalePrice="/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:ID="SelectDeliveryType">
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#DeliveryTypeInput="/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:ID="WaiveShippingCharges">
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#WaiveShippingPreCondition="/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
<daml:Class rdf:ID="LevyShippingCharges">
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#LevyShippingPreCondition"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#ShippingCharges"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:ID="FinalizeBuy">
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#E-Receipt"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<!-- Inputs, Outputs, Preconditions and Effects -->
<!--

<rdf:Property rdf:ID="Name">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#input"/>
  <rdfs:domain rdf:resource="#LoadAccount"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#String"/>
</rdf:Property>

<rdf:Property rdf:ID="Password">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#input"/>
  <rdfs:domain rdf:resource="#LoadAccount"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#String"/>
</rdf:Property>

<rdf:Property rdf:ID="AccountNumber">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#output"/>
  <rdfs:domain rdf:resource="#LoadAccount"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#Integer"/>
</rdf:Property>

75
<rdf:Property rdf:ID="LoginID">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#input" />
  <rdfs:domain rdf:resource="#CreateNewAccount" />
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#Integer" />
</rdf:Property>

<rdf:Property rdf:about="Password">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#input" />
  <rdfs:domain rdf:resource="#CreateNewAccount" />
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#Integer" />
</rdf:Property>

<rdf:Property rdf:ID="CreditCardNumber">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#input" />
  <rdfs:domain rdf:resource="#CreateNewAccount" />
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#Integer" />
</rdf:Property>

<rdf:Property rdf:ID="CreditCardType">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#input" />
  <rdfs:domain rdf:resource="#CreateNewAccount" />
  <rdfs:range rdf:resource="#CreditCardType" />
</rdf:Property>

<rdf:Property rdf:ID="AccountNumber">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#output" />
  <rdfs:domain rdf:resource="#CreateNewAccount" />
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#Integer" />
</rdf:Property>

<rdf:Property rdf:ID="AccountDoesNotExistPrecondition">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#preCondition" />
  <rdfs:domain rdf:resource="#CreateNewAccount" />
  <rdfs:range rdf:resource="true" />
</rdf:Property>

<rdf:Property rdf:ID="ValidCreditCardPrecondition">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#preCondition" />
  <rdfs:domain rdf:resource="#CreateNewAccount" />
  <rdfs:range rdf:resource="true" />
</rdf:Property>

<rdf:Property rdf:ID="NewAccountEffect">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#effect" />
  <rdfs:domain rdf:resource="#CreateNewAccount" />
  <rdfs:range rdf:resource="true" />
</rdf:Property>
<rdf:Property rdf:ID="AccountExistsPrecondition">
    <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#precondition" />
    <rdfs:domain rdf:resource="#LoadAccount" />
    <rdfs:range rdf:resource="true" />
</rdf:Property>

<rdf:Property rdf:ID="InputVendorName">
    <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#input" />
    <rdfs:domain rdf:resource="#SelectVendor" />
    <rdfs:range rdf:resource="#VendorName" />
</rdf:Property>

<rdf:Property rdf:ID="InputComponentType">
    <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#input" />
    <rdfs:domain rdf:resource="#AddComponent" />
    <rdfs:range rdf:resource="#ComponentType" />
</rdf:Property>

<rdf:Property rdf:ID="ComponentAddedEffect">
    <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#effect" />
    <rdfs:domain rdf:resource="#AddComponent" />
    <rdfs:range rdf:resource="true" />
</rdf:Property>

<rdf:Property rdf:ID="PCConfiguration">
    <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#output" />
    <rdfs:domain rdf:resource="#AddComponent" />
    <rdfs:range rdf:resource="#String" />
</rdf:Property>

<rdf:Property rdf:ID="InputTransactionType">
    <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#input" />
    <rdfs:domain rdf:resource="#SelectTransactionType" />
    <rdfs:range rdf:resource="#string" />
</rdf:Property>

<rdf:Property rdf:ID="LeasePeriod">
    <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#input" />
    <rdfs:domain rdf:resource="#LeaseTransaction" />
    <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#Integer" />
</rdf:Property>
<rdf:Property rdf:ID="LeaseRental">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#output" />
  <rdfs:domain rdf:resource="#LeaseTransaction" />
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#Integer" />
</rdf:Property>

<rdf:Property rdf:ID="HPPeriod">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#input" />
  <rdfs:domain rdf:resource="#HirePurchaseTransaction" />
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#Integer" />
</rdf:Property>

<rdf:Property rdf:ID="InstalmentAmount">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#output" />
  <rdfs:domain rdf:resource="#HirePurchaseTransaction" />
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#Integer" />
</rdf:Property>

<rdf:Property rdf:ID="Warranty">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#output" />
  <rdfs:domain rdf:resource="#HirePurchaseTransaction" />
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#Integer" />
</rdf:Property>

<rdf:Property rdf:ID="SalePrice">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#output" />
  <rdfs:domain rdf:resource="#DirectSale" />
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#Integer" />
</rdf:Property>

<rdf:Property rdf:ID="DeliveryType">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#output" />
  <rdfs:domain rdf:resource="#SelectDeliveryType" />
  <rdfs:range rdf:resource="#DeliveryTYpe" />
</rdf:Property>

<rdf:Property rdf:ID="WaiveShippingPrecondition">
  <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#precondition" />
</rdf:Property>
<rdf:RDF>

  <rdf:Property rdf:ID="LevyShippingPrecondition">
    <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#precondition"/>
    <rdfs:domain rdf:resource="#LevyShippingCharges"/>
    <rdfs:range rdf:resource="true"/>
  </rdf:Property>

  <rdf:Property rdf:ID="E-Receipt">
    <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#output"/>
    <rdfs:domain rdf:resource="#FinalizeBuy"/>
    <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#String"/>
  </rdf:Property>

  <rdf:Property rdf:ID="ShippingCharges">
    <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#output"/>
    <rdfs:domain rdf:resource="#LevyShippingCharges"/>
    <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#Integer"/>
  </rdf:Property>

  <rdf:Property rdf:ID="NilShippingCharges">
    <rdfs:subPropertyOf rdf:resource="http://www.daml.org/services/daml-s/2001/10/Process.daml#output"/>
    <rdfs:domain rdf:resource="#WaiveShippingCharges"/>
    <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#Integer"/>
  </rdf:Property>

</rdf:RDF>