DEVELOPING A SEMANTIC WEB MATCHMAKER FOR MULTIMEDIA LEARNING OBJECTS

A Case Study on Creating a Web Service Interface for OO-JDrew

Jinan Fiaidhi, Sabah Mohammed and Marshall Hahn
Department of Computer Science, Lakehead University, 955 Oliver Road, Thunder Bay, Ontario P7B 5E1, Canada

Keywords: Semantic Web, Matchmaking, Learning Objects, e-Learning.

Abstract: This paper introduces a matchmaker to discover multimedia learning objects on the semantic Web. The increase in learning objects and lack of semantic base in search mechanisms of the semantic Web discovery engine (UDDI) make it difficult for clients to find a required learning object service (LOs). Using metadata to find distributed educational Web services or LOs that meet one's functional requirements is only the first step. LOs requestors may have additional requirements such as presenting a flexible query to find a relevant multimedia course. The LOs need to employ a matchmaking engine that can process the rules and conditions of the requesters. This article provides a draft implementation of an Apache Axis matchmaker, which embeds the OO-jDREW rule-based engine. The developed first prototype provides a lightweight and customizable reasoner for allocating multimedia LOs courseware presentations.

1 INTRODUCTION

Key to the success of effectively retrieving relevant services in the future semantic Web is how well intelligent service mediators may perform semantic matching in a way that goes far beyond of what standard service discovery protocols such as UDDI (http://www.uddi.org/). Existing resource description and resource selection on the Semantic Web is highly complicated. Traditional resource matching, as exemplified by the IMPACT (Smith, 1998), InfoSlenth (Klusch, Fris and Sycara, 2006) and Retsina/Larks (Klusch and Sycara, 2006) is done based on symmetric, attribute-based matching. A matchmaker is a computational software entity that has access to one or multiple, heterogeneous, and distributed data and information sources; proactively searches for, mediates, and maintains relevant information on behalf of its human users or other agents, preferably just-in-time. In other words, it is managing the matching of incoming requests with advertised services. Most of the available matchmakers are based on the OWL ontologies and utilize complicated tools and APIs (Sycara et al, 2001).

The OWL tools and APIs utilize certain type of reasoners (e.g. Jena, F-OWL, CoBrA, Protégé-OWL API, Racer, Pellet, FaCT). These reasoners are not quite flexible as well as they have performance issues (e.g. inconsistencies, misclassifications, irrelevant query responses)(Liebig et al, 2005). Obviously we are lacking a lightweight and flexible reasoning which does not need to be very powerful, but it should be highly customizable. Actually even for a complex ontology, the queries processed may be very simple and can be processed by simpler reasoners. Then choosing a less powerful, but more efficient, reasoner provides better performance. Ontologies can be considered as playing a key part in the Semantic Web since they provide the vocabulary needed for semantic mark-up. But rules are also required for the Web, and most people now agree that a Web rule language is needed. According to the Semantic Web stack, rules are on the top of ontologies. But in many cases, ontologies alone are not enough. Using rules *in conjunction* with ontologies is a major challenge for the Semantic Web. SeetRules, Flora2, OOJdrew, SWRL, Hoolet, Jena2, and ROWL are examples of some notable resoners.

Since our problem combines ontological inference and matchmaking, we believe that flexible reasoning methods are necessary. This project focuses on two areas; 1) embedding OO-jDREW into an application, and 2) rules and rule-processing in the LOs domain. The primary focus of the project is the integration of OO-jDREW into a semantic Web application that hides the complexities of OO-

Fiaidhi J., Mohammed S. and Hahn M. (2007).
DEVELOPING A SEMANTIC WEB MATCHMAKER FOR MULTIMEDIA LEARNING OBJECTS - A Case Study on Creating a Web Service Interface for OO-JDrew.
In *Proceedings of the Second International Conference on e-Business*, pages 287-292
DOI: 10.5220/0002106302870292

jDREW. The inclusion of OO-jDREW is a step towards defining a shared Rule Markup Language (RuleML), permitting both forward (bottom-up) and backward (top-down) rules in XML for deduction, rewriting, and further inferential-matchmaking tasks.

2 SVG LEARNING OBJECTS

In our earlier work we developed a Java utility called the Learning Object Presentation (LOP) Generator, which generates an SVG slideshow presentation based upon an xml description. This is the type of LO we will focus on in this paper. The input file format of this utility is shown in Figure 1. All data describing the presentation is contained within the <ss:presentation> tag. In this direction, the CanCore metadata describing the presentation is placed within the <ss:cancore> tag. Global properties such as the transition type (shift, fade or none) can placed under the <ss:properties> tag. Each slide is described within an <ss:slide> tag. The delay attribute specifies how long the slide should be displayed if the slideshow is in "play mode". Each slide must have a titlebox and may have one bodybox and\or one image.

```
<ss:presentation
xmlns:ss='urn:SLIDESHOW:0-395-36341-6'>
  <ss:cancore>...
  </ss:cancore>
  <ss:properties>
    <ss:transition type="fade"
      duration="1000" frames="20"/>
  </ss:properties>
  <ss:slide delay="">
    <ss:titlebox>
      <ss:title>The Title</ss:title>
      <ss:subtitle>Slide1</ss:subtitle>
    </ss:titlebox>
    <ss:bodybox>
      <ss:point>
        <ss:text>The Text<ss:text>
        <ss:point>...<ss:point>  ...
      </ss:point>...
    </ss:bodybox>
    <ss:images>
<ss:image path="p.jpg" x="470" y="160"
       width="500" height="550" />
    </ss:images>
  </ss:slide>
  <ss:slide delay="">...
  </ss:slide> ...
</ss:presentation>
```

Figure 1: The XML LOP Input File.

3 OO-JDREW AND RULEML

The OO-jDREW reasoning engine contains two modes: a Bottom-Up (forward chaining of rules) version, and a goal driven top-down (backward chaining of rules) version which works in a fashion similar to most Prolog systems. The task of developing reference implementations for RuleML, which is an evolving standard, is made easier by using a tool-box approach, which is one of the design issues for jDREW: the flexibility to quickly cope with changes to the syntax and required operations to implement the various semantics. There are utilities in the jDREW toolbox for various tasks: reading files of RuleML statements into the internal clause data structure, storing and manipulating clauses, unification of clauses according to the positions of the selected literals, a basic resolution engine, clause to clause subsumption and clause to clause list subsumption, choice point managers, priority queues for various reasoning tasks, and readable top-level procedures (Spencer, 2002). An ontological graph is used as a knowledge representation of the ontology (Biletskiy, 2006).The ontological graph is implemented using Rule Markup Language (RuleML). The use of RuleML allows the ontology to flexible, extensible and platform-independent. So, the ontology can be easily integrated with other ontologies (Boley, 2003).

4 THE MATCHMAKER

The Multimedia Learning Object Matchmaker (MLOM) architecture is shown in Figure 1. The MLOM allows one to find all MLOs meeting particular criteria. Currently, SVG slideshow presentations are the only type of MLO supported by the system. All SVGs are distributed across a number SVG Web Services. A set of facts describing each MLO must be stored within MLOM's RuleML knowledge base. MLOM's remotely callable interface provides a method through which these facts can be registered. Its interface also provides a method through which a Client can send a query in POSL format.

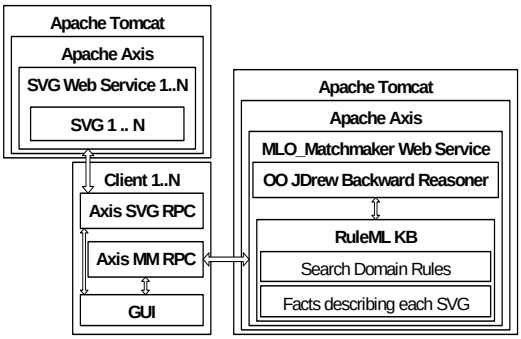

Figure 2: The MLO_Matchmaker Architecture.

Using such a query as the goal, OO JDrew will determine which set of MLOs are described by the facts necessary to achieve the goal. The MLOs set will be returned to the Client after which the user may choose to view any number of them. All web services utilize the Apache Axis framework which runs within an Apache Tomcat web server. Axis allows one to deploy a Java class as a web service. Figure 2 illustrates the class used to create the MLO_Matchmaker Web Service as well as some of the components it depends on.

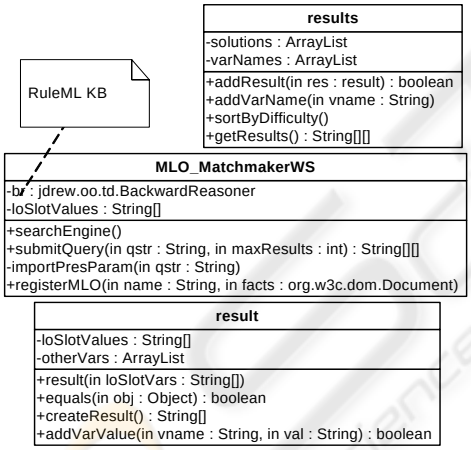

Figure 3: Matchmaker UML Class Diagram.

4.1 The MLO_Matchmaker Knowledge-Base

For MLO_Matchmaker to be capable of performing an intelligent MLO search, a domain-specific knowledge base must be provided. A portion of a knowledge base representing Java and C++ is shown in Figure 3. So suppose a presentation had the fact "for" specified within its metadata. If the goal controlStructures were sent to MLO_Matchmaker, OO-jDREW would determine that it is possible to infer this goal. The MLO_Matchmaker would then return the SVG's name, description and other information to the Client. But how will OO-jDREW know which facts in the knowledge base belong to which SVG? Our approach would be best explained with a simple example. We will use the POSL short-form syntax for ruleML to save space and for extra clarity.

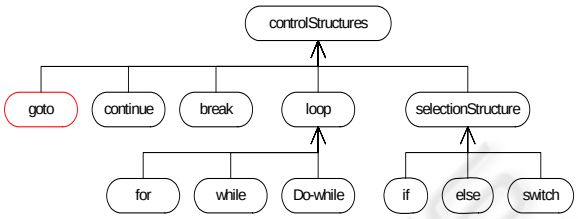

Figure 4: All rules except goto -> controlStructures apply to both C++ and Java.

Suppose an SVG presentation named "Java Control Structures" with name and URL are "javacs.svg"& "localhost:9080/axis/SVGserver.jws" respectively. Also assume this presentation describes only loops. The following set of facts could be used to represent this presentation:

LearningObject(
name->"Java Control Structures";
desc->"loops in Java";
url->"http://localhost:9080/axis/SVGserver.jws";
file->"javacs.svg";
difficulty->"6.5"
pres->"UUID-550e8400-e29b-41d4-a716-446655440000").
for(pres->"UUID-550e8400-e29b-41d4-a716-446655440000").
while(pres->"UUID-550e8400-e29b-41d4-a716446655440000").
do-while(pres->"UUID-550e8400-e29b-41d4-a716 446655440000").

The url and file slots store the information a Client needs to retrieve the LO. The difficulty slot is used to store a number in the range 0.0 – 10.0 meant to indicate how difficult the presentation is. We use the pres slot to link all facts related to a presentation together. OO-jDREW allows a number of goals to be anded together. Therefore, the following query can be used to retrive a presentation's contact information:

for(pres->?PRES),while(pres->?PRES),LearningObject(name->?NAME;desc-?DESC; url->?URL; file->?FILE;
difficulty->?DIFF;pres->?PRES).

The variable bindings resulting from this query are the following:

NAME = "Java Control Structures"
DESC = "loops in Java"
URL = "http://localhost:9080/axis/SVGserver.jws"
FILE = "javacs.svg"
DIFFICULTY = "6.5"

PRES = ”UUID-550e8400-e29b-41d4-a716 446655440000”

The presentation can now be retrieved by the Client based upon the URL and filename.As another example, suppose the query was the following:

controlStructure(pres->?PRES),LearningObject(name->?NAME; desc-?DESC; url->?URL; file->?FILE; difficulity->?DIFF; pres->?PRES).

This goal can be inferred given a LearningObject fact and the following rules:

controlStructure(pres->?PRES) :- loop(pres->?PRES). loop(pres->?PRES) :- for(pres->?PRES).

4.2 The Matchmaker Class

As previously mentioned, the MLO_Matchmaker class is designed to be deployed as an Apache Axis Web Service. When Axis creates an object of this class, its constructor will load the rdfs types file found in the TOMCAT_PATH\bin folder into the OO-jDREW type system. It will also load all ruleML documents found at TOMCAT_PATH\bin\metadata into the jdrew.oo.td.BackwardReasoner instance member. The facts related to each registered SVG are placed into a separate file so that they can be updated more easily.

MLOM’s registerLO method takes as its first argument the name of an SVG and as its second argument a ruleML document describing the SVG. It will create a copy of the document in the TOMCAT_PATH\bin\metadata directory.

MLOM’s submitQuery method takes a POSL query string as its first argument and an integer specifying the maximum number of results to return as its second argument. Its return type is an array of String arrays. The first String array stores the names of the variables found in the query. Each subsequent String array stores a query solution(see Table 1).

Table 1**:** The Return Query Format.

Name	Desc.	url	filename	Diff.	Other 0...N
Solutions one’s bindings for each variable					
Solutions two’s bindings for each variable					
…					
Solutions M’s bindings for each variable					

Name, description, url, filename, and difficulty are always returned as the first five columns whether they were specified in the query or not. This is done to ensure (1) an entry for each learning object found is always listed and (2) that the information needed to retrive the LO associated with a result is always present. The following steps are carried to generate a query’s results:

STEP 1: Query preprocessing.

The query is preprocessed via a call to method importPresParam, which will insert things into the query string that are considered implicit. Recall the following query:

for(pres->?PRES),while(pres->?PRES),LearningObject(name->?NAME;desc-?DESC; url->?URL; file->?FILE; difficulity->?DIFF;pres->?PRES).

The user doesn’t have to actually enter all that to perform a search. The user only has to enter the following:

for(),while().

The above example illustrates that after preprocessing, (1) pres->?PRES is inserted into all slotted predicates to ensure that a solution’s variable bindings will all be related to the same LO and (2) the values of all LearningObject slots will be retrieved by the query. Regarding the latter, the user may override the default variable names used: ?NAME, ?DESC, and so on. In any case, the variable names used are saved into MLO_Matchmaker’s loSlotValues instance member.

STEP 2: Parse the query into a DefinateClause instance. The following code will perform this task:

```
POSLParser pp = new POSLParser();
DefiniteClause dc = pp.parseQueryString(qstr);
```

STEP 3: Generate and return the query results.

Each solution’s variable bindings are stored in a result instance. A call to addVarValue will add a variable’s binding to the result. Based on information contained MLOM’s loSlotValues instance member obtained via result’s constructor, the method can determine whether or not the variable is used within an LO predicate slot. If yes, the method returns true. All results are stored in results a instance. These objects have special names for each LO predicate slot hard-coded within them such as Name, Description, and so on. Any other variable names must be added manually via a call to addVarName. A call to addResult is used to add a result to the results list. The method returns true if the result is not duplicate of some result already in the list. Duplicates arise when there is more than one way to infer an LO meets the criteria of the search. Since the Client is not aware of logic OO-jDREW used to arrive at its decisions, duplicate results are suppressed.

4.3 Sharing Rules

OO-jDREW allows the type of a variable to be specified. Two variables will only unify if they have the same type or if one of the types is a subclass of

the other type, assuming that the other conditions needed for unification have been met. We need not rely on OO JDrew's built-in types alone. User defined types can be created. These types must be loaded from an rdfs document. We utilize this mechanism to share rules between languages. The idea is to assign one of the following types to the variable used in the pres slot of all rules and facts: Language, Java or Cpp. Our input rdfs document is shown below:

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdfs:Class rdf:ID="Language"/>
<rdfs:Class rdf:ID="Java"> <rdfs:subClassOf rdf:resource="Language"/> </rdfs:Class>
<rdfs:Class rdf:ID="Cpp"> <rdfs:subClassOf rdf:resource="Language"/> </rdfs:Class>
</rdf:RDF>

Suppose that two SVG presentations have been registered and that one deals with Cpp and one with Java. Assume that *while loops do not exist in Java* and that the Java presentation's metadata mistakenly states that the presentation covers Java while loops. If we ignore the LearningObject facts for simplicity, a possible knowledge base would be the following:

while(pres->"UUID-550e8400-e29b-41d4-a716-446655440000":Cpp).
while(pres->"UUID-234e4354-fr43-4334-a755-126655333000":Java).
controlStructure(pres->?PRES:Language) :- loop(pres->?PRES:Language).
loop(pres->?PRES:Language) :- while(pres->?PRES:Cpp).

The query controlStructure(pres->?PRES:Cpp) would succeed. However the query controlStructure(pres->?PRES:Java) would fail, since the PRES variable in the predicate while(pres->?PRES:Cpp) cannot unify with while(pres->"UUID-234e4354-fr43-4334-a755-126655333000":Java) because the types do not match. As a result, the rule loop(pres->?PRES:Language) :- while(pres->?PRES:Cpp) is properly limited to the C++ language only.

4.4 LO SVG Web Service

Class SVGWebService is designed to be deployed as an Apache Axis web service and is shown in Figure 5. Its only method will return the SVG document indicated by the argument. The method expects to find all available SVGs in the TOMCAT_PATH\bin\svg folder.

SVGWebService
-doc : org.w3c.dom.Document
+getSVG(String filename)() : org.w3c.dom.Document

Figure 5**:** The LO SVG Web Service.

4.5 The Client

A screen shot of the Client's GUI is shown in Figure 4. Queries can be entered in the query textbox. The execute button can be pressed to execute the query. If the query is successful, its results will be listed below the query textbox. Any SVG in the results list can be retrieved and viewed by double clicking that results entry in the list.

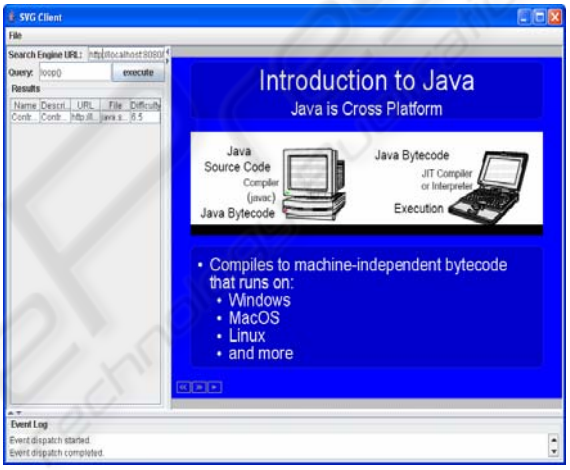

Figure 6**:** The Matchmaker GUI.

All code needed to communicate with an axis web service has been placed into a package called axisRPC. The content of this package is shown in Figure 6. The axisRPC class contains a collection of objects (some from the Axis libraries) needed to carry out a remote procedure call to a RPC style web service. Its constructor takes three arguments: the url of the web service, the service name and the port name. The service name and port name can be determined by an examination of the WSDL document describing the web service you wish to call. An instance of axisMM can be used to submit a query to the MLO_Matchmaker web service. Its constructor takes the url of the web service and will initialize the axisRPC object it inherits from with the appropriate service name and port name. An instance of axisSVG can be used to retrieve an SVG from an SVG web service. Its getSVG method will call the getSVG method of the web service indicated by the url passed to its constructor.

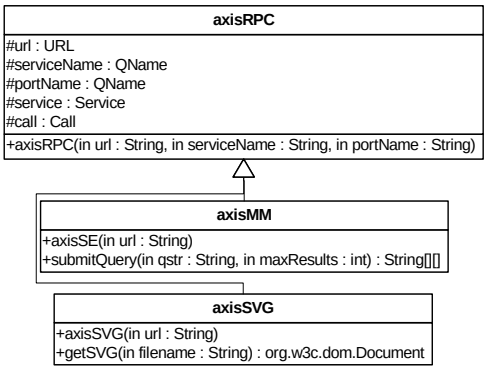

Figure 7: The Axis Mediator Part of the Client.

4 EXAMPLES OF QUERIES

Suppose that three LOs have been registered and that they are described by the following metadata (omitting the desc, url and file slots):

for(pres->"UUID-550e…")
while(pres->"UUID-550e…")
doWhile (pres->"UUID-550e…")
LearningObject(name->"Repetition in Java"; difficulty->"6.5"; pres->"UUID-550e…":Java)
selectionStructure (pres->"UUID-770e…")
LearningObject(name->"Selection in Java"; difficulty->"5.5"; pres->"UUID-770e…":Java)
for(pres->"UUID-660e…")
while(pres->"UUID-660e…")
doWhile (pres->"UUID-660e…")
LearningObject(name->"Repetition in C++"; difficulty->"7.5"; pres->"UUID-660e…":Cpp)

The following query would return all LOs:

controlStructures()

The following query would return the LO related to C++: **LearningObject(pres->?PRES:Cpp)**

The following query would return the LO related to Java that deals with for loops:

for(), LearningObject(pres->?PRES:Java)

The following query would return the two LO's dealing with loops: **loop()**

The following query would return nothing since there are no presentations dealing with both selection structures and loops:

selectionStructure(),loop(), LearningObject(pres->?PRES:Language)

The query *contains(?N, "in"), LearningObject(name->?N)* would fail because OO-jDREW's contains built-in rejects variable parameters. Likewise, the query *greaterThan(?DIFF, "5.0"), LearningObject(difficulty->?DIFF)* would also fail for the same reason. OO-jDREW's extensible built-in architecture makes it possible to define new built-ins that would make the above two queries possible.

5 CONCLUSIONS

Using rules in conjunction with ontologies is a major challenge for the Semantic Web. This paper propose an approach for reasoning with RuleML/ POSL rules and ontologies expressed by CanCore metadata. The reasoning is focused on matchmaking between learners and learning objects. Apache Axis has been used as a communication mediator. The learning objects are considered to be web services representing multimedia SVG presentations.

REFERENCES

Biletskiy Y., Boley H., Zhu L. 2006. A RuleML-Based Ontology for Interoperation between Learning Objects and Learners. *UCFV Research Review*, Issue 1. Available: http://journals.ucfv.ca/ojs/rr/

Boley, H., 2003. Object-Oriented RuleML: User-Level Roles, URI Grounded Clauses, and Order-Sorted Terms, , In Proc. Rules and Rule Markup Languages for the Semantic Web (RuleML). Sanibel Island, Florida, LNCS 2876, Springer-Verlag.

Klusch, M, Sycara, K. 2001. Brokering and Matchmaking for Coordination of Agent Societies: A Survey. In Coordination of Internet Agents, A. Omicini et al. (eds.), ISBN 3-540-41613-7, Springer

Klusch, M., Fries, B., Sycara, K., 2006. Automated Semantic Web Service, *Discovery with OWLS-MX. Proceedings of 5th International Conference on Autonomous Agents and Multi-Agent Systems* (AAMAS), Hakodate, Japan, ACM Press.

Klusch, M, Sycara, K. 2000. Brokering and Matchmaking for Coordination of Agent Societies: A Survey. In Coordination of Internet Agents, Springer.

Liebig T., Luther M., Noppens O., Paolucci M., Matthias W., Henke F., 2005. Building Applications and Tools for OWL Experiences and Suggestions, Workshop on OWL Experiences and Directions 2005, Ireland.

Sycara, K., Widoff, S., Klusch, M., Lu, J. 2001. LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. Journal on Autonomous Agents and Multi-Agent Systems, Kluwer Academic, vol. 4 (4).

Spencer B., 2002. The Design of j-DREW: A Deductive Reasoning Engine for the Web, , In Proceedings of the First CologNET Workshop on Component-Based Software Development and Implementation Technology for Computational Logic Systems. CBD ITCLS, Madrid. Sep 20, pp. 155-166.