Developing a Brokering Architecture for Multimedia Learning Objects on the Semantic Web

Jinan Fiaidhi, Sabah Mohammed and Marshall Hahn,

Department of Computer Science, Lakehead University,
Thunder Bay, Ontario P7B 5E1, CANADA

Summary
The World Wide Web is changing. While once conceived of and implemented as a collection of static pages for browsing, it now promises to become a web of services--a dynamic aggregate of interactive, automated, and intelligent services that interoperate via the Internet. For the web of services model to succeed, techniques which match service requestors with service providers must be developed. In particular, such techniques are especially important for highly demanding applications such as e-Learning. Such educational applications developed for the Semantic Web require some kind of reasoning capability. Providing sound and complete reasoning services is essential for many of these applications to function properly. In this paper, we present a Multimedia Learning Object Brokering Architecture implemented using Apache Axis, Jena and Pellet. The Broker component determines which MLOs satisfy a query based upon information contained in one or more domain-specific ontologies. The system's requestor and provider components are designed specifically for use with SVG slideshow presentations described using the CanCore standard. All queries are expressed in OWL-OQL, a concise OWL query language created for use with our brokering system.

Key words:
Multimedia Learning Objects, Semantic Web, Semantic Broker.

1. Introduction

While eLearning systems become more and more popular in daily education, available applications lack opportunities to structure, annotate and manage their contents, as well as their interactive components in a high-level fashion. General efforts to improve these deficits are taken by initiatives to define rich meta data sets and a semantic web layer so enable users to access the heterogeneous educational services. Although the accessibility to various learning repositories is still manual and time-consuming, the realization of the Semantic Web provides several new AI-inspired content markup languages (e.g. OIL, DAML+OIL, DAML-L, RDF, OWL and OWL-S)[1] that enable the manipulation of complex taxonomic and logical relations between entities on the Web. Such languages utilize ontologies to provide basis for the interoperability among different distributed entities. The ontologies offer a way to apply structured and well-defined meanings. In order to achieve the vision of semantic web (based on such ontologies) and the various educational services, we need to integrate certain technologies like matchmaking or brokering. The matchmaker differs from the broker primarily in that the matchmaker actually passes responsibility for service provision directly to the matched server and requesting client, whereas the broker completely hides the identity of the server from the client and vice versa [2].

On one hand, matchmakers increasingly appear in standard for Web services infrastructures such as UDDI. The Matchmaker is a web service that helps make connections between service requesters and service providers. The Matchmaker serves as a "yellow pages" of service capabilities. The Matchmaker may employ techniques from information retrieval, AI, and software engineering to compute the syntactical and semantic similarity among service capability descriptions. The matching engine may contain several filters for namespace comparison, word frequency comparison, ontology similarity matching, ontology subsumption matching, and constraint matching. The user may configure these filters to achieve the desired tradeoff between performance and matching quality. Most of the available matchmaking engines adopt OWL-S and implement a version of the matching algorithm/filters which can be used to provide capability matching to the UDDI registry. Thus, the Matchmaker also serves as a liaison between a service requester and a service provider. However, the main limitation of OWL based matchmakers is their lack of a definition of rules and an associated reasoner. Therefore, some researchers tried to couple OWL-S with RuleML (e.g SWRL [3]) and other reasoning policies. Such approaches may end in non-standard systems that make the matchmaker more complex to be used in practice [4,5].

On the other hand, brokers play key roles in many Semantic web applications. Thus in general, brokers facilitate the interaction between two or more parties. For example, if two parties want to communicate, but they do not share a common language, brokers may provide translation services, or if the two parties do not trust each other, a broker may provide a trusted intermediary. Brokers are widely used in distributed information systems. Yet, there has not been a detailed analysis of a Broker's architecture and no general solution has been proposed on how the Broker's tasks should be accomplished. Brokers need more integral framework to represent and use ontologies. Thus researchers like [6,7,8,9] recommended the use of a more comprehensive approach based on a new standard--Web Service Modeling Language (WSMO), which specifies the key elements for describing Semantic Web Services and their usage. Such elements include Ontologies, Goals, Web Services and Mediators. Ontologies are the key to describe all of these elements through concepts, relations, functions, instances and axioms. Goals specify what the user wants from a particular web service. Web Service descriptions specify what a service can provide in terms of a capability and any number of interfaces. Mediators are the coordinating engines/policies of the brokers and help to link heterogeneous elements between each other in order to enable heterogeneous components to interoperate.

Moreover, the need for content-based retrieval from visual media, such as multimedia learning objects, is ever increasing rapidly in many educational applications. The research in this article aims at the development of an architecture for an innovative brokerage system in the e-learning domain that makes use of the emerging Semantic Web to provide high-level services for people looking for appropriate online courses. The brokering architecture aims to describe how Web Service and Semantic Web technologies can be utilized to support visual media or course presentation retrieval. The brokering architecture explains how metadata like CanCore and ontology can be utilized to realize more intelligent content-based retrieval on visual media data.

2. Multimedia Learning Objects

In [23], we developed a Java utility called the Learning Object Presentation (LOP) Generator, which generates an SVG slideshow presentation based upon an xml description (i.e. CanCore Metadata). This is the type of LO we will focus on in this paper. The input file format of this utility is shown below. 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>Slide 1</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>
```

3. Related Research Work

The whole research work on the Semantic web does not seem to have moved significantly forward because it is only at the beginning. There are still quite a number of open issues to be resolved and several experiments/attempts need to be analyzed. At the same time, there are an increasingly large number of learning objects resulting from the growing interest in e-learning and there is a large demand for a variety of agents and brokers that can interact with these learning objects to provide services for learners. Such agents and brokers

need to ensure that these services will retrieve relevant resources. The Semantic web is an important initiative to provide such agents and brokers with the mechanisms and standards to achieve their goals. Table 1 list some of the earlier attempts that use variety of theoretical or non-standard techniques (e.g. inference Rules, Collaborative Repositories, middleware and Portals)

SPEL	**Portal**	**[10]**
Diogene	**Inference Rules**	**[11]**
Universal	**Collaborative Repositories**	**[12]**
LOMster	**LOM middleware**	**[13]**

Table 1: Notable Non-Standard Brokers.

However, the major outcome of the research being undertaken over the last few years in the area of artificial intelligence for the Semantic Web is the benefit of using ontologies for content-based information retrieval. The integration of broker technology and ontologies could significantly affect the use of Web services and the ability to extend programs to perform tasks for users more efficiently and with less human intervention. Such approaches have been applied specifically to learning objects[14,15] where they were used to develop a learning-ontology-broker able to offer different learning ontologies corresponding to various educational contexts that may be of user interest. As a result, the learning materials to be taken into account are the ones that belong to the educational context relevant for the user. Such learning objects are most likely the ones that best fit the user needs.

More general approaches that were applied to other areas rely more on intelligent reasoning services to access and integrate information sources (e.g. Ontobroker, SHOE, OntoSeek, BUSTER, IM, SIMS, OBSERVER, COIN; and Swoogle)[16,17]. The majority of these systems provide representation mechanisms based on description logic for ontology-based content explication. The most sophisticated system among the above mentioned systems is BUSTER. The BUSTER[18] client provides an ontology-driven user interface to specify queries and to present the results of the retrieval. However, the communication between the clients and the Web service clusters is implemented via Remote Method Invocation (RMI) and not upon standard protocols (e.g. SOAP).

4. Developing a Broker with a Standard Reasoner

Web services interact with each other by exchanging SOAP messages. Such services are described using machine-readable metadata. The metadata describes the capabilities and requirements of a service — often called the service policy. If such policies can be mapped into a standardized logic, then we can benefit from using standard reasoners that accept such logic. The W3C-endorsed Web Ontology Language (OWL) is based upon description logic and is ideal for the goals of our Broker. In particular, the OWL-DL subset of OWL Full is most appropriate since it is known to be sound and complete. Many OWL reasoners are available such as Pellet, Racer [22], FaCT [21] and FaCT++ [20]. However, Pellet is the only sound and complete OWL-DL reasoner. Moreover, it has extensive support for reasoning with individuals (including nominal support and conjunctive query), user-defined datatypes, and debugging support for ontologies [19]. Rather then developing their own API, the developers of Pellet choose to support a number of standard APIs such Jena, OWL API and the DIG interface. We chose to access Pellet using Jena since it is one of the most widely used Java APIs for RDF and OWL, providing further services for model representation, parsing, database persistence and some visualization tools.

5. The Brokering Architecture

Our system's design requirements are:
1. **Centralized store**. Metadata from a variety of existing multimedia learning objects will be integrated and stored in unified way.
2. **Flexible data model**. The repository should have the ability to store all existing content items and also be extensible in the future as new requirements arise.
3. **Scalable.** The repository must support our rather large multimedia learning objects.
4. **Query Performance**. The repository will be the backbone of the Broker.
5. **Distributable**. The broker needs to have uniform distribution agreements with requesters. Therefore it should conform to industry standards wherever possible.
6. **Comprehensible.** The broker needs to have an effective reasoner that finds relevant learning objects from the available repository.

Based on these requirements, an RDF/OWL model was chosen for this research. The RDF/OWL model is naturally flexible, and standard vocabularies such as CanCore can be used to meet the distributable requirement. Centralization was achieved by careful modeling based on Apache Axis. Scalability and query performance were key goals in the selection of the RDF/OWL engine for this research – as discussed later. Integration, scalability, and query performance were addressed in the system architecture shown in Figure 1.

The architecture consists of three main components: the *Broker*, *Requestor* and *Provider*. Although many types of requestors are possible, we choose to implement a requestor (Figure 2) designed to help a user find, retrieve and display SVG slideshow presentations, such as those produced by the Learning Object Presentation Generator [23]. Likewise, we have implemented a provider designed to make a number of SVG slideshow presentations available to a requestor. The Broker will help the requestor search amongst the LOs available. These latter two components are implemented using Apache Axis.

The Ontology used to represent the capabilities of each provider is shown in Figure 3. Each learning object will be represented using exactly one instance of learningObject and any number of concept instances. An instance of learningObject has five properties: *name*, *description*, *url*, *filename* and *describes*. The url and filename properties are necessary to provide the requestor with the details required to retrieve a learning object. The describes property is intended to indicate the scope of the learning object's subject matter. We envision that the values of this property will come from some imported domain-specific ontology. Within the top-level ontology, we add the restriction that the root class of each domain-specific ontology must be a subclass of concept. As a result, our Broker's knowledge of various domains can be extended with ease. In our prototype, we experiment with a simple programming languages ontology shown in Figure 4. This ontology describes a small subset of Java and C++.

Unfortunately each provider's contents must be manually registered within the Ontology. Such a task is most easily accomplished through use of an Ontology editor such as Protégé[24]. In the near future, we plan to have each Provider Web Service automatically register itself upon start-up and un-register itself upon shut-down. This will help ensure that all content advertised by the broker is available.

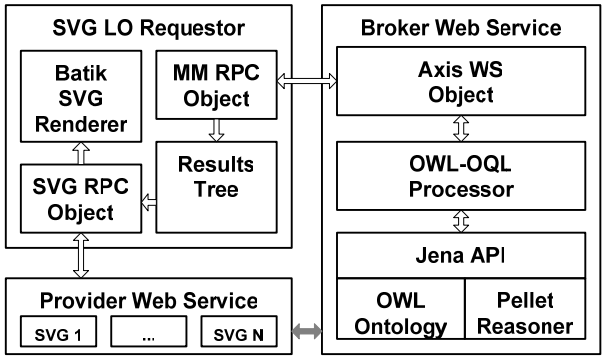


Fig. 1: The system architecture.

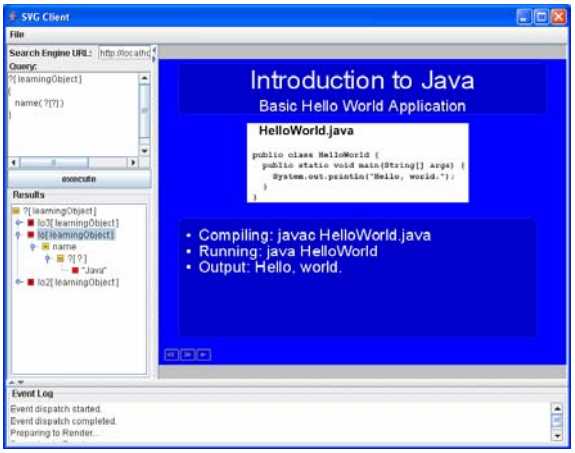


Fig. 2: The Requestor GUI.

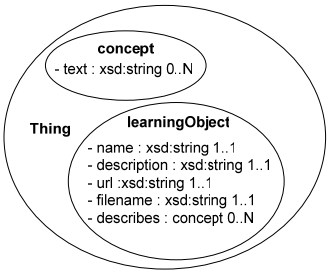


Fig. 3: Learning Object Ontology

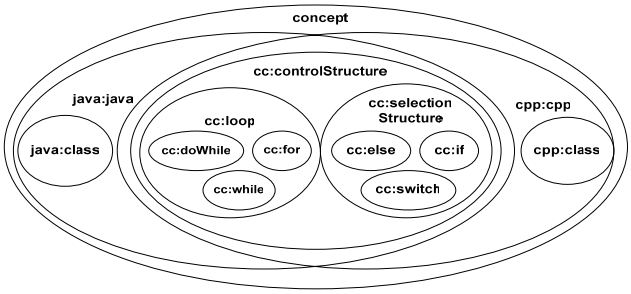


Fig. 4: A simple programming languages Ontology.

6. Developing OWL Object Query Language

A number of different query languages for OWL exist such as Versa, SeRQL, SPARQL, and OWL-QL[25]. All of them are too verbose for our purposes. As a result, we have created a new OWL query language called OWL Object Query Language (OWL-OQL).

A simple instance query (SIQ) can have the forms shown in the following table:

iname[cname]	Find a specific instance of a specific class.
?[cname]	Find all instances of a specific class.
iname[?]	Find a specific instance of any class.
?[?]	Find all instances of all classes.

We use a question mark to indicate that something is variable[1]. More complex queries require all solutions to have particular properties whose values satisfy conditions specified via sub SIQs. They have the following form:

SIQ
{
prop1(SIQ_1{ … }, …, SIQ_N$_{p1}${ … }),
…,
propS(SIQ_1{ … }, …,SIQ_N$_{pS}${ … })
}

All required properties all listed between {} brackets. A list of SIQs to be performed on the values of a property are found between () brackets. Property value SIQs can also take the form of a regular expression found between two quotes. Of course, such queries can only succeed if the property in question has string values.

The conditions that must be fulfilled for a complex query to succeed are (1) every SIQ must have at least one solution and (2) each solution to an SIQ must have all properties specified. Regarding the latter, a solution need not have only the properties specified.

The query's solutions will be returned in multi-node tree data structure expressed in XML. There are three types of nodes in the tree: SIQ nodes, solution nodes, and property nodes. The children of an SIQ node are its solutions. Each solution node has its properties (the ones specified in SIQ only) as children. Each property node has its property value SIQs as children. Our requestor creates a graphical representation of this tree in which SIQ, solution, and property nodes are shown in different colors or gray scales.

We now present a number of example queries to help clarify the ideas expressed in the above. All examples will use the ontologies shown in Figures 3 and 4. Ignoring the url and filename fields, the following table describes the learning objects we will use in our examples:

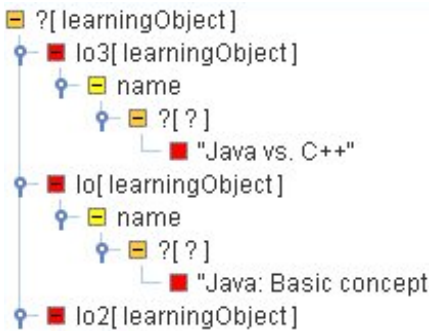
[1] Optionally a variable name can be provided after the question mark. This does not effect how the query is performed. The purpose of this feature is to help ease the extraction of data from the results xml document.

Inst.	Name	Description	Describes
lo	Java: Basic concepts	A discussion of control structures and classes.	loi1[java:class], loi2[cc:loop], loi3[cc:selectionStructure],
lo2	C++: Basic concepts	A discussion of control structures and classes.	lo2i1[cpp:class], lo2i2[cc:loop], lo2i3[cc:controlStructure],
lo3	Java vs. C++	A comparison.	lo3i1[java:java], lo3i2[cpp:cpp]

The following query can be used to obtain the name of each learning object:

?[learningObject]{ name(?[?]) }

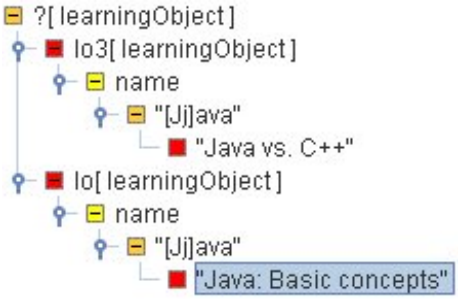
The results tree is the following:



The following query will return all learning objects whose name contains Java or java:

?[learningObject]{ name("[Jj]ava") }

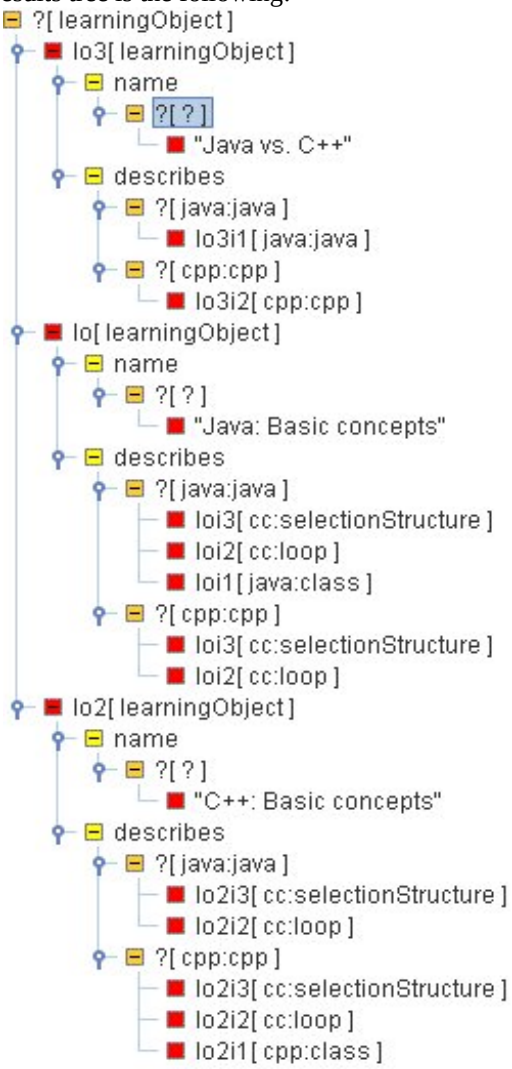
The results tree is the following:



The following query could be used to get the names of all learning objects that are related to Java and C++ in some way:

?[learningObject]{
name(?[?]),
describes(?[java:java], ?[cpp:cpp])
}

The results tree is the following:

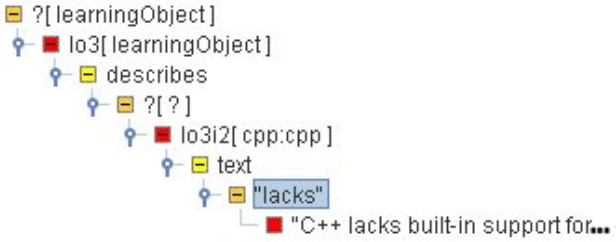


Recall that according to Figure 3, a concept has a property called text. The idea behind this field is as follows. The text field of some concept instance can optionally contain all text contained in the LO that is related to that concept. This allows combined keyword-based and ontology-based searches to be performed. Such a feature would be useful when searching for something that is not described in the ontology directly. For example, suppose that the text field of the "Java vs. C++" presentation's cpp:cpp instance had the value "C++ lacks built-in support for multithreading. Therefore, developers must rely on the facilities provided by the OS.".

The following query would find this presentation:

```
?[ learningObject ]{
  Describes( ?[ cpp:cpp ]{ text("lacks") } )
}
```

The results tree is the following:



6. Implementation

In this section, we mostly describe the implementation of our Broker shown in Figure 1. The other components are very simple and do not to warrant a detailed discussion. The provider is a simple remote procedure call (RPC) web service with one method which returns the SVG specified via the filename argument. The requestor consists of mostly GUI code.

Initialization of all Broker components is performed by the AxisWS constructor. It will initialize a Jena OntModel instance with the ontologies shown in Figures 4 and 5. This OntModel instance is then passed to an OWLOQLqueryProcessor instance, the sole member variable of AxisWS. As for the other responsibility of AxisWS, it provides a remotely callable interface currently consisting of one method that takes a query string as an argument and returns an xml results document.

The OWL-OQL processor is the most complex component of the Broker. The architecture of this component is shown in Figure 5 (The figure omits some details: the return types of all class methods, all arguments of resultsDocBuilder's methods, and OWLPropertyModel's overriding of all inherited methods).

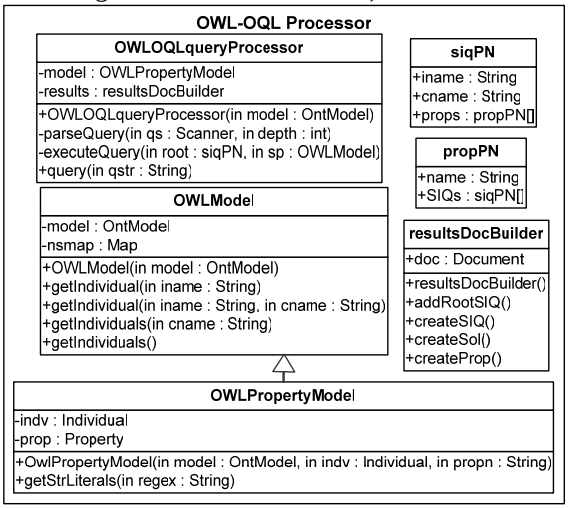


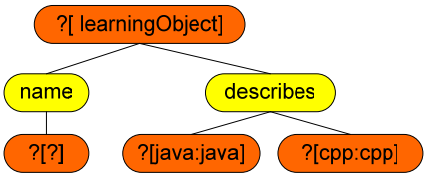
Fig.5: Architecture of OWL-OQL Processor.

The only class users of this component need knowledge of is OWLOQLqueryProcessor. It has one public method called query that takes a query string as an argument. This method will parse a given query, execute the query, and return an XML document containing the results. It is implemented using the recursive parseQuery and executeQuery methods.

A call to parseQuery will build a tree data structure representing the query. The function relies on the regular expression facilities provided by Java to extract each component of the query. These components are placed into a tree with two types of nodes: SIQ nodes and property nodes. As an example, recall the following query:

```
?[ learningObject ]{
  name( ?[?] ),
  describes( ?[java:java], ?[cpp:cpp] )
}
```

Given the above query, parseQuery would produce the following data structure:



After creating the parse tree, the query method will call executeQuery with the parse tree root and an instance of OWLModel as arguments. The pseudocode shown in Figure 6 describes the executeQuery method and how it is called.

The following is a list of key points regarding the algorithm:

- since each solution has its own set of values for each property, all property value SIQs must be performed once for each solution.
- A recursive call to executeQuery is used to process each property value SIQ. These calls are passed an OWLPropertyModel instead of an OWLModel to limit the search space to the values of the property in question.
- If any of these recursive calls fail (ie., the SIQ has no solutions), the remaining SIQs are not executed for the current solution under consideration since all property value SIQs must succeed for the solution to be valid.

```
Document query( String qstr )
{
   siqPN root = parseQuery( new Scanner(qstr), 0 );
   Element r = executeQuery( root, model );
   //recall that model is an instance of OWLModel
   //so the first SIQ will consider all individuals
   results.addRootSIQ( r );
   return results.doc;
}

Element executeQuery( siqPN root, OWLModel sp )
{
   Iterator solutions = call to appropriate method of sp.
      Information needed for args taken from root;
   propPN[] props = root.props;
   ArrayList solList = new ArrayList();

   failed: while( solutions.hasNext() )
   {
      Solution sol = (Solution)instances.next();
      ArrayList propList = new ArrayList();

      for( int i=0; i<props.length; i++ )
      {
         OWLPropertyModel m = new
            OWLPropertyModel( sp.model, sol, props[i].name );
         ArrayList siqList = new ArrayList();

         for( int j=0; j<props[i].SIQs.length; j++)
         {
            Element inst = executeQuery( props[i].SIQs[j], m );
               // switch to model m to search amongst
               // the values of property i
             if( inst == null ) continue failed;
             else siqList.add( inst );
         }
         //create a property Element with SIQs as children
         propList.add(results.createProp(
            root.props[i].name, siqList ));
      } //end of properties loop
      String iname = sol.getInstanceName();
      String cname = sol.getClassName();
      //create a solutions Element with props as children
      solList.add( results.createSol( iname, cname, propList ) );
   } //end of solutions loop
   if( solList.size() > 0 ) return results.createSIQ(
      root.iname, root.cname, solList);
   else return null;
}
```

Fig. 6: Pseudocode describing the executeQuery method.

We conclude this section with a brief discussion of the implementations of OWLModel and OWLPropertyModel. The query methods of OWLModel are mostly thin wrappers of methods provided by Jena's OntModel class. Their implementations can be summarized as follows:

Method	Description of Implementation
getIndividual(String iname)	Calls model.getIndividual(String iname) to find the individual in the Ontology whose name is iname.
getIndividual(String iname, String cname)	Same as above with one difference: the query will fail if the individual is not a member of class cname.
getIndividuals(String cname)	Calls model.getOntClass(String cname) to retrieve an instance of OntClass representing the class. Then calls OntModel.listIndividuals(OntClass c) to find all individual belonging to that class.
getIndividuals( )	Calls model.listIndividuals() to get all individuals in the Ontology.

OWLPropertyModel overrides all individual retrieval methods it inherits. Its constructor will initialize the Individual and Property member variables. The property member is initialized by calling model.getOntProperty(propn). Each query method begins by obtaining an Iterator of property values by calling indv.listPropertyValues(prop). Each property value is inspected one by one in a sequential manner to produce a new Iterator containing only those property values that meet the search criteria. The methods conclude by returning this Iterator.

7. Conclusion

This article describes a new brokering architecture that exploits Semantic Web technologies for locating multimedia learning objects. Central to our architecture is the Brokering Web Service, which provides lightweight query processing and reasoning in response to Requestor OWL-OQL queries. We describe the use of this Broker, its associated ontologies and its reasoning mechanism in an intelligent multimedia learning object search engine prototype. The prototype has been tested via searches for multimedia learning objects representing SVG presentations on programming languages like Java and C++. The authors aim to develop this prototype to include different types of learning objects and to expand it to register such learning objects using a standard registry service like the UDDI. We are also considering further development of OWL-OQL as a general-purpose OWL query language.

Acknowledgment

This research is supported by the first author NSERC Discovery grant 50-16110105-2004.

References

[1] Reformat, M. DengMing Li Cuong Ly, Approximate reasoning and Semantic Web Services, IEEE Annual Meeting on Fuzzy Information, 2004, Proceeding of NAFIPS '04, 27-30 June 2004 Volume: 1, On page(s): 413- 418 Vol.1

[2] Brenner, W.; Zarnekow, R.; & Wittig, H. Intelligent Software Agents, Foundations and Applications. Berlin, Germany: Springer-Verlag, 1998.

[3] Antonio Guerrero, Vector A. Villagrd, Jorge E. Lpez de Vergara, Including management behavior defined with SWRL rules in an Ontology-based management framework, Proceedings of the 12th Annual Workshop of HP Openview University Association, Porto, Portugal, 10-13 July 2005.

[4] Christopher J. Matheus, Mitch M. Kokar, Kenneth Baclawski and Jerzy Letkowski, Constructing RuleML-Based Domain Theories on Top of OWL Ontologies, Lecture Notes in Computer Science, Book Name: Rules and Rule Markup Languages for the Semantic Web, Volume 2876/2003.

[5] F. Tewissen, N. Baloian, U. Hoppe, E. Reimberg, ""MatchMaker": Synchronising Objects in Replicated Software-Architectures," criwg, p. 60, 6th International Workshop on Groupware (CRIWG'00), 2000.

[6] Rubén Lara, Dumitru Roman, Axel Polleres and Dieter Fensel, A Conceptual Comparison of WSMO and OWL-S, Lecture Notes in Computer Science, Volume 3250/2004

[7] Massimo Paolucci, Naveen Srinivasan, and Katia Sycara, Expressing WSMO Mediators in OWL-S, The Third International Semantic Web Conference (ISWC 2004), November 8, 2004, Hiroshima,Japan.

[8] Dieter Fensel, Stefan Decker, M. Erdmann, and Rudi Studer, 'Ontobroker: The very high idea', in 11. International Flairs Conference (FLAIRS-98), Sanibal Island, USA, (1998). AAAI Press.

[9] Juan M. Santos, Luis Anido and Martín Llamas, Design of a Semantic Web-based Brokerage Architecture for the E-learning Domain,A Proposal for a Suitable Ontology, 35th ASEE/IEEE Frontiers in Education Conference, October 19 – 22, 2005, Indianapolis, IN

[10] Arthur Stutt, Maria Cleci Martins and John Domingue, Tribalization, E-learning and the Semantic Web, First Intl. Conference on Educational Technology in Cultural Context, Joensuu, Finland, 2002

[11] N. Capuano, M. Gaeta, A. Micarelli, E. Sangineto Diogene: a Semantic Web-Based Automatic Brokering System, AIS SIGSEMIS bulletin, vol. 1, no. 3, p. 65-67, October 2004.

[12] Lai-Chong Law and E.T. Hvannberg, Complementarity and convergence of heuristic evaluation and usability test: a case study of universal

brokerage platform, ACM International Conference Proceeding Series; Vol. 31, Proceedings of the second Nordic conference on Human-computer interaction, Aarhus, Denmark, pp71 – 80, 2002.

[13] Duval, E., Vandepitte, P., & Ternier, S., LOMster: Peer-to-peer Learning Object Metadata, World Conference on Educational Multimedia, Hypermedia and Telecommunications 2002(1), 1942-1943.

[14] A. Cavallaro, A.Faro, D.Giordano and S. Mineo, A brokerage system for context-based retrieval of learning objects, Proceedings of the 3rd International Conference on Multimedia and Information and Communication Technologies in Education, m-ICTE2005, Seville (Spain) 29 November till 2 December 2006.

[15] Kay, J., & Holden, S. (2002). Automatic Extraction of Ontologies from Teaching Document Metadata. In L. Aroyo, & D. Dicheva (Eds.) Proceedings of ICCE 2002 Workshop on Concepts and Ontologies in Web-based Educational Systems (pp. 25-28). Auckland, New Zealand.

[16] Holger Wache, Thomas Vogele, Ubbo Visser, Heiner Stuckenschmidt, Gerhard Schuster, Holger Neumann, and Sebastian Hubner, 'Ontology based integration of information - a survey of existing approaches', in IJCAI-01 Workshop: Ontologies and Information Sharing, pp. 108–117, Seattle, WA, (2001).

[17] Li Ding et al., Swoogle: a search and metadata engine for the semantic web, Conference on Information and Knowledge Management, Proceedings of the thirteenth ACM international conference on Information and knowledge management, Washington, D.C., USA Pages: 652 – 659, 2004

[18] Visser, U. and Schuster, G., Finding and Integration of Information-A Practical Solution for the Semantic Web,In:Proceedings of ECAI 02, Workshop on Ontologies and Semantic Interoperability, Lyon, France, 73-78, 2002.

[19] Bijan Parsia and Evren Sirin, Pellet: An OWL DL Reasoner, Third International Semantic Web Conference (ISWC04)November 7-11, 2004, Hiroshima, Japan.

[20] Ian Horrocks. Fact++ web site. http://owl.man.ac.uk/factplusplus/.

[21] Ian Horrocks. The fact system. In Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98, pages 307 – 312. Springer-Verlag, May 1998.

[22] Ralf Moller Volker Haarslev. Racer system description. In International Joint Conference on Automated Reasoning, IJCAR 2001, 2001.

[23] J. Fiaidhi, S. Mohammed, M. Hahn, Developing Multimedia Learning Objects for the Semantic Web, Int. Journal of Computer Science and Network Security, Vol. 6, No. 11, Nov. 2006, pp 121-129

[24] protege.stanford.edu/

[25] www.w3.org/TR/owl-features/

Jinan Fiaidhi received her PgD and PhD in Computer Science from England UK (Essex and Brunel Universities) during 1983 and 1986 respectively. She served as faculty member at various universities including Philadelphia, Applied Science, Sultan Qaboos and Lakehead Universities. Since January 2002, she is with Lakehead University, Ontario/Canada. Currently she holds the rank of Professor and the designates of MBCS and I.S.P. of Canada. Her research interests include Learning Objects, Multimedia/Virtual Environments and Peer-to-Peer Learning.

Sabah Mohammed received his MSc and PhD in Computer Science from England UK (Glasgow and Brunel Universities) during 1981 and 1986 respectively. He served as faculty member at various universities including Amman, Philadelphia, Applied Science, Oman HCT and Lakehead Universities. Since January 2002, he is with Lakehead University, Ontario/Canada. Currently he holds the rank of Professor and the designates of MBCS. His research interests include Image Segmentation, Image Protection and Open Source Multimedia.

Marshall Hahn is HBSc fourth year Computer Science student. He is conducting the research in this article as part of his NSERC grant on multimedia programming.