Advanced Research & Studies Journal

A collaborative Model to Filter Java Learning Objects

Kayla Huff†

† University of Virginia, Charlottesville, USA

Paper type: Research article
Journal ISSN: 2692-2800

Abstract

The model enforces a collaborative infrastructure for authoring, searching, recommending and presenting Java source code learning objects. The new model uses two specialized filtering engines which work simultaneously: CollabroSearch and CollabroRecommender to present relevant LOs from presented queries or from the mined text collected from the collaborative chatting channel between users. Experiments on Java source code testbed indicated that the proposed model is able to outperform any primitive collaborative environment that support some searching and discovery primitives such as the I-help system developed by ARIES Lab, Department of Computer Science, University of Saskatchewan (http://www.cs.usask.ca:7777/ihelp11/entrance.html) and any of the commercial collaborative environments such as JCE, Centra99, PaceWare, Grooves and Tango. The combination methods of the flexible mixture model (ReccoSearch) is rather preliminary. As a near future work, we plan to explore approximate searching techniques based on query expansion (Fiaidhi, Mohammed, Jaam, Hasnah 2003) as well as to explore different types of recommending algorithms (besides the k-nearest collaborative filtering that we originally used) such as the Item-Basedrecommending algorithm (Sarwar et al, 2001). The ultimate future goal of this research is to link the RecoSearch system to the POOL of LOs repositories(Hatala and Richards 2002) via utilizing the JXTA APIs (www.jxta.org/) to unable collaborators to search and recommend other LOs from the major repositories available to the academia.

Introduction

'On Shifting Sands' as a conceptual metaphor for this paper is a consequence of my interest in
Digital repositories populated with learning objects are becoming popular tools in the creation of instructional technologies (Recker and Wiley, 2001). Many current efforts to facilitate the discovery and instructional use of learning objects (LOs) recommend the use of simple content-based search engine (e.g. ONES Project: Puusttjarvi and Poyry 2003) or the use of smart interface associated with learning object repositories (e.g. 4-Tier IBM Learning Interface: Dodani 2002). While both methods have their own advantages, they fail to filter useful learning objects in many situations (Recker and Walker 2000). However, as we argue in this paper, that by incorporating components from both methods with a LOs recommender system capabilities, one can overcome these shortcomings. In this paper, we present an elegant and effective model for combining content, collaboration, collaborative filtering and searching techniques in an integral engine that we call RecoSearch.

Advanced Research & Studies Journal

To achieve a working model prototype for our integration, we will restrict our application to filtering "Java Learning Objects" in a collaborative teaching environment. In this direction, the challenging aspect of teaching a programming course is how to provide the right information in the right context at the right time to the right person. The introductory object-oriented programming course (taught in Java) at Lakehead is made up of students who come from a variety of disciplines (computer science, mathematics and engineering) and have different levels of programming experience. For example, some have taken C++ with object first approach and some have extensive programming experience in C or another procedural language like Fortran 90. This audience has little conceptual understanding of multi-class programs, object oriented design, class methods, parameter passing, and inheritance. The first challenge in such introductory programming courses is to bring all students to a common learning environment within 3-4 weeks. One of the major problems of bringing students to the same level of understanding in a short period of time is the lack of an effective communication mechanism between instructor-student and student-student to share crucial knowledge at the right time. Students often misunderstand concepts and thus apply them incorrectly; which leads to hours of wasted time spent on debugging logically incorrect code.

We believe that we can address some of these problems by creating our RecoSearch environment. We will focus in this environment on two aspects:

- presenting key Java programming concepts by utilizing learning objects;
- establishing a collaborative platform for discussions, searching, recommending and exchanging Java learning materials.

Addressing the first aspect, we need to make sure that all course related material is organized into a repository of information objects. The key components of the repository are textbook content, Java source code examples and review questions. Then instructors can create learning objects using the repository of information objects; and share them with colleagues and students. In the long run, this will lead to a library of learning objects to which instructors can make contributions, as well as use publicly available learning objects in their own courses.

Addressing the second aspect, we need to make certain that all knowledge components are assembled under one collaborative environment. We will achieve this through creating a collaborative environment that employs at least two XML messenger channels (one for sending JLO and the other for users chatting). The description of such collaborative environment has been explained in our earlier article (Fiaidhi and Mohammed 2004). However, our ultimate future goal to link this collaborative environment to the POOL of learning objects (Hatala and Richards 2002) utilizing the JXTA APIs (www.jxta.org). This collaborative environment will serve in various ways, including a) it will allow instructors to create and share customized content to meet the learning objectives of individuals or groups; b) it will allow students to create personalized learning profiles and share them with others; c) it will create an environment where students can discuss course-related java programming material in its own context. Figure 1 illustrates the main components considered by the RecoSearch model.

The features of this model will be described in the next sequel. However, we would like first to shade the light on why having only a single search engine or recommender engine is not effective for searching relevant learning objects.

The Problems of using Pure LOs Search Engines:

There are presently countless Learning Objects available for corporate and academic use. Table 1 list few of such notable repositories.

Table 1: Notable LOs Repositories.

Repository Name	URL Reference
eduSource	http://edusource.netera.ca
Splash	http://edusplash.net
MERLOT	http://www.merlot.org
CAREO	http://careo.netera.ca

Advanced Research & Studies Journal

ESCOT	http://www.escot.org
EOE	http://www.eoe.org
GEM	http://www.thegetway.org
IDEAS	http://ideas.wisconsin.edu
LRC	http://wwww.edlrc.unsw.edu.au

Despite the advantages of having access to such ever-growing object libraries, E-learning paradigm now faces a more pressing challenge: how to find the most appropriate learning object for a given user/purpose? Indeed, there are many different ways in which to locate material in a Learning Object repository. Searching and browsing are two obvious methods but the value of these methods depends on the information and organizational structure or standard of the repository. On one hand, browsing represent the ability to explore through categories and see all that is on offer in each category. This is the "discovery" mode in which unknown nuggets are often uncovered. If a repository has many objects, say 10,000 or more, the classification categories need to be well structured and extend several layers deep to enable each component of the classification "tree" to contain a manageable number of objects. General classification categories, or taxonomies, are widely used in libraries (e.g. the Dewey system or National Library of Congress system) but many subjects also have much more detailed subject-specific taxonomies. These taxonomies are essential tools for people browsing through repositories. In contrast to a library full of books, where the physical book can only sit on one book-shelf, learning objects repositories can have a single asset represented at many different locations in the taxonomy. This means that many different browsing approaches can lead to the discovery o f suitable objects. One problem with browsing is that it can be time-consuming. Imagine how much more effective it would be for each person using the repository to define their own taxonomy and "store" learning objects in the context that means most to them - locating these objects again and again would be simple.

On the other hand, searching is often based on keywords or the use of metadata tags. This works well if the search is concerned only with the content of the material. Keyword searches can be expanded to include the text of the material itself. The true power of searching is enabled when objects in the repository include metadata description. Library Science has long recognized this type of searching as the old-fashioned card indexes which have been given way to computer-based records using one of several established standards to describe published works/objects. Learning objects require considerably more metadata details concerning how the material may be used: the type of resource, who might learn from it, in which context they might learn from it, age or experience of expected learner, typical time required for learning, and many more. There are many standard specifications, such as IMS and ADL, IEEE standard Learning Object Metadata (LOM), and CanCore which uses as many as 70 fields/tags to describe and classify learning objects. Searching becomes even more powerful when each specific metadata field is used. Another mode of searching a digital repository is by means of a software agent. In this case a computer application, such as an LCMS or another digital repository, interrogates the digital repository based on some defined query. The results of the query are returned through the application, which initiated the query using any query processing language (e.g. XML schema, XQuery, XPath, XQL, XML-QL, QUILT). Moreover as the granularity of the learning objects decreases and as the size of repositories increases, there will also be a need for much more fine-grained topic descriptions than any standard can provide. Even advanced searches can overwhelmingly return hundreds of thousands of results (Gaaster 1997).

However, searching for LOs within heterogeneous repositories is a far more complicated problem. In searching for such LOs we must first decide on appropriate metadata scheme. But which one! Typically, these learning objects may be lesson content stored as text, audio-visual or interactive media files, or simply learning activity templates expressed in a learning design format. Despite their apparent ubiquity, the locating and re-use of LOs is hampered by a lack of coordinated effort in addressing issues related to their storage, cataloguing and rights management. Strident efforts have been made to create portal repositories by communities such as CANARIE, Merlot, SMETE and CAREO. Not surprisingly, each entity produces a rather individual reflection of its own perceived organizational needs, and the

Advanced Research & Studies Journal

concept of making all these repositories work together while laudable, has received less attention. More recently, the E-learning community has been focusing on the ability to connect and use resources located in distributed and heterogeneous repositories. This process closely resembles the initiatives in the domain of digital libraries, to the extent that there are initiatives such as the POOL, NSDL, IMS DRL, eduSourceCanada, and the OKI projects, for connecting different types of LOs repositories as well as the traditional digital libraries networks (Hatala and G. Richards, 2002). This will provides us with an effective searching infrastructure when creating such large and open networks. Unfortunately these efforts are just at their initial stages and require huge resources and synchronizations to be mature.
Overall, the traditional search process within single or heterogeneous LO repositories may prove to be inadequate in a society that demands immediate, reliable results in order to meet the demands of their customers. What we argue in this article is that one can alleviate such problems by trying to collaboratively "predict" what users will want rather than expect them to completely define their needs through searching parameters only.

Problems using Pure LOs Recommender Systems:

Recommender systems (RS) (Resnick and Varian 1997) suggest information sources and products to users based on learning from examples of their direct likes and dislikes or from their collaborative or group previous preferences. Unfortunately such systems are rarely used for recommending LOs, but they are widely and successfully used in many other online systems (e.g. in amazon.com) to suggest items that users may "find interesting". The RS recommendations are generated using two main techniques: content-based, and collaborative filtering. Content-based systems require manual intervention, and do not scale to large item bases. Using such technique users are required to specify their preferences explicitly and in detail, this process can become so tedious or impractical that the system is essentially inaccessible to some users. However with collaborative filtering (CF) (Goldberg et al, 1992) systems do not depend on the semantics of items under consideration; instead, they automate the recommendation process based solely on other user opinions. This is the most interesting point in CF where their algorithms doesn't need a representation of the items in term of features but it is based only on the judgments of the user community. Because of this, CF can be applied to virtually any kind of item or object: papers, news, web sites, movies, songs, books, learning objects, locations of holidays, stocks. Since CF techniques don't require any human intervention for tagging content, they promise to scale well to large item bases.
While CF algorithms are promising for implementing large-scale recommender systems, they have their share of problems. The problems with pure CF systems can be classified in three domains: problems affecting new user start up, sparsity of useful information for existing users, and relatively easy attacks on system correctness by malicious insiders (Hayes et al, 2002).

Java Learning Objects: Authoring, Packaging and Presentation

Java learning object (JLO) can be defined as an integrated module containing the core text, code examples, review questions, supplementary material, and Java programming lab exercises. The traditional standard format used for representing Java source code as well as its related materials is plain text-based. However, the basic shortcoming of the plain text format is its "flatness", the absence of almost any explicit structure. A free-form plain text document represents a series of tokens, where every token is a simple character string. Any structure required by the programming language has to be coded into the relationships between such tokens. This structure becomes apparent only after a rather sophisticated and complicated process of parsing. On the other hand, the XML document model has inherent hierarchical structure easily designed to accommodate any structure including Java source code constructs. For this particular reason, learning object is uniquely described by an XML document that includes metadata and semantic relationships between LO components. The XML document also serves

Advanced Research & Studies Journal

as an interface for future search and retrieval of LO's. Moreover, the LO will provide an open interface for a connection to other components, such as external assessment engines.
Each LO complies to a standard metadata format. However, most of the academia uses the CanCore standard (Friesen 2002), which will assure the intercompatibility and reusability among most of learning platforms. CanCore is an emerging standard for creating, sharing and extending learning objects independent of the platform or the audience and used widely by most of the academic institutions in North America. Each LO is editable and can be tailored (by creating a learning profile) to meet the needs of an individual student or a group of students. Learning object profiles (or schema/DTD) then will be distributed to students as email attachments or can be placed in a course management system for easy downloads. Learning object profile is a relatively small XML document that describes the components within the learning object. The XML Schema is a valuable concept which enables you to define your own XML vocabulary. An XML vocabulary is an industry-specific XML information model or document type that you define for XML data sharing. In other words, you define constraints that specify what a particular group of XML documents should always look like. Document creators, programmers, graphic designers, and database specialists use a constrained document type as the basis for creating compatible application pieces. You can define an XML vocabulary by constraining your XML file. Figure 2 illustrates a simple Java source code and its equivalent XML and Schema.

```
import java.applet.*;
import java.awt.*;
public class FirstApplet   extends Applet {
  public void paint(Graphics g) {
    g.drawString("FirstApplet", 25, 50);
  }
}
```

– – – – – – – – – – (a) Java Source Code – – – – – – – – – – – – –

```
<java-source-program>
<import-declaration>import java.applet.*;
  </import-declaration>
<import-declaration>import java.awt.*;
  </import-declaration>
<class-declaration>
<modifiers>public</modifiers> class
  <class-name>FirstApplet</class-name>  extends  <superclass>Applet</superclass>
{
 <method-definition>
  <modifiers>public</modifiers>
       <return-type>void</return-type>
    <method-name>paint</method-name>
     (<formal-arguments>
        <type>Graphics</type>
        <name>g</name>
       </formal-arguments>)
    <statements>{
    g.drawString("FirstApplet", 25, 50);
  } </statements> </method-definition>
}
</class-declaration>
</java-source-program>
```

– – – – – - (b) XML equivalent code – – – – – – – – – – – – – – - -

Advanced Research & Studies Journal

```
<?xml version="1.0" encoding="utf-16"?>
<xsd:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
version="1.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="java-source-program">
  <xsd:complexType>
   <xsd:sequence>
                <xsd:element maxOccurs="unbounded" name="import-declaration"
type="xsd:string" />
       <xsd:element name="class-declaration">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:element name="modifiers" type="xsd:string" />
          <xsd:element name="class-name" type="xsd:string" />
          <xsd:element name="superclass" type="xsd:string" />
          <xsd:element name="method-definition">
           <xsd:complexType>
            <xsd:sequence>
             <xsd:element name="modifiers" type="xsd:string" />
             <xsd:element name="return-type" type="xsd:string" />
             <xsd:element name="method-name" type="xsd:string" />
             <xsd:element name="formal-arguments">
              <xsd:complexType>
               <xsd:sequence>
                <xsd:element name="type" type="xsd:string" />
                <xsd:element name="name" type="xsd:string" />
               </xsd:sequence>
              </xsd:complexType>
             </xsd:element>
             <xsd:element name="statements" type="xsd:string" />
            </xsd:sequence>
           </xsd:complexType>
          </xsd:element>
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

----(c) Equivalent XML Schema **----------**

Figure 2: FirstApplet.java converted to XML and XML Schema.

Fortunately, all Java source code comply to the same syntactic schema as defined by the Javadoc. One can use the Javadoc™ plug-in with the XXE editor (http://webdesign.about.com/cs/software/gr/aapr-xxe.htm) to directly convert the Java source file on the fly to an equivalent XML file formated with the standard Javadoc tags. In this case, no schema validation will be required. However, there are many other dedicated tools and APIs for this that can be used to used any to convert Java source files into XML format (e.g BeautyJ (http://beautyj.berlios.de/), Jato API (Krumel 2001), and JavaML

Advanced Research & Studies Journal

(http://www.cs.washington.edu/homes/gjb/JavaML/)) but without the use of the Javadoc plug-in. In this case, one need to validate the XML schema with the XML file generated. The designer can generate XML Schema for any typical XML file format using simple tools as the XML Schema Generator (http://www.xmlforasp.net/CodeBank/System_Xml_Schema/BuildSchema/BuildXMLSchema.aspx) and then uses a simple program as the one displayed in figure 3 to validate the given XML file with the schema generated.

```
import org.w3c.dom.*;
import org.apache.xerces.parsers.*;
import org.apache.xerces.dom.*;
import org.xml.sax.*;
import java.io.*;
import java.util.*;
import javax.xml.parsers.*;
public class TestValidator {
   /** Creates a new instance of TestValidator */
   public TestValidator() {
   }
   public static void main(String[] ar)
   {
      try
      {
         DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
         dbf.setValidating(true);
         dbf.setAttribute(
         "http://java.sun.com/xml/jaxp/properties/schemaLanguage",
         "http://www.w3.org/2001/XMLSchema");
         DocumentBuilder db = dbf.newDocumentBuilder();
         Document doc = db.parse("C:\\new.xml");
      }catch(Exception e)
      {
         System.out.println("Exception: "+e.getMessage());
      }
  }
}
```

Figure 3: A Java Program to validate the XML file against given XML Schema.

The BeautyJ, JavaML and Jato API are all open-source which can be downloaded and incorporated with any Java-based application environment. There are many advantages of using these packages/APIs over directly employing traditional Java XML APIs such as JDOM, SAX, XSLT or DOM(Simic and M. Topolnik 2003). With such packages/APIs, developers simply express the XML elements that map from specific Java Source. Their packages/APIs interpreters then implement the necessary parsing and generation algorithms to accomplish the desired actions. As such, you avoid the monotonous, monolithic, and difficult-to-maintain XML parsing and generation code using the mentioned traditional Java XML APIs.

Binding Java Source Expressed in XML to Java Objects:

Currently XML and Java technology are recognized as ideal building blocks for developing Web services and applications that access such services including eLearning systems built upon learning objects. But how do you couple these two technologies in practice? More specifically, how do you

Advanced Research & Studies Journal

access and use an XML document (that is, a file containing XML-tagged data) through the Java programming language? One way to do this, perhaps the most typical way, is through parsers that conform to the Simple API for XML (SAX) or the Document Object Model (DOM). Both of these parsers are provided by Java API for XML Processing (JAXP). Java developers can invoke a SAX or DOM parser in an application through the JAXP API to parse an XML document – that is, scan the document and logically break it up into discrete pieces. The parsed content is then made available to the application. In the SAX approach, the parser starts at the beginning of the document and passes each piece of the document to the application in the sequence it finds it. Nothing is saved in memory. The application can take action on the data as it gets it from the parser, but it can't do any in-memory manipulation of the data. For example, it can't update the data in memory and return the updated data to the XML file. In the DOM approach, the parser creates a tree of objects that represents the content and organization of data in the document. In this case, the tree exists in memory. The application can then navigate through the tree to access the data it needs, and if appropriate, manipulate it.
Now-a-days developers have another Java APIs at their disposal that can make it easier to access XML documents: Java Architecture for XML Binding (JAXB). (http://java.sun.com/xml/jaxb/), Caster (http://www.castor.org/), and the JiBX (http://jibx.sourceforge.net) which provide APIs and tools that automate the mapping between XML documents and Java objects. It makes XML easy to use by compiling an XML schema into one or more Java technology classes. The combination of the schema derived classes and the binding framework enable one to perform the following operations on an XML document: (1) unmarshal XML content into a Java representation; (2) access, update and validate the Java representation against schema constraint; (3) marshal the Java representation of the XML content into XML content. However, marshaling a Java object means converting it to XML format for storage or for sending, and turning an XML document back into useable Java objects is called unmarshaling. Figure 4 illustrates the JAXB architecturefor binding XML files into Java Objects.

The Collaborative RecoSearch Approach:

Converting Java source code learning objects to XML and having the ability to paring it using sophisticated APIs like JAXB will not solve primarily the problem of searching and recommending LOs to learners especially in a collaborative and distributed environments. Even the use of the SCORM (the Sharable Content Object Reference Model developed by ADL http://www.adlnet.org/) which is the standard that is supported by the large e-learning players that supposedly ensures that learning content level This would obviously be a very good thing, but the problem is that SCORM emerges from the world of learning content management systems, so the emphasis is on how content is presented to individual learners and how individual learners' paths through a course can be sequenced and tracked - not on how learners and teachers can work together to create new knowledge. Hence there is a pressing need to develop extensions to SCORM which can support the following issues:

- Support for distributed, collaborative development of consensus ontologies. This should include schema integration and merging similar ontology with slight lexical differences.
- Metadata. The ability to create, remove, and query information about LOs, such as its author, creation date, etc according to one acceptable standard like CanCore.
- Name space management. The ability to copy and move LOs, and to receive a listing of LOs at a particular hierarchy level (like a directory listing in a file system).
- Overwrite prevention. The ability to keep more than one person from working on a LO at the same time. This prevents the "lost update problem" in which modifications are lost as first one author, then another writes their changes without merging the other author's changes.
- Version management. The ability to store important revisions of a LO for later retrieval. Version management can also support collaboration by allowing two or more authors to work on the same LO in parallel tracks.

Advanced Research & Studies Journal

- Relevant LOs Recommendation. This ability to recommend relevant LOs from the relevant previous queries of other users.

Some of the above issues are related to the creation of a LO based collaborative environment. In this direction one can use the standard WebDAV based toolkits developed by the Internet Engineering Task Force (IETF) during early 1998 (Whitehead and Wiggins 1998). However, packaging LOs and publishing it on any user machine is yet another service to be added to these toolkits. In this direction we recommend the use of SceneBeans as a model for packaging Java-Based LOs (Fiaidhi, Mohammed, Sisko 2004) or simply use a JSP-Based LO publishing toolkits developed by Sun Microsystems (java.sun.com/products/jsp/docs.html).
The other two issues are related to designing two engines that can work simultaneously to search and recommend relevant or related LOs within a collaborative and distributed environment. We call these two engines as CollabroSearch and CollabroRecommender. The design issues related to these two engines are provided in the following two sections.

The CollaboSearch Engine:

Many researchers believe that searching for XML-Based learning objects within a single repository should be straightforward via searching for a matching metadata. This believe came from the fact that XML is a form of a database(Rizzolo and Mendelzon 2001) and hence searching for an XML metadata should be as easy as querying a database. According to such believe, many organizations developed various searching engines for the XML databases of documents(e.g. Amberfish, IXIASOFT, Infonyte Query, XML Query Engine, Tamino, MLE, Ultraseek, SIM, X-Hive, Xdirect, Xset, fxgrep, Xtenint, and Lore). As a "database" format, XML has some advantages. For example, it is self-describing (the markup describes the structure and type names of the data, although not the semantics), it is portable (Unicode), and it can describe data in tree or graph structures. It also has some disadvantages. For example, it is verbose and access to the data is slow due to parsing and text conversion. Actually an XML document is a database only in the strictest sense of the term. On the plus side, XML technology provides many of the things found in databases: storage, query languages, programming interfaces, and so on. On the minus side, it lacks many of the things found in real databases: efficient storage, indexes, security, transactions and data integrity, collaborative access, triggers, queries across multiple documents, and so on. For this purpose, many surrounding technologies have been developed for treating XML documents as a database management system DBMS (e.g. DTD, XML schema, XQuery, XPath, XQL, XML-QL, QUILT). However, none of such technologies are readily designed to deal with collaborative and distributed searching. There are two major issues related with searching for collaborative and distributed environment: The schema integration and the collaborative ontology.
Searching for LOs within heterogeneous repositories as well as within collaborative repositories is far more complicated problem. In searching for such LOs we must first decide on appropriate metadata schema, but which one! However, the most notable approach available today for extracting information out of various learning object with different source schemas is based on schema integration/matching techniques. Schema matching is an operation that takes two schemas as input and returns a mapping that identifies corresponding elements in the two schemas. Schema matching and integration is a critical step in many other applications: in eBusiness, to help map messages between different XML formats; in data warehouses, to map data sources into warehouse schemas; and in mediators, to identify points of integration between heterogeneous databases. Although the techniques used for schema matching includes variety of mechanisms(E. Rahm, and P. A. Bernstein 2001) (e.g. linguistic matching, machine learning, structural match, constraint match, and hybrid matchers), only the simple linguistic matching techniques are used (e.g. Cupid and SPHINX Systems). For the purpose of effectively matching schemas to extract information out of learning objects we find the most relevant type matching should be based on semantic relationships (Fiaidhi, Passi and Mohammed 2004). There are six semantic relationships defined in (Passi et al 2002) for schema integration – identical, equal, equivalent, subset, unique, and incompatible. Elements are identical if they have the same name and belong to the same namespace, since each namespace is unique and each element name

Advanced Research & Studies Journal

within a given namespace is unique. Elements are equal if they have the same name and same definitions but belong to different namespaces. Elements are equivalent if they have different names but the same definitions. Elements with the same name, different namespaces, and the condition that the children of one element exist as a direct child group of the second element that is defined in terms of an all or choice satisfy the subset semantic relationship. Elements are unique if they have different names and different definitions that are not equivalent to the definition of any other element across all the local schemas. Elements with the same name, different namespaces and definitions that do not satisfy the subset semantic relationship are seen as incompatible. The above semantic relationships help identify matches and mismatches between elements and conflict resolution.

Collaborative ontology is the other key factor for enabling effective search in a collaborative and distributed environment. Normally, ontologies are normally built and maintained independently of each other in a distributed or collaborative environment. Therefore searching for LOs described by two different schemas, cannot be easily achieved because of the different reference ontologies (Klein 2001). Obviously, a solution to this problem requires the construction of an integral or collaborative ontology (Fiaidhi, Mohammed, Jaam and Hasnah 2003). There are many researchers who attempted to develop one general-purpose easy-to-use tools for creating, evaluating, accessing, using, and maintaining collaborative ontology such as Ontolingua (http://ontolingua.stanford.edu/). However, incorporating such one big tool in a learning environment that deals with Java LOs may impose several technical difficulties. In this direction, we find the use of several dedicated tools for editing ontologies such as Java Ontology Editor (JOE http://www.cse.sc.edu/research/cit/demos/java/joe/) and the JADE tool for ontology integration (http://gaper.swi.psy.uva.nl/beangenerator/content/main.php) is more effective to support our design objectives. Using the two mentioned tools, one can construct an ontology server that supports not only the development of ontologies by individuals, but also the process of achieving consensus on common ontologies by distributed and collaborative groups. The ontology server will provide many of the facilities that are crucial for promoting the use of ontologies for collaborative search including:

Browsing and retrieval of ontologies from repositories.

Assembly, customization, and extension of ontologies from repositories. This requires the ability to identify and resolve name conflicts and to augment descriptions of terms from the assembled ontologies. We may use for this purpose our Fuzzy Similarity algorithm which matches ontologies by considering their fuzzy lexical differences (Fiaidhi and mohammed 2004, Fiaidhi and Mohammed 2004a).

Facilities for translating ontologies from repositories into typical application environments. We can use for this purpose translators that use for example CORBA's IDL representation (Mowbray and Zahavi 1995)

Facilities for programmatic access to ontologies so that remote applications have reliable access to up-to-date term definitions.

One more issue that is particularly important for searching LOs within a collaborative and distributed environment is the ability to use indexed search. With indexed search the search engine search the collaborative index of all the LOs within the collaborative/distributed environment. This type of search prove to be be very fast (PCAI 2001). The final component of the CollabroSearch engine is the query server. This server must have the ability for searching collections of XML-based LOs beyond the capabilities found in both XML databases query languages or the simple SQL full-text search engine. The primary difference is in the retrieval mechanism. The LO query server Search, by contrast, should supports a rich query language which provides both sophisticated full-text retrieval and retrieval of highly-structured LOs. It should utilize both the collaborative ontology and the schema integration services as well as to be able to work in parallel with the CollabroRecommender engine. In this direction we are imagining a query server like the Dieselpoint (www.dieselpoint.com/) which can provide some of the required processing services besides providing a full range of linguistic tools, including a thesaurus, stemming, and a "Did you mean..." feature to alert users to possible misspellings.

The CollabroRecommender Engine

Advanced Research & Studies Journal

The CollabroRecommender is sort of filtering engine or a sort of recommender algorithm where the recommendations are based on a database of the users ontology ratings as opposed to content-based recommender algorithms that are based on the characteristics of the learning objects to be recommend. The basic principle behind such type of filtering is that clients must first share some information about themselves by rating some of the learning objects features they know, so that, in turn, they can get accurate recommendations based on the premise that users looking for LOs should be able to make use of what others have already found and evaluated. The current recommender systems provide tools for readers to filter documents based on which ones were read and liked (i.e. highly rated) by previous readers. Recommender systems based on automated filtering should predict new LOs for a user based on predictive relationships discovered between that user and other participants of a collaborative community. Most of the successful research and commercial systems in this area use a nearest-neighbour algorithm model for generating predictions. Such predictions that are based on the nearest-neighbour method work in three simple phases:
Users of a recommender system rate LOs that they have previously experienced.
The recommender then matches the user with other participants of the system who have similar rating patterns (i.e. they have similar opinions on experienced LOs.) This is usually done through statistical correlation. The closest matches are selected, becoming known as neighbours of the user, or collectively as the neighbourhood.
LOs that the neighbours have experienced and rated highly, but which the user has not yet experienced, will be recommended to the user, ranked based on the closeness of the neighbours to the user and the consistency of opinion within the neighbourhood.
From an algorithmic point of view, it is convenient to classify the recommender filtering algorithms in three classes depending on their query and update costs (Lemire 2004): learning-free, memory-based and model-based. Obviously, there might be many types of operations that could be described as an update or a query, but we focus our attention on adding a user and its ratings to a database (update) or asking for a prediction of all ratings for a given user (query). We say that an operation whose complexity is independent of the number of users offers constant-time performance (with respect to the number of users). Essentially, the cheapest schemes are described as learning-free and have both constant-time updates and queries while schemes involving a comparison with users in the database are classified as memory-based and offer constant-time updates but linear-time queries, and finally the schemes requiring more than linear time learning or more sophisticated updates are said to be model-based. For purpose of this article, we are proposing a modified memory-based algorithm. The traditional memory-based algorithm requires us to go through a large set of preferences each time a prediction is required. This task can quickly become expensive: doubling the number of users, roughly doubles the response time of the system(Anderson et. al 2003). Ideally, one would want on-line constant time answers while using only a marginal amount of resources. As a more scalable alternative, we are proposing the use of the Bias From Mean algorithm (Lemire 2004).

Conclusions

This article presented a flexible mixture model for searching and recommending Java learning objects. The model enforces a collaborative infrastructure for authoring, searching, recommending and presenting Java source code learning objects. The new model uses two specialized filtering engines which work simultaneously: CollabroSearch and CollabroRecommender to present relevant LOs from presented queries or from the mined text collected from the collaborative chatting channel between users. Experiments on Java source code testbed indicated that the proposed model is able to outperform any primitive collaborative environment that support some searching and discovery primitives such as the I-help system developed by ARIES Lab, Department of Computer Science, University of Saskatchewan (http://www.cs.usask.ca:7777/ihelp11/entrance.html) and any of the commercial collaborative environments such as JCE, Centra99, PaceWare, Grooves and Tango. The combination methods of the flexible mixture model (ReccoSearch) is rather preliminary. As a near future work, we

Advanced Research & Studies Journal

plan to explore approximate searching techniques based on query expansion (Fiaidhi, Mohammed, Jaam, Hasnah 2003) as well as to explore different types of recommending algorithms (besides the k-nearest collaborative filtering that we originally used) such as the Item-Basedrecommending algorithm (Sarwar et al, 2001). The ultimate future goal of this research is to link the RecoSearch system to the POOL of LOs repositories(Hatala and Richards 2002) via utilizing the JXTA APIs (www.jxta.org/) to unable collaborators to search and recommend other LOs from the major repositories available to the academia.

Acknowledgments

This research is supported in part by my CFI grant to establish the new Lakehead University Virtual Learning and Training Centre. This is also the research topic for my two graduate students (S. Sisko and T. Song) whom they are implementing and experimenting on different aspects of this article.

References

Anderson Michelle et. al, RACOFI: A Rule-Applying Collaborative Filtering System, Workshop on Collaboration Agents: Autonomous Agents for Collaborative Environments, December 13, 2003 - Halifax Canada.

Dodani M., The Dark Side of Object Learning: Learning Objects, in Journal of Object Technology, vol. 1, no. 5, November-December 2002, pp. 37-42.

Fiaidhi J., Mohammed S., Jaam J., and Hasnah A., A Standard Framework for Search Hosting via Ontology Based Query Expansion, The SCI03 Multi-Conference (Orlando, Florida, USA, July 27 - 30) – AITOCP03 Session, 2003.

Fiaidhi J., Mohammed S., Design Issues Involved in Using Learning Objects for Teaching a Programming Language within a Collaborative eLearning Environment, International Journal of Instructional Technology and Distance Learning (USA), Vol. 1, No. 3., March 2004, pp.39-53.

Fiaidhi J., Mohammed S., Developing a Search engine for Learning Objects, Asian Journal of Information Technology, Vol. 3, No. 7, July 2004a.

Fiaidhi J., Passi K., and Mohammed S., Developing a Framework for Learning Objects Search Engine, The 2004 International Conference on Internet Computing (IC04), Las Vegas, Nevada, USA, June 21-24, 2004.

Fiaidhi J., Mohammed S., S. Sisko, SceneBeans: A Tool for Constructing Collaborative Multimedia Learning Objects, 9th Western Canadian Conference on Computing Education (WCCE04), Kelowna, BC, Canada, May 6-7, 2004. (Refereed- Regular Research Paper)

Friesen N., Roberts A. and Fisher S., Canadian Journal of Learning and Technology Volume 28(3) Fall / autumn, 2002 CanCore: Metadata for Learning Objects

Gaaster T., Cooperative Answering through Controlled Query Relaxation, IEEE Intelligent Systems, Vol. 12, No. 5, 1997.

Goldberg D., Nichols D., Oki B.M., Terry and D., Using collaborative filtering to weave an information tapestry. Communications of the ACM, 35(12):61{70, 1992.

Hatala M. and Richards G., POOL, POND and SPLASH: A Canadian Infrastructure for Learning Object Repositories. Paper presented at the 5th IASTED International Multi-Conference Computers and Advanced Technology in Education (CATE 2002). Cancun, Mexico, 2002.

Hatala, M., Richards, G., Eap, T., Willms, J. The Interoperability of Learning Object Repositories and Services: Standards, Implementations and Lessons Learned. ACM WWW2004 (Education Track) Conference, May 17-22,2004, New York, USA.

Hayes C., Massa P., Avesani P., Cunningham P., An on-line evaluation framework for recommender systems, In Proceedings of the Workshop on Personalization and Recommendation in E-Commerce, Malaga, 2002.

Klein M., Combining and relating ontologies: an analysis of problems and solutions, In. proceedings of the 17th International Joint Conference on Artificial Intelligence IJCAI-01 Workshop: Ontologies and Information Sharing, (Seattle, USA), 2001.

Advanced Research & Studies Journal

Krumel Andy, Jato: The New Kid on the Open Source Block, Part 1," (JavaWorld), March 16, 2001.
Lemire D., Daniel, Scale And Translation Invariant Collaborative Filtering Systems. Information Retrieval, Vol. 7, pages 1 – 22, 2004. (NRC 46508)
McLaughlin B. , Java and XML Binding, O'Reilly Publications, May 2002, ISBN: 0-596-00278-5
Mowbray, T. J. and Zahavi R.. (1995). The ESSENTIAL CORBA: System Integration Using Distributed Objects.: John Wiley and Object Management Group.
Rahm E., and Bernstein P. A., On Matching Schemas Automatically. MSR Tech. Report MSR-TR-2001-17, 2001, http://www.research.microsoft.com/pubs, 2001.
Recker M. and Wiley D., A Non-authoritative Educational Metadata Ontology for Filtering and Recommending Learning Objects, Interactive Learning Environments 2001, Vol.9, No.3, pp. 255-271
Resnick P. and Varian H.R., Recommender systems. Communications of the ACM, 40(3):56{58, 1997.
Rizzolo F. and Mendelzon A., Indexing XML Data with ToXin, Fourth International Workshop on the Web and Databases, (Santa Barbara, CA.), 2001.
Passi K., Lane L., Madria S., Sakamuri B. C., Mohania M., and Bhowmick S. S., A Model for XML Schema Integration. EC-Web 2002: 193-202.
PCAI Magazine, Distributed Indexed Searching: Evolution to XML, July/August 2001, Vol. 15, No. 4.
Puustjarvi J. and Poyry P., Searching Learning Objects from virtual Universties, IEEE Int. workshop on Multimedia Technologies in e-Learning and Collaboration (WOMTEC03), December 17, 2003, Nice, France.
Sarwar B. et al, Item-Based Collaborative Filtering Recommendation Algorithms, ACM WWW10 Conference, Hong Kong, May 1-5 2001.
Simic H. and Topolnik M., Prospects of encoding Java source code in XMLA paper published on ConTel 2003: 7th International Conference on Telecommunications, June 11-13 2003, Zagreb, Croatia.
WEBDAV: IETF Standard for Collaborative Authoring on the Web", by Jim Whitehead and Meredith Wiggins, which appeared in the September/December, 1998 issue of IEEE Internet Computing, pages 34-40.
World Wide Web Consortium, XML Schema Part 1: Structures, May 2001, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502
World Wide Web Consortium, XML Schema Part 2: Datatypes, May 2001, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502