

# A Match-Making System for Learners and Learning Objects

Harold Boley<sup>1</sup>, Virendrakumar C. Bhavsar<sup>2</sup>, David Hirtle<sup>2</sup>,  
Anurag Singh<sup>2</sup>, Zhongwei Sun<sup>2</sup>, and Lu Yang<sup>2</sup>

<sup>1</sup> *Institute for Information Technology - e-Business, National Research Council of Canada (NRC)  
46 Dineen Drive, Fredericton, NB E3B 9W4, Canada  
harold.bole AT nrc-cnrc.gc.ca*

<sup>2</sup> *Faculty of Computer Science, University of New Brunswick, P.O. Box 4400, 540 Windsor Street  
Gillin Hall, Fredericton, NB, E3B 5A3, Canada  
{bhavsar, david.hirtle, anurag.singh, e28a1, o6c11} AT unb.ca*

## Abstract

We have proposed and implemented AgentMatcher, an architecture for match-making in e-Business applications. It uses arc-labeled and arc-weighted trees to match buyers and sellers via our novel similarity algorithm. This paper adapts the architecture for match-making between learners and learning objects (LOs). It uses the Canadian Learning Object Metadata (CanLOM) repository of the eduSource e-Learning project. Through AgentMatcher's new indexing component, known as Learning Object Metadata Generator (LOMGen), metadata is extracted from HTML LOs for use in CanLOM. LOMGen semi-automatically generates the LO metadata by combining a word frequency count and dictionary lookup. A subset of these metadata terms can be selected from a query interface, which permits adjustment of weights that express user preferences. Web-based prefiltering is then performed over the CanLOM metadata kept in a relational database. Using an XSLT (Extensible Stylesheet Language Transformations) translator, the prefiltered result is transformed into an XML representation, called Weighted Object-Oriented (WOO) RuleML (Rule Markup Language). This is compared to the WOO RuleML representation obtained from the query interface by AgentMatcher's core Similarity Engine. The final result is presented as a ranked LO list with a user-specified threshold.

Keywords: AgentMatcher, CanLOM, e-Business, e-Learning, Learning Objects, match-making, metadata, metadata generator, RuleML

## 1. INTRODUCTION

We have developed the AgentMatcher system [Sarno et al. 2003] for match-making between buyer and seller agents. The present paper describes the application of this system for searching procurable learning objects (LOs) in an e-Learning environment. Keywords and keyphrases are often used to describe LOs as well as learner queries in such environments. However, such a flat representation does not lend itself to hierarchical LO matching enabled by the Learning Object Metadata (LOM) standard and is also not likely to reflect user preferences about the relative importance or weighting of

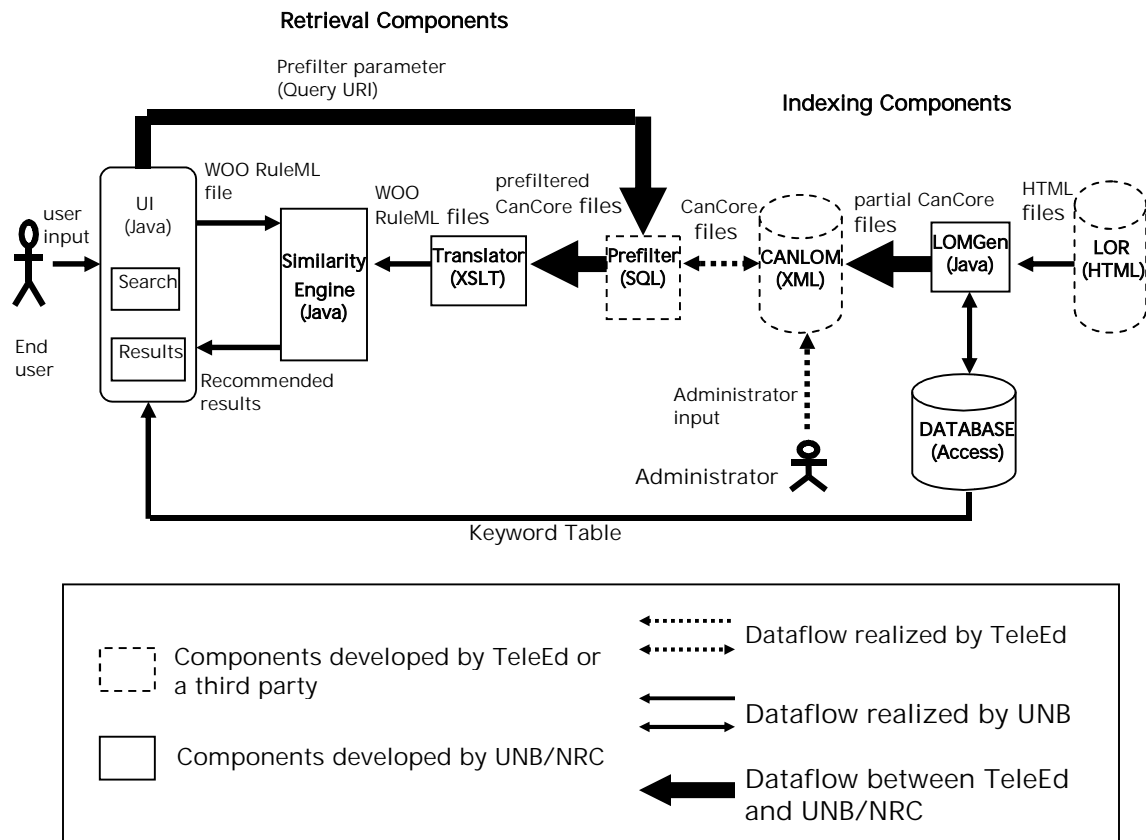
the parts of a LOM description. AgentMatcher takes into account both of these dimensions — hierarchical matching and differential weighting — via tree-structured descriptions with arc weights for the queries, enhancing the precision of LO retrieval.

In this paper we describe the Java-based AgentMatcher match-making architecture as applied to the XML-based Canadian Learning Object Metadata (CanLOM) repository of the Canadian eduSource project [eduSource 2004]. The CanLOM repository is built using the LOM standard specified by the CanCore [CanCore 2004] guidelines. The Learning Object Metadata Generator (LOMGen) extracts CanCore metadata from HTML LOs in the domain of ‘Computing’, speeding up the process of metadata generation [Singh et al. 2004]. LOMGen-extracted terms are offered to learners for selection from a query interface that permits convenient entry of relevant tree components and weights. Web-based prefiltering is then performed over the CanLOM metadata kept in the relational database of the KnowledgeAgora e-Learning repository of TeleEducation New Brunswick (TeleEd). The prefiltered result is transformed to Weighted Object-Oriented (WOO) RuleML [Boley 2003] via an XML-to-XML translator. Finally, this is compared to the WOO RuleML-serialised tree obtained from the query interface using the weighted tree similarity algorithm [Bhavsar et al. 2004] embedded in the AgentMatcher Similarity Engine, and a percentage-ranked LO list is presented to the learner.

## 2. OVERVIEW

The AgentMatcher architecture can be applied to match-making [Sycara et al. 2001] between *buyer* and *seller* agents in e-Business, e-Learning and other environments. The core engine of AgentMatcher performs similarity computation between metadata descriptions carried by buyer and seller agents. In the AgentMatcher instantiation for e-Learning, "buyers" are learners and "sellers" are learning object (LO) providers. We use the guidelines specified by CanCore to describe LOs. Thus, the match-making between buyer and seller agents corresponds to the matching of learner queries and CanCore descriptions.

The architecture of the AgentMatcher as adapted to e-Learning is depicted in Fig. 1, showing the top-level retrieval and indexing components.



**Figure 1.** The AgentMatcher architecture.

There are three retrieval components: the *User Interface*, *Similarity Engine* and *Translator*. The LOM Generator (LOMGen) in parallel performs indexing to support retrieval. Each of these four major components of the system is detailed in the ensuing sections.

### 3. USER INTERFACE

The user interface permits a user to enter as well as assign weights to search parameters and retrieve ranked search results in a new browser window.

**Advanced Search**

Please use the sliders to represent the importance of each choice.

Please hold "ctrl" key to select multiple keywords. Your input is required for keywords.

**General**

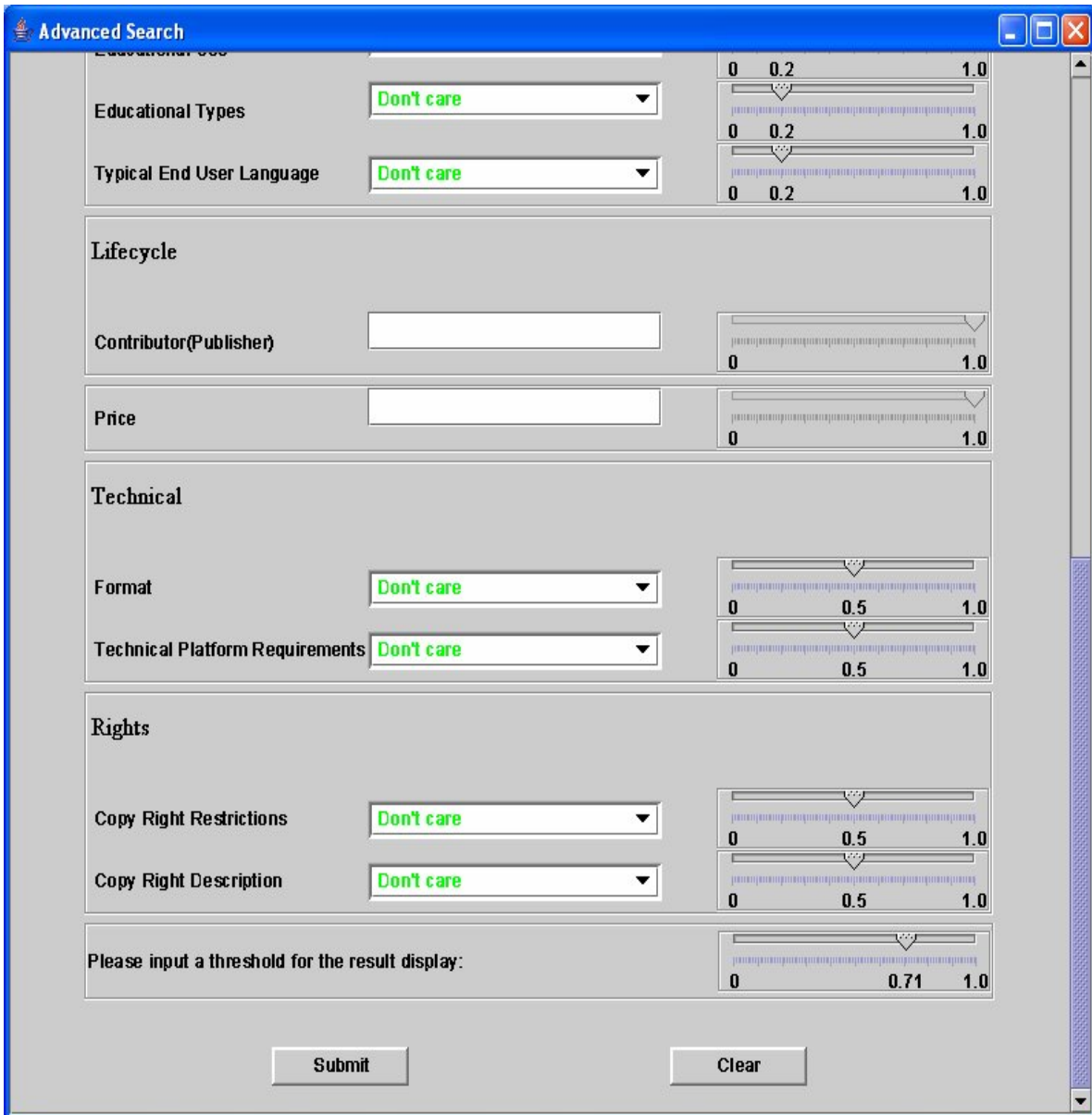
Title	<input type="text"/>	<input type="range" value="0.33"/>
Keywords	<ul style="list-style-type: none"><li>ANSI</li><li>artificial intelligence</li><li>bayesian filter</li><li>C Programming Language</li></ul>	<input type="range" value="0.33"/>
Language	<input type="text" value="en"/>	<input type="range" value="0.33"/>

**Classification**

Discipline	<input type="text" value="Don't care"/>	<input type="range" value="0"/>
------------	---	---------------------------------

**Educational**

Intended End User	<input type="text" value="Learner"/>	<input type="range" value="0.2"/>
Typical Age Range	<input type="text" value="18-25"/>	<input type="range" value="0.2"/>
Educational Use	<input type="text" value="Don't care"/>	<input type="range" value="0.2"/>
Educational Types	<input type="text" value="Don't care"/>	<input type="range" value="0.2"/>
Typical End User Language	<input type="text" value="Don't care"/>	<input type="range" value="0.2"/>



**Figure 2.** Search screen of the user interface.

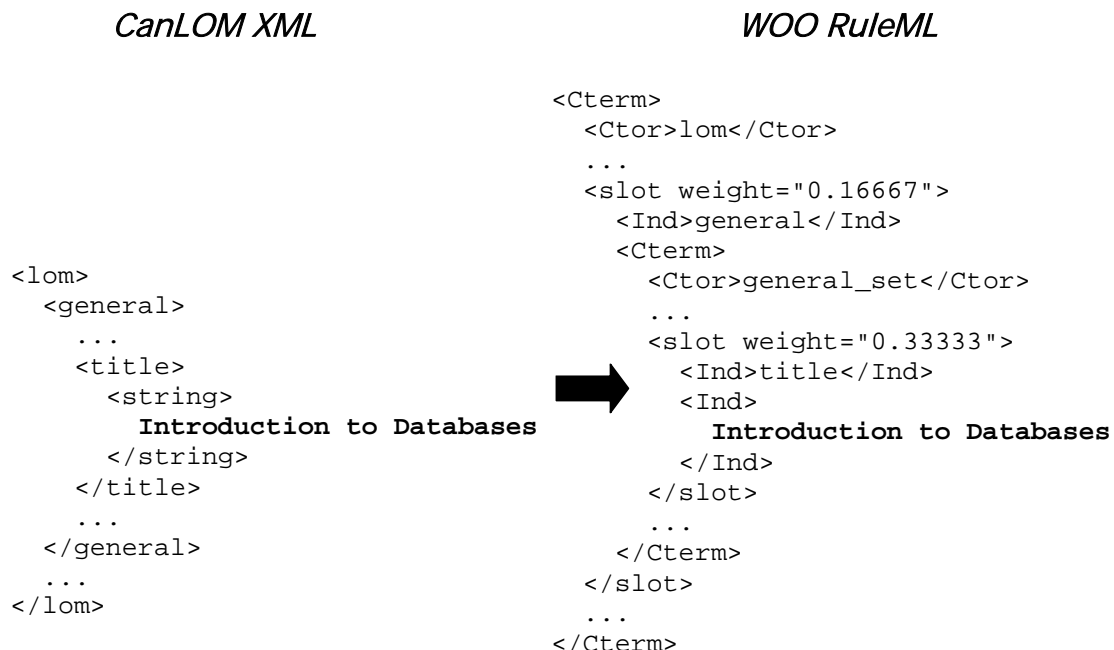
As shown in Fig. 2, the search screen is split into multiple boxes reflecting the top-level branches ('General', 'Classification', etc.) of the query tree structure; each of these boxes contains one or more search parameters chosen from the same category as found in the CanCore schema. Accompanying each search parameter is a slider, permitting the user to input not only the parameter but also its corresponding weight. This weight indicates the importance of a parameter to the user relative to other parameters within the same category. All the weights within one box are constrained to add up to 1.0. The user is also able to input a threshold for the search result recommendations, causing all the LOs with a similarity value above the threshold to be considered as the recommendations.

After the user submits the advanced search request, the internal functions will be invoked according to the dataflow in Fig. 1. First of all, a Weighted Object-Oriented RuleML (WOO RuleML) parameter file (hereafter referred to as `user.xml`) is generated by the user interface. WOO RuleML is the format required by the Similarity Engine. Then selected search parameters are sent via a query URI to the KnowledgeAgora database server for pre-filtering, using the database query functionality to select relevant LOs by examining their Learning Object Metadata (LOM). The response from KnowledgeAgora is parsed into multiple XML files. These files are translated by the Translator into WOO RuleML files and passed to the Similarity Engine. At this point, the `user.xml` file is compared with each of the LOM files translated into WOO RuleML. The final result of the similarity computations is then displayed as a list of LOs ranked according to their similarity to the original search parameters entered by the user (see Section 5). Only those LOs with similarities above the threshold are recommended to the user.

#### 4. TRANSLATOR

The translator is responsible for translating the pre-filtered LOM files from the CanLOM repository into Weighted Object-Oriented RuleML, required by the Similarity Engine. It defaults LOM weights to equal values (up to rounding) on all tree levels, since this e-Learning application of AgentMatcher uses proper weights only for the query trees.

The (abbreviated) sample illustrated in Fig. 3 demonstrates the mapping between the two formats. Extensible Stylesheet Language Transformations (XSLT), a W3C recommended language for transforming XML documents into other XML documents, is used to accomplish this mapping. Additional information about this translation process is available in a separate report [Hirtle and Sun 2003]. When translation is complete, the resulting WOO RuleML files are passed to the Similarity Engine for comparison to the WOO RuleML representation of the search parameters specified by the user.

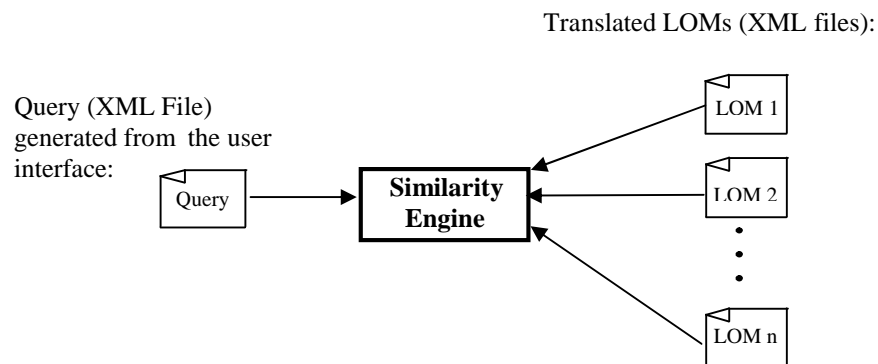


**Figure 3.** Mapping from CanLOM XML to WOO RuleML.

## 5. SIMILARITY ENGINE

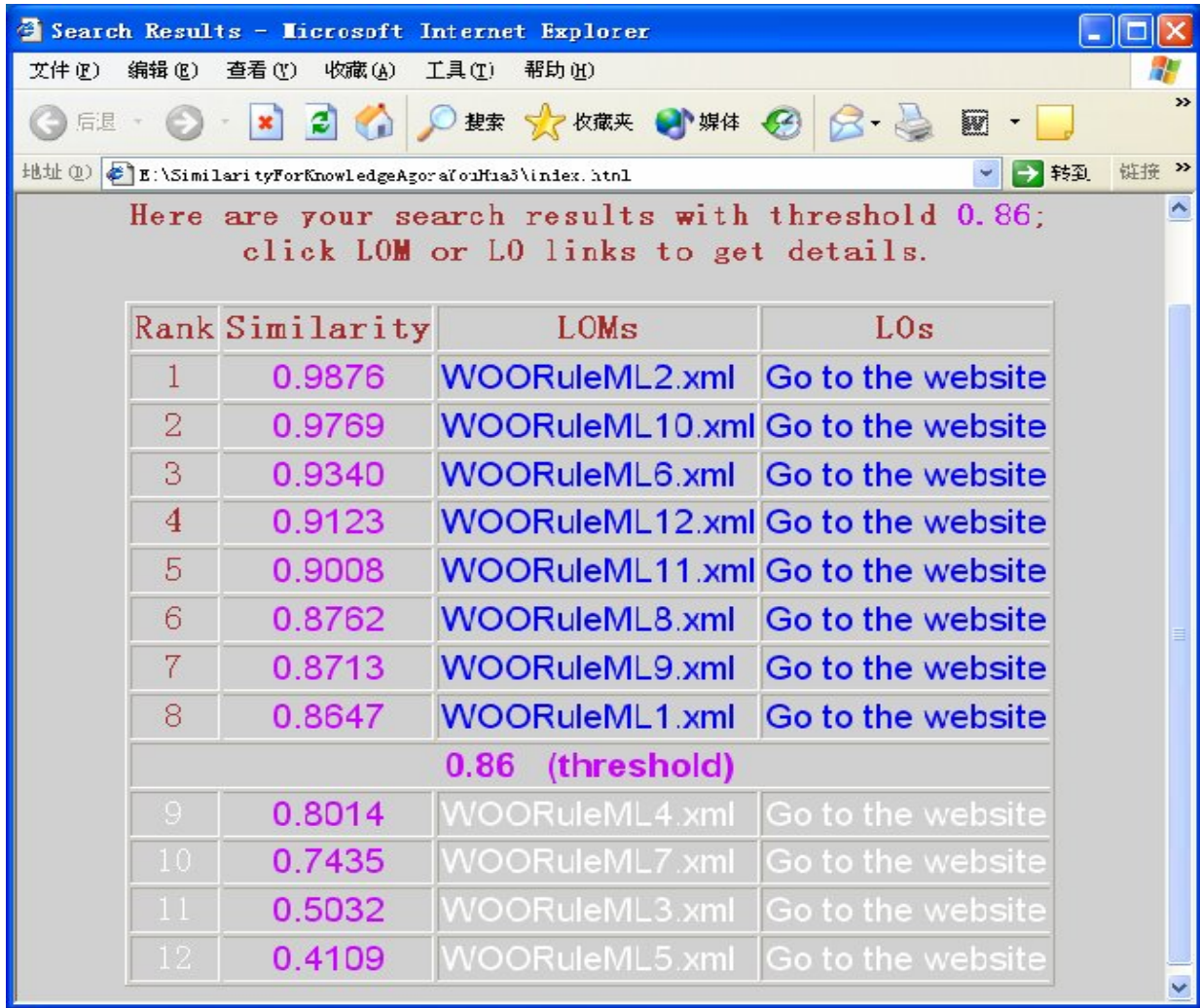
The Similarity Engine is responsible for computing the similarity between the query file and prefiltered LOM files using our tree similarity algorithm [Bhavsar et al. 2004]. It constructs a ranked list of search results and displays it in a browser window.

As shown in Fig. 4, the inputs of the Similarity Engine are the query file `user.xml` generated from the user interface (as discussed in Section 3) and the translated LOM files (as discussed in Section 4). We use our tree similarity algorithm, embedded in the Similarity Engine, to compute, one by one, similarity values between the query and each LOM. Due to our unique tree representation for learners and learning objects, our tree similarity algorithm is quite different from previous work [Liu and Geiger 1999] [Wang et al. 1998]. These similarity values are constrained to the real interval [0.0, 1.0].



**Figure 4.** Inputs of the Similarity Engine.

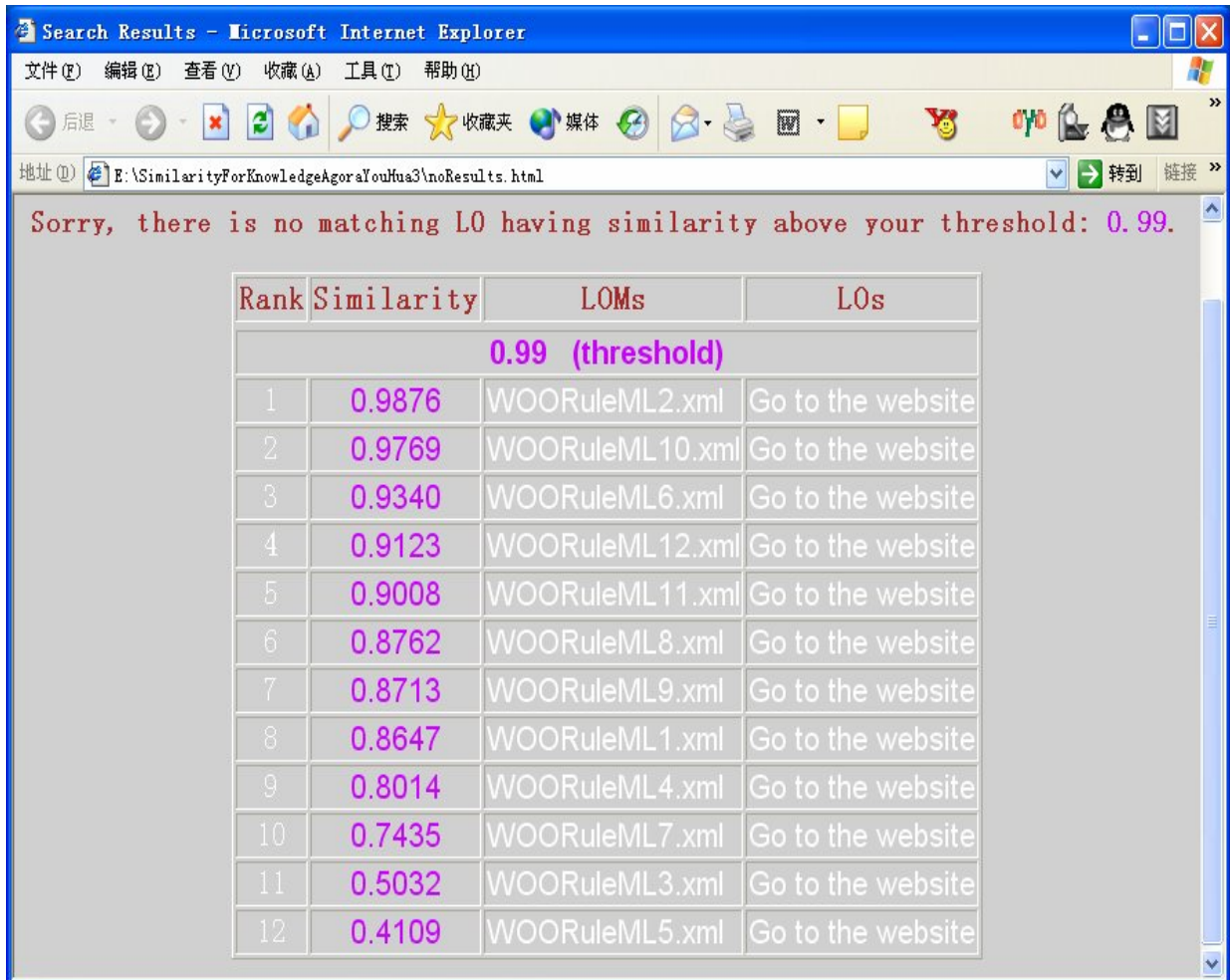
After computing the similarity between the query and LOMs, the Similarity Engine ranks all of the LOs in descending order of similarity, graphically separating those results whose similarity values fall below the threshold. The user will find on the top of the list the LOM that has the highest similarity value with his/her query.



**Figure 5.** Snapshot of search results (low threshold).

Fig. 5 shows the HTML output for a relatively low threshold of 0.86. There are four columns in the result table: Rank, Similarity, LOMs and LOs. The rank represents the descending similarity order of the LOs, where highest rank corresponds to the highest similarity value. The similarity values are displayed in the second column. The LOMs and LOs are shown in the final two columns; clicking the link of a LOM record (e.g. WOORuleML10.xml) displays the metadata (in XML format) corresponding to the LO. The “Go to the website” links in the last column point to organizations’ websites that give the content of LOs.

Besides showing the search results above the threshold, we also show those that are below the threshold in case some users want to see more LOMs and LOs. Links for these results are displayed in white.



**Figure 6.** Snapshot of search results (high threshold).

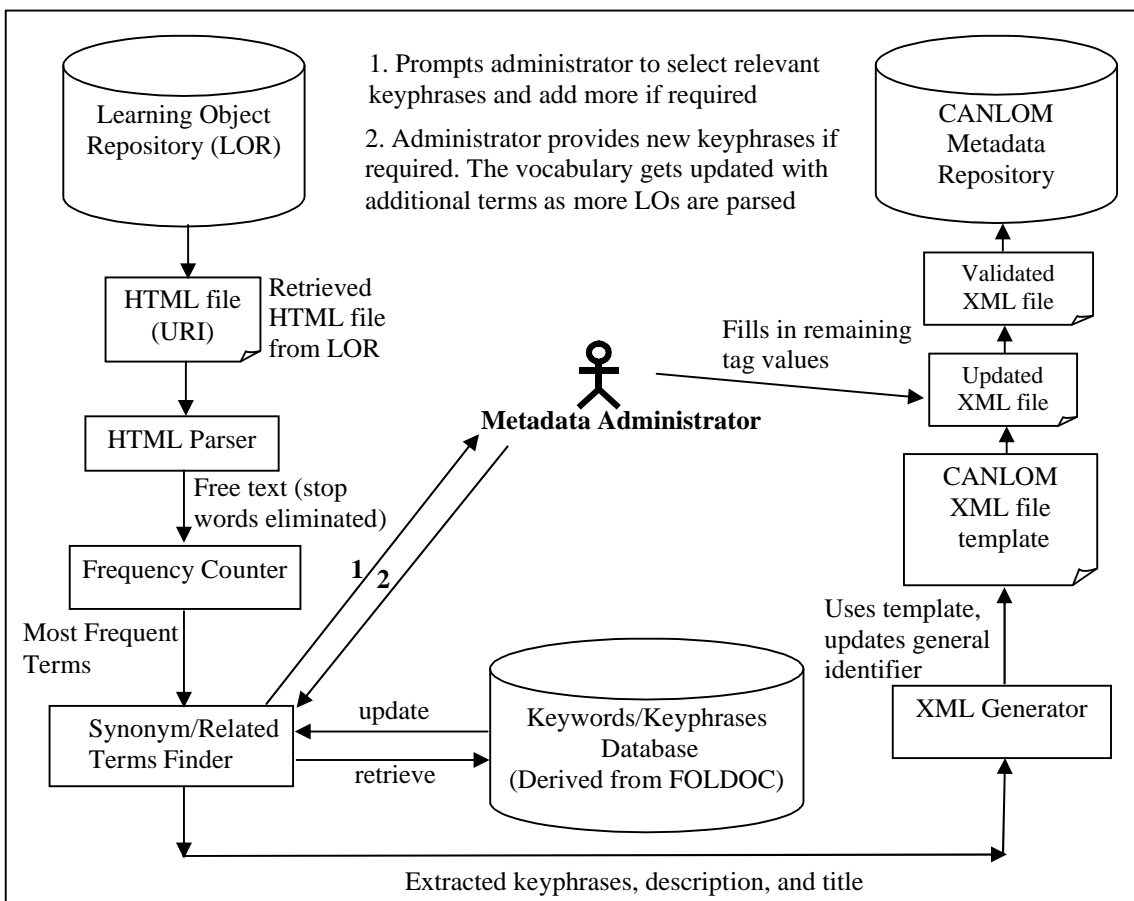
Sometimes a user may input a similarity threshold that is too high, resulting in a failed search. In this case, we do not direct users back to the search screen to adjust the threshold, but instead give users warning that their threshold is too high and still show all the search results that are below the threshold. Fig. 6 shows the search results in such a situation. Of course, if users want to change inputs (e.g., keywords), they have to go back to the interface to input again.

## 6. LOM GENERATOR (LOMGen)

The process of manually entering metadata to describe an LO is a time-consuming process. It generally requires the metadata administrator/author to be intimately familiar with the LO content. A semi-automated process which extracts information from the LO can alleviate the difficulties associated with this time-consuming process. LOMGen aims at automating the metadata extraction process with minimal user intervention.

LOMGen works with LOs constituted as HTML files. LOMGen uses the Free Online Dictionary of Computing (FOLDOC) to generate keywords and keyphrases from an LO. The use of FOLDOC currently restricts LOMGen applicability to LOs in the domain of 'Computing'.

As shown in Fig. 7, the LOMGen architecture consists mainly of an HTML file reader module (which reads an LO file from a URI), an HTML parser, a word frequency counter, a database interface module, and an XML file writer (which updates the metadata repository with a newly generated LOM file).



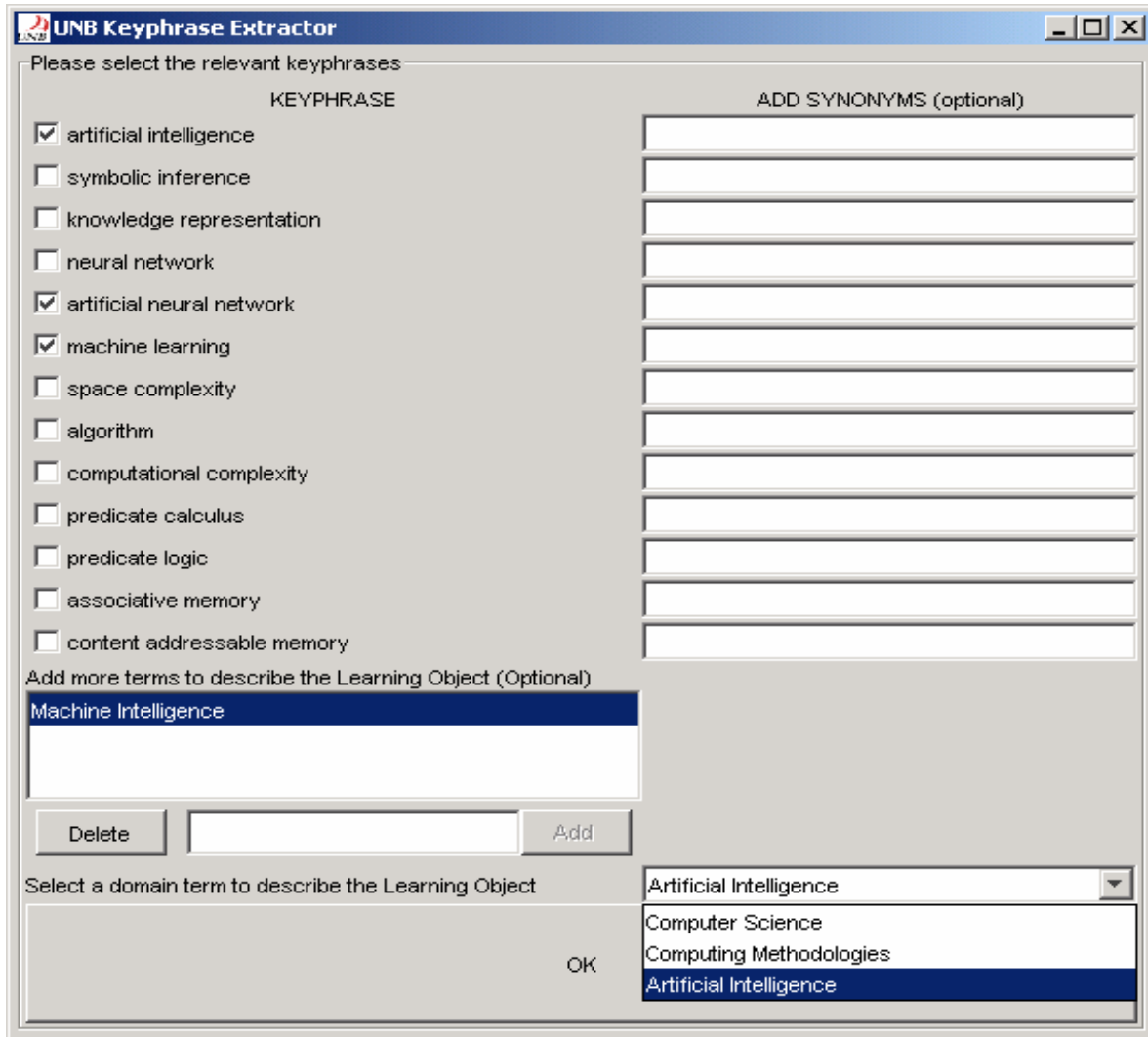
**Figure 7.** LOMGen architecture.

LOMGen obtains the most frequent words and phrases from the content of the LO. In addition, FOLDOC is referenced to find terms related to 'Computing' present in the LO. In order to get relevant results, frequently occurring stop words such as "is", "are", "the", and "in" are eliminated. The result is combined with information found in the HTML "meta" tags and additional keyphrases (which may not be present in the LO) are generated with the help of FOLDOC. These keyphrases are obtained by looking up synonyms and related terms for words or phrases. All these words and phrases are presented to the metadata administrator through the LOMGen Graphical User Interface (GUI) for keyphrase selection as well as synonym and term addition. The updates made by the administrator are stored in the keywords/keyphrases database. The newly added terms are considered to provide better choices to the metadata administrator when processing similar LOs.

A snapshot of the GUI presented to the metadata administrator is shown in Fig. 8. The GUI presents a list of keywords and keyphrases that were extracted or derived from the LO. The checkboxes present under the title "KEYPHRASE" allow the metadata administrator to select the most important keywords or keyphrases. The textboxes under "ADD SYNONYMS" allow the administrator to add alternate but similar terms corresponding to the keyphrase on the left. As the metadata administrator selects keyphrases in the GUI, the domain term dropdown listbox gets populated with the domains for those choices. These domain terms are obtained from FOLDOC. The metadata administrator selects the most relevant domain and FOLDOC is used to derive a hierarchy for classifying the LO.

If an LO lacks sufficient information in the text and HTML metatags, the quality of the keywords or keyphrases extracted by LOMGen may not be satisfactory. In such a scenario, the GUI enables the administrator to add more terms explicitly to describe the LO.

Finally, clicking the "OK" button generates a LOM file and posts it to the CanLOM repository through a standard interface for posting XML files (provided by CanLOM).



**Figure 8.** GUI for keyphrase selection.

The LOMGen component can be used as a training module for a text summarizer that uses machine learning techniques, with the intention of eliminating many of the administrator inputs.

## 7. CONCLUSION AND FUTURE WORK

The AgentMatcher match-making system is applied to e-Learning, where learners are in search of procurable LOs. The resulting Java-based architecture takes advantage of the added expressiveness obtained from tree-based matching and user-assigned weights. CanCore metadata is extracted from HTML LOs by our LOMGen indexer, speeding up the task of metadata generation. The metadata is first prefiltered via a query URI, and then transformed to Weighted Object-Oriented RuleML via an XSLT translator. The results are then compared to another tree representation of the learner query as generated by the user interface. Finally, a list of learning objects is presented to the learner in

descending order of similarity, computed by the weighted tree similarity algorithm. This application of AgentMatcher, restricted to the ‘Computing’ domain in this project, demonstrates enhanced precision achievable relative to standard keyword-based searches.

Generally, we showed that the AgentMatcher architecture can be easily instantiated for e-Learning applications, where match-making between buyers and sellers in e-Business is transferred to learners and learning object providers, respectively. The system is available online via the page [www.cs.unb.ca/agentmatcher](http://www.cs.unb.ca/agentmatcher). AgentMatcher has also been adapted to match-making in another domain, namely technology transfer wherein buyers and sellers can be venture capitalists and startups (visit the [www.teclantic.ca](http://www.teclantic.ca) portal for details).

In future, the tree similarity algorithm embedded in the Similarity Engine can be enhanced, e.g. by adding local similarity measures. Our pairing algorithm [Sarno et al. 2003] can be modified to pair learners and learning objects. The user interface can also be improved. The LOMGen indexing module can be enhanced by natural language processing techniques for syntactic and semantic analysis of LOs; these techniques are expected to improve the quality of the metadata generated and further automate the metadata extraction process.

### **Acknowledgements**

We thank the CANARIE eduSource Project and NSERC as well as the New Brunswick Department of Education for their support. We also appreciate valuable comments from referees.

### **References**

Bhavsar, V.C., H. Boley, L. Yang, 2003, “A Weighted-Tree Similarity Algorithm for Multi-Agent Systems in E-Business Environments”, *In Proceedings of 2003 Workshop on Business Agents and the Semantic Web*, Halifax, June 14, 2003, National Research Council of Canada, Institute for Information Technology, Fredericton, pp. 53-72, 2003. Revised version appears in *Computational Intelligence*, 20(4), pp. 584-602.

Boley, H., 2003, Object-Oriented RuleML: User-Level Roles, URI-Grounded Clauses and Order-Sorted Terms. In Schroeder, M. Wagner, G. (Eds.): *Rules and Rule Markup Languages for the Semantic, Web* Springer-Verlag, Heidelberg, LNCS-2876, pp. 1-16.

CanCore home page. (2003, April 23): Canadian Core Learning Resource Metadata Application Profile. Retrieved April 25, 2003, from <http://www.cancore.ca>

eduSource Canada. (no date): Canadian Network of Learning Object Repositories. Retrieved March 29, 2003, from <http://www.edusource.ca>

Hirtle, D. and Z. Sun, 2003, “CanCore ⇔ WOO RuleML”, Internal Report, Faculty of Computer Science, University of New Brunswick, [www.cs.unb.ca/agentmatcher/translators](http://www.cs.unb.ca/agentmatcher/translators).

T-L. Liu and D. Geiger, 1999, "Approximate Tree Matching and Shape Similarity," In Proceedings, 7th International Conference on Computer Vision, pp. 456-462, Kerkyra, Greece, 1999.

Sarno, R., L. Yang, V.C. Bhavsar and H. Boley, 2003, "The AgentMatcher architecture applied to power grid transactions", *In Proceedings of the First International Workshop on Knowledge Grid and Grid Intelligence*, Halifax, Canada, pp. 92-99.

Singh, A., H. Boley and V.C. Bhavsar, 2004, "A Learning Object Metadata Generator Applied to Computer Science Terminology," Presented at eduSource Learning Objects Summit, National Research Council of Canada, Fredericton, March 29-30, 2004.

Sycara, K., M. Paolucci, M. van Velsen, and J. A. Giampapa, 2001, The RETSINA MAS infrastructure. Robotics Institute, Carnegie Mellon University, CMU-RI-TR-01-05.

Wang, J. T., B. A. Shapiro, D. Shasha, K. Zhang and K. M. Currey, 1998, "An Algorithm for Finding the Largest Approximately Common Substructures of Two Trees", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8): 889-895.