Focused Crawling for Downloading Learning Objects – An Architectural Perspective

Yevgen Biletskiy, Michael Wojcenovic, and Hamidreza Baghi
University of New Brunswick, Canada

biletski@unb.ca; q0mim@unb.ca; hamid.baghi@unb.ca

Abstract

The present paper describes architecture, design, and implementation details of a Web Crawler for learning objects. The Web Crawler developed provides a technical solution to mass downloading learning objects from digital libraries on the Web for further search and delivery of appropriate learning objects to learners using various e-Learning services (e.g. personalization). The task of mass downloading learning objects, which would be long if done manually, can be performed quickly using the present system. Throughout the development of the system, the main objective was to decrease the time and user interaction needed to obtain the desired information from the Web repositories containing learning objects. The present system implements features of finding, parsing, and downloading web pages with learning object metadata and content. The system has been developed with reusability in mind, so this system can be used as the basis for any other task related to downloading XML-based documents from the Web. The final result of the presented work is a functioning system that meets the objectives and requirements given.

Keywords: e-Learning, Learning Object, Learning Object Repository, Learning Object Metadata, Learning Object Interoperability, Personalization

Introduction

Searching and delivering the appropriate learning objects to learners are challenging tasks of e-Learning, because: firstly, the learning object content must fit in the learning field and learning objectives; secondly, the learning object's content must be provided to an acceptable level of the learner's understanding; and finally, the learning object as an educational unit must fit well in the design of a course or a learning program. At the present time, there are many approaches and techniques developed within e-Learning initiatives that facilitate the search and delivery of appropriate learning objects to learners, for example, concept-based search, context-sensitive delivery and personalization, ontology-based course assembly and learning content development, adaptive learning and adaptive media, etc. (Aroyo et al., 2006; Baghi, Biletskiy, & Li, 2008; Biletskiy, Vorochek, & Medovoy, 2004; Biletskiy, Baghi, Keleberda, & Fleming, 2008; Biletskiy, Brown, & Ranganathan, 2009; Boley et al., 2005; Devedzic, 2004; Friezen, 2005; Jovanovic, Gašević, Knight, & Richards, 2007; Keleberda, Repka, & Biletskiy, 2006; Pahl & Hollogan, 2009; Wang, Tsai, Lee, & Chiu, 2007; Zouaq, Nkambou, & Frasson, 2007). Usually, these techniques involve a vast number of learning objects accompanied by learning object metadata (LOM) distributed on the Web, in digital

Editor: William Hoseel

libraries and learning object repositories, such as NEEDS (2008) and SMETE (2008). The digital library services provide search capabilities, but there is no way to automatically access learning object metadata (LOM) without manually downloading learning objects from the digital libraries. The number of learning objects usually needed would be impossible to manually download in a reasonable amount of time. Therefore, a technical solution needed to be developed to perform the job of downloading the learning objects. The solution to this problem is a focused web crawler.

Web crawlers are used to automate the job of traversing through web pages for a specific purpose (Blum, Keislar, Wheaton, & Wold, 1998). The specific purpose of the presented web crawler is to traverse web pages to find and download learning objects, namely their LOM, which references learning object contents in a repository on the Web. The main objective of this web crawler is to facilitate a long and tedious process of downloading learning objects and make this process possible within minutes and with minimum action from the user.

In the context of personalized search and delivery of learning objects, the crawler can be used as depicted in Figure 1. More specifically, the crawler accesses a learning object repository on the Web (LOR Web), parses web pages containing LOM (LOM(XML)) according to some predetermined patterns (Web LOR URL Patterns; described later) and downloads a pre-determined number of LOM files to a local repository (LOR Local). The LOM contains references (URL) to files with actual learning object content. LOM as well as the content can then be used by a specific learning object search (Search Engine) and delivery service (e.g. personalization, which delivers learning objects to learners according to the Learner Profile explicitly or implicitly provided by the Learner). Storing LOM in a local repository decreases the time for search and delivery of learning objects to learners. The next sections of the paper describe the top-level architecture of the crawler, implementation details, which are useful for re-use of the crawler, evaluation of performance and concluding remarks.

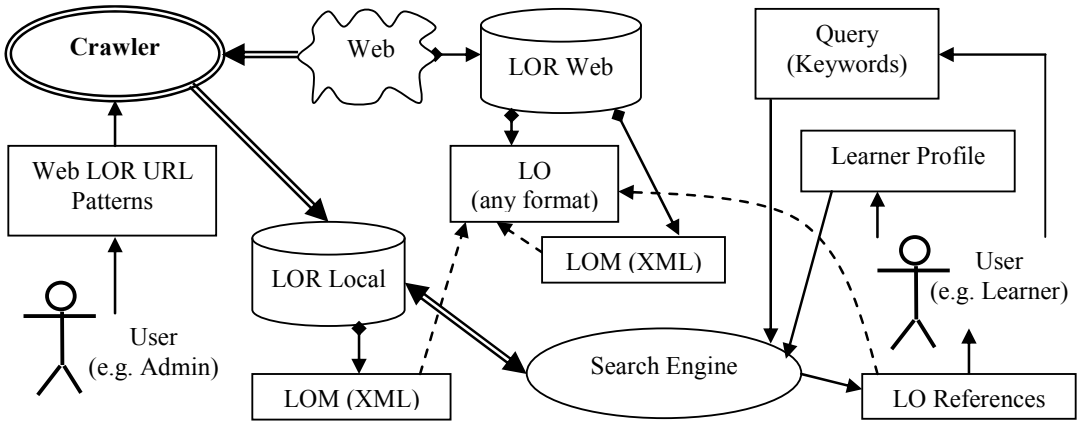


Figure 1: Web Crawler for search and delivery of learning objects.

The Focused Web Crawler System Architecture

The Focused Web Crawler system consists of two main applications (Figure 2), where the information from the first application is passed to the second one. The first application, ID Web Crawler, accesses the Web to get page information, parses web page information and finds the Identifiers (IDs) that correspond to learning objects. The IDs in the present work are obtained

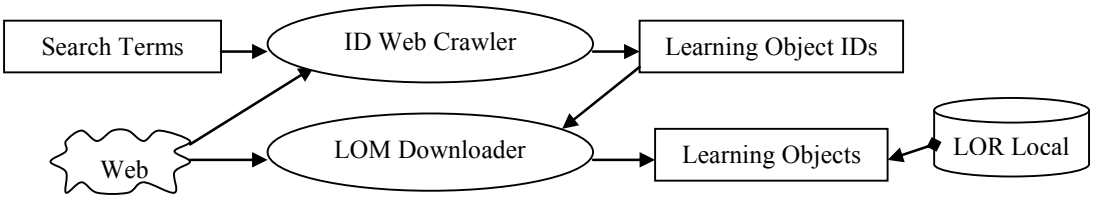


Figure 2: Web Crawler system architecture.

from NEEDS (2008), a digital library with links to learning material for engineering education, and SMETE (2008), a digital library and portal to access teaching and learning material in science, engineering and technology. The ID Web Crawler creates search queries and sends them to NEEDS and SMETE using the HTTP Post Method (Blum et al., 1998) to get search result pages. Each query is an HTTP Post Message using one of the keywords from the set of predefined keywords (usually frequent keywords from a domain of interest or from various domains). The ID Web Crawler sends the Post Message over the search URL. The second application, LOM Downloader, downloads the learning object metadata using IDs from the ID Web Crawler. It must be noted that the present Web Crawler does not download files with learning object content, but downloads files with learning object metadata (LOM). LOM usually includes the URL of actual learning object content; hence, there is no reason to download the learning object content for the purpose of the present crawler.

ID Web Crawler

The first application of the Web Crawler system, ID Web Crawler (Figure 3), handles both accessing the Web (performed by the Browser) and parsing the learning object web pages to obtain their IDs (performed by the Parser). Details of the ID Web Crawler showing the data flow of the developed application are presented in Figure 4. The application takes information on the website containing learning objects and takes the search terms used to generate the first page. The first page is used to get the query information that is needed to generate other pages within the search results. The application then enters a loop where it gets a page and parses the IDs from it, continuing until it reaches the last search result page.

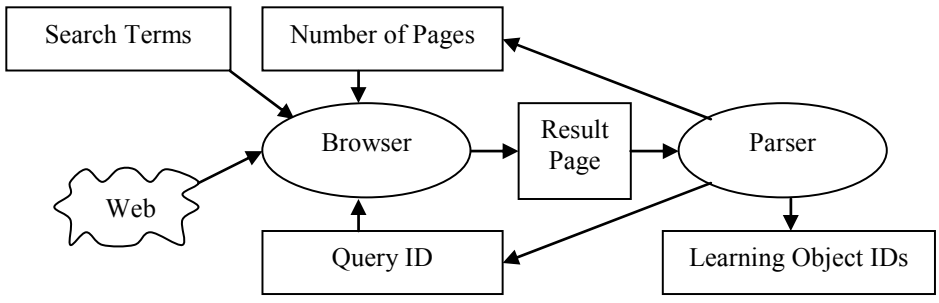


Figure 3: ID Web Crawler subsystem architecture.

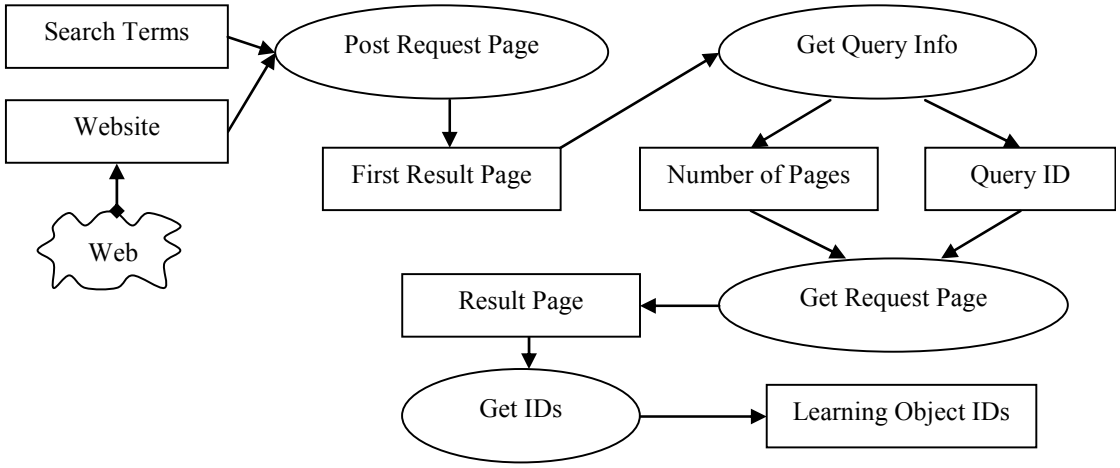


Figure 4: ID Web Crawler detailed data flow.

LOM Downloader

The second application, LOM Downloader handles both accessing the web and downloading XML files with learning object metadata (LOM). Figure 5 presents the data flow for this application. This system receives the learning object IDs from the ID Web Crawler and the URL for the learning object files excluding an actual ID. The application goes through and creates a complete URL for each learning object, and gets the LOM. The data is then written to a file on the local learning object repository.

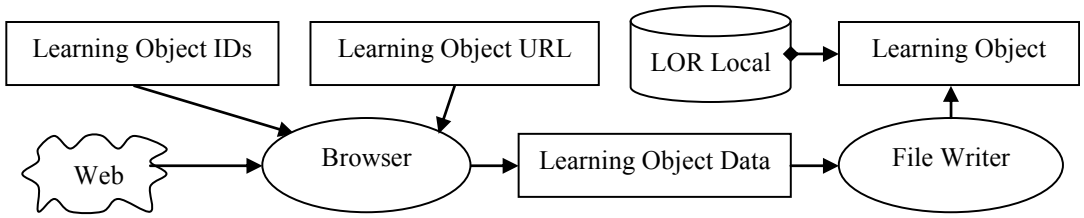


Figure 5: LOM Downloader data flow.

Auxiliary Components

The auxiliary components for the Web Crawler are the HTTP Library, Web browser, HTML parser, and page downloader. The HttpClient module has been re-used to build the web browser because it supports regular web browsing (GET) as well as submitting information through forms (POST), which can easily be distinguished with the separate classes GetMethod and PostMethod (Apache Software Foundation, 2008).This is the only needed functionality for web browsing, so from these specifications and a test run of a sample application it was chosen for all web browsing in both the ID and LOM Dowloader.

A compatible HTML parser was needed; in particular it was necessary to use a parser that would be able to accurately parse HTML documents even if they are not W3C validated. A validation of learning object web pages was run against the main page of both the search sites (NEEDS and SMETE), and neither one is valid (Validator, 2008). Therefore, it was recommended to use the Cobra HTML Parser for all parsing in the ID Web Crawler, because one of the specifications of this parser is the ability to parse "street HTML" (The Lobo Project, 2008).

After downloading each LOM XML file using the HTTPGet class it is written to a newly created file in the local learning object repository. For downloading learning objects, the BufferedWriter class from java.io was chosen. This can take data and write it to a newly created file in the local learning object repository.

The Web Crawler Design and Implementation

HTML Information

Before the system was created, the structure of the search website needed to be investigated to determine all the information needed to perform the task of searching and getting desired information from the results. The necessary information was determined from studying the HTML source code of search and result pages. Table 1 presents the necessary information to make a search and get information. Some of this information is needed to run the Web Crawler system application, and some of the information will need to be determined once the search query is made.

Table 1: Information needed on the search site

Name	Description
Domain	Main part of the URL such as http://www.needs.org
Search Action URL	Value of action attribute of the form tag. This is where the query will be processed.
Number of Results	Amount of results which will be used to determine the number of result pages.
Results per Page	Number of search results that appear on a single page. This will be used with the number results to determine the number of pages.
Query ID	ID for a specific search.
Result Url	URL that will lead to a search result page.
Result Parameter Names	Name of the GET parameters for page number and query id.
Learning Object URL	Used to get the actual learning object data.
Input Names	Name of input tags within the form which will be used to set POST parameters.
Input Values	The values for the inputs.
HTML Tags	Where the query ID, number of results, and Learning Object IDs appear.
Useless Information	Needed to determine what to cut out of the tag contents when parsing.

Web Browser

The web browser aspects of the application have separate methods in the ID Web Crawler. There is one method to handle making the search query and getting the first page. This method uses the PostMethod class. The second method handles getting all other search result pages for the query. This method implements the GetMethod class. The PageLoader class handles all web browsing in the first application. One instance of this class will exist for every site. Each time a page is needed the method of the current instance is called.

HTML Parser

Parsing is handled in a few places. The first place a parser was needed was the Cobra HTML Parser which is used to convert the result pages to instances of the Document class. Two other parsers are needed to get information from the Document instances. There needs to be a parser that will get the query ID and number of results for the search, and a parser that will get the learning object IDs. To get the query ID, number of search results, and the learning object IDs, HTML tags and regular expressions are used. For each piece of information, the specific HTML tag is found. This HTML tag includes the name of the tag, an attribute of the tag, and the value or part of the value of the attribute depending on whether the desired information appears within the attribute value or the contents of the tag. After a tag is found the desired information can be obtained. However, this information will have useless information around it. Regular expressions will be used to cut the useless information from both ends of the desired information. Table 2 shows the full contents of an HTML tag, the regular expressions that were used to cut the useless information, and the result.

Table 2: Parsing information within the contents of an HTML tag through the use of regular expressions

Full String	Search Results: 1 to 10 of 386 total results <!--, sorted by <select name="sort"> ...
Pre-RegEx	Search Results:[\D]+[\d]+ to[\D]+[\d]+[\D]+ where [\D] represents a non-digit and \d represents a digit
Post-RegEx	total results[\d | \s | \w]+ where \d represents a digit, \s represents a whitespace character, and \w represents an alphanumeric character
Desired Information	386

ID Web Crawler and LOM Downloader

Figure 6 illustrates the final class design of the Web ID Crawler with reusability in mind. If IDs are taken from search sites other than NEEDS and SMETE, then a few modifications will be needed. Classes were created with a goal to increase cohesion in the system. This was done to increase the reuse of different classes. This also makes it possible for classes to require only slight modifications or replacement classes to change the functionality of the system. For example, the parsers are separated into their own classes. If different information is desired from search results in the case of reuse, the PageIdParser class could be replaced with a class that finds some other information from the search result pages.

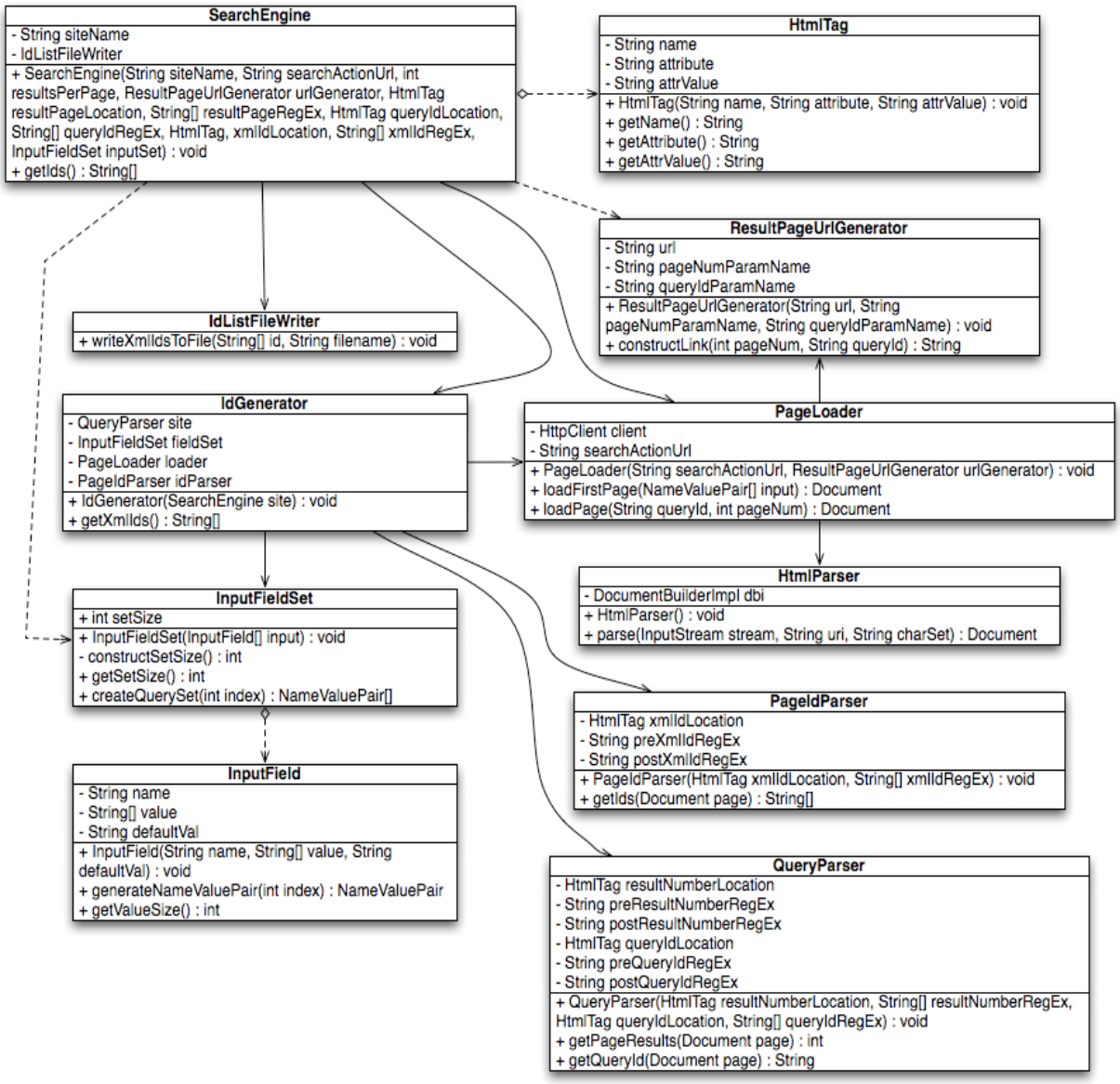


Figure 6: UML class diagram of the ID Web Crawler.

The LOM Downloader was developed reusing some code from the ID Web Crawler. Figure 7 illustrates the final design for the LOM Downloader. The LOM Downloader uses a Buffered Reader to read in the list of IDs, a method that uses the GetMethod class to get learning object data, and a BufferedWriter to write learning objects (in fact, their LOM) to files. Details of specifications of methods used in the application design are described below.

All of the information on the search website and the search queries are set when constructing an instance of the **SearchEngine** class. This class initializes other needed classes in the constructor, including a PageLoader and IDGenerator. SearchEngine has only one method, getIds(), which generates a complete list of learning object IDs based on the search queries and saves them to a file.

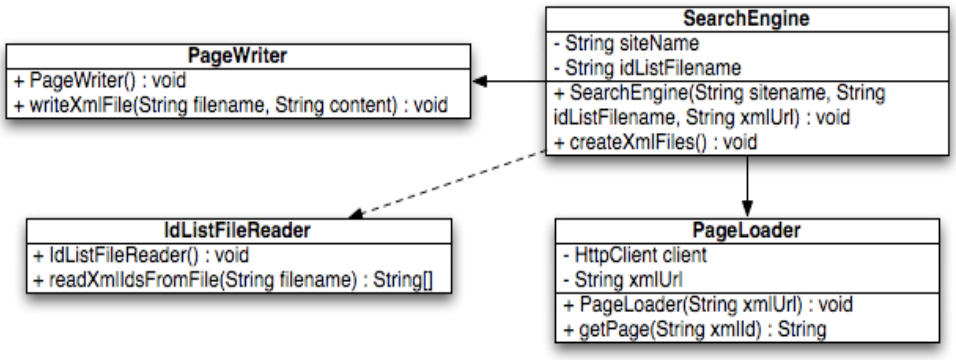


Figure 7: UML class diagram of the LOM Downloader.

A simple class was needed to represent HTML tags that would be used by the parser. The **HtmlTag** class was created to hold three pieces of information on an HTML tag, as follows: <name attribute=”attrValue”>. The HtmlTag is constructed with the tag name, an attribute, and the value of that attribute. This class has accessor methods for those three parameters.

The class **ResultPageUrlGenerator** is needed to generate the URL for each result page. There is the base of URL and parameter names that will always stay the same, but there is also the parameter values which will change for different queries and pages. Table 3 shows the design of a result URL. The constructor takes the base of the URL and parameter names to construct an instance of this class. Every time a URL needs to be generated, the constructLink(int pageNum, String queryId method can be called to get the new URL as follows: http://www.needs.org/needs/public/search/search_results/index.jhtml?queryId=FQ-1209430152461&page=2.

Table 3: The design of a result page URL (parts labeled constant never change between sites; therefore, these are added in by the class itself when a URL is generated)

Base	http://www.needs.org/needs/public/search/search_results/index.jhtml
Constant	?
Parameter Name	queryId=FQ-1209430152461&
Constant	=
Parameter Value	FQ-1209430152461
Constant	&
Parameter Name	page
Constant	=
Parameter Value	2

The **HtmlParser** class makes use of the Cobra Html Parser. The constructor sets up the parser with the method parse(), which takes information about a result page, and creates a Document instance, which can be used to easily get information from the page.

The **PageLoader** class is used to generate Document instances for search result pages. It has an instance of the ResultPageURLGenerator and the HtmlParser. The method loadFirstPage() takes a

search query and generates the first result page. The method loadPage() takes the page number and query id and can load any result page using the ResultPageURLGenerator.

In order to get query information, the **QueryParser** class is used. This parser is initialized with HtmlTag instances and regular expressions. These parameters are used to generate the query ID and the number of results using the methods getQueryId(Document page) and getPageResults(Document page) respectively. The query ID is found within the HTML tag attribute value and the number of results is found within the contents of an HTML tag.

The **PageIdParser** is initialized with the HtmlTag the learning object IDs appear in and regular expressions for the unnecessary information around the IDs. The learning object IDs are obtained within the HTML tag attribute value. The method getIds(Document page) takes a page and gets all of the learning object IDs from it.

InputField is a simple class to represent an input field on the search page. It is initialized with the name of the field, all of the values to use in this field, and a default value. The method getValueSize() returns the number of values the class was initialized with. The method generateNameValuePair(int index) creates a NameValuePair instance that can be added to an array for making a search query. If the index is greater than or equal to the number of values the default value is used.

An array of InputField instances for a site is held in an **InputFieldSet** instance. This class can construct an array of NameValuePair instances for each query to be made for a certain search site.

The **IdGenerator** handles the interoperation between many of the other classes and handles making new searches, getting the query information, parsing pages, and returning the list of learning object IDs.

A simple class is also needed to write the learning object IDs to a file. The **IdListFileWriter** takes an array of learning object IDs and uses a BufferedWriter and FileWriter to write them to a file.

The LOM Downloader consists of four classes that are very similar to the classes used in the ID Web Crawler. The **SearchEngine** class is initiated with the URL to learning object excluding the specific ID and the file containing all of the IDs. It has one method, createXmlFiles(), which handles reading the list of learning object IDs and saving the learning object to the computer.

In order to get an array of the learning object IDs, a BufferedReader is used to read the learning object ID list file and save each line to a new element in the array. This is all handled in the **IdListFileReader**.

In the LOM Dowloader, no POST requests need to be made. Therefore, the **PageLoader** class for this application only has a method to load a page taking in the ID for a learning object and combining it with the base URL to a learning object.

Given the data of a learning object and the desired filename, the **PageWriter** class has a method to create the learning object file, using the BufferedWriter class.

Evaluation

The following aspect of the Web Crawler has been evaluated: how performance is changed with the increase of number of downloaded web pages (learning objects). Since the Crawler is devoted to download LOM's, but not to retrieve information from LO's, the traditional measures of precision and recall are not used for the evaluation. The most important criterion in this case is efficiency, in particular liner complexity of downloading on the raise of the number of downloaded LOM's. To evaluate the change of performance, the Web Crawler has been run with web pages

from both NEEDS and SMETE digital libraries (NEEDS, 2008, SMETE, 2008). The number of learning objects downloaded has been counted at equal time steps (approximately every 55 seconds). The result of these search experiment is presented in Figures 8 and 9. The result demonstrates linear dependence of the number of downloaded learning objects on time spent until the digital library begins to exhaust. This demonstrates the crawler meets the objectives.

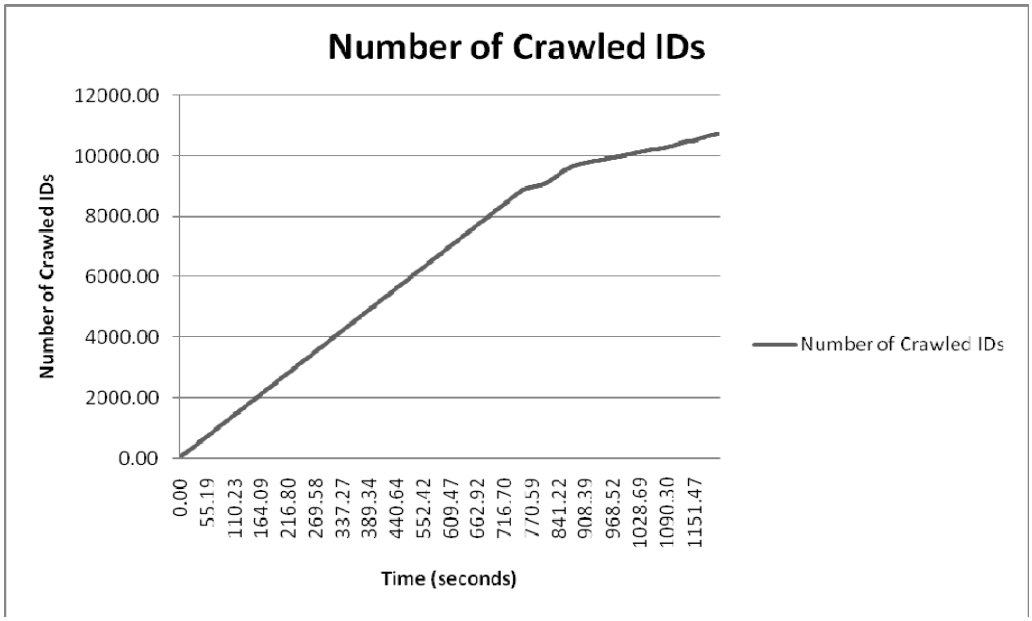


Figure 8: Performance of the crawler for NEEDS repository

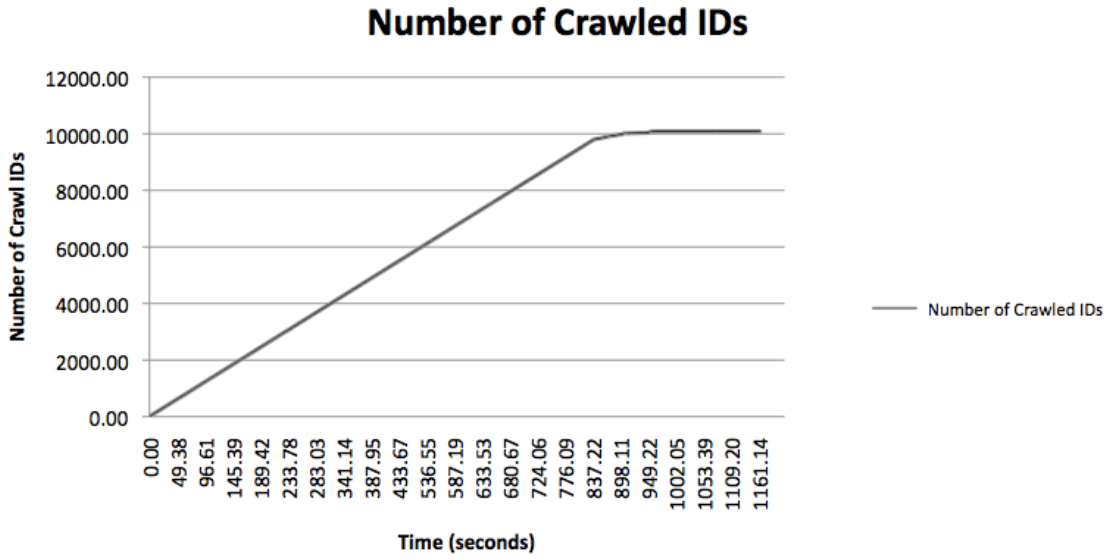


Figure 9: Performance of the crawler for SMETE repository

Conclusion

The paper presented a technical solution, Web Crawler, to automatically download a vast number of learning objects (together with LOM) from open digital libraries on the Web to a local learning object repository with the purpose to essentially increase efficiency of search and delivery of learning objects to learners using various services, such as personalization, learning course or program assembly, etc. Another important objective of crawling learning objects is to release users from long routine work on finding and downloading learning objects. Evaluation of the presented Web Crawler involving two open web digital libraries of learning material, NEEDS and SMETE, demonstrated that the crawler meets the objectives. In addition, the crawler uses the HTML Parser with the capability to accurately parse HTML documents, which are not W3C validated. This capability is important for digital libraries of learning objects available on the Web.

There are some directions for future work. The presented Web Crawler needs the development of another parser(s) that would be able to search hierarchy of XML tags instead of just a single tag. An API must be developed to make the Crawler more usable and more universal to a wider set of digital libraries.

References

Apache Software Foundation. (2008). *HTTP Client*. Retrieved August 2, 2008, from http://hc.apache.org/httpclient-3.x/features.html

Aroyo, L., Dolog, P., Houben, G. J., Kravcik, M., Naeve, A., Nilsson, M., & Wild, F. (2006). Interoperability in personalized adaptive learning. *Educational Technology & Society, 9*(2), 4–18.

Baghi, H., Biletskiy, Y., & Li, H. (2008). A blended approach for search of learning objects. *IEEE Canadian Conference on Electrical and Computer Engineering CCECE08*, Niagara Falls, Canada, pp. 1805-1809.

Biletskiy, Y., Baghi, H., Keleberda, I., & Fleming, M. (2008). Adjustable personalization of search and delivery of learning objects to learners. *Expert Systems with Applications.* doi:10.1016/j.eswa.2008.12.038.

Biletskiy, Y., Brown, J. A., & Ranganathan, G. R. (2009). Information extraction from syllabi for e-advising. *Expert Systems with Applications,* Elsevier, *36*(3), Part 1, 4508-4516.

Biletskiy, Y., Vorochek O., & Medovoy A. (2004). Building ontologies for interoperability among learning objects and learners. *Lecture Notes in Computer Science*, Springer-Verlag, 3029/2004, pp. 977-986.

Blum, T., Keislar, D., Wheaton, J., & Wold. E. (1998). *Writing a web crawler in the Java programming language*. Sun Developer Network. Retrieved August 2, 2008, from http://java.sun.com/developer/technicalArticles/ThirdParty/WebCrawler/index.html

Boley, H., Bhavsar V. C., Hirtle, D., Singh, A., Sun, Z., & Yang, L. (2005). A match making system for learners and learning objects. *Interactive Technology and Smart Education*, *2*(3), 171-178.

Devedzic, V. (2004). Education and the semantic web. *International Journal of Artificial Intelligence in Education*, 14, 39-65.

Friesen, N. (2005). Interoperability and learning objects: An overview of e-learning standardization. *Interdisciplinary Journal of Knowledge and Learning Objects*, *1,* 23-31. Retrieved from http://ijello.org/Volume1/v1p023-031Friesen.pdf

Jovanović, J., Gašević, D., Knight, C., & Richards, G. (2007). Ontologies for effective use of context in e-learning settings. *Educational Technology & Society*, *10*(3), 47-59.

Keleberda, I., Repka, V., & Biletskiy, Y. (2006). Semantic mining based on the learner's preferences. *The 2006 IEEE Canadian Conference on Electrical and Computer Engineering CCECE06*, Ottawa, Canada, pp. 502-504.

The Lobo Project. (2008). *Cobra: Java HTML parser*. Retrieved August 2, 2008, from http://lobobrowser.org/cobra/java-html-parser.jsp

NEEDS. (2008). *A digital library for engineering education*. Retrieved August 2, 2008, from: http://www.needs.org/needs/

Pahl, C., & Hologan, E. (2009). Applications of semantic web technology to support learning content development *Interdisciplinary Journal of E-Learning and Learning Objects*, 5, 1-25. Retrieved from http://ijello.org/Volume5/IJELLOv5p001-025Pahl409.pdf

SMETE. (2008). *SMETE digital library*. Retrieved August 2, 2008, from http://www.smete.org/smete

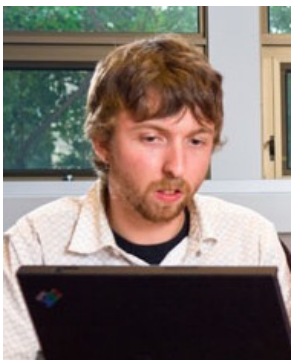
Validator. (2008). *W3C markup validation service*. Retrieved August 2, 2008, from http://validator.w3.org

Wang, T. I., Tsai, K. H., Lee, M. C., & Chiu, T. K. (2007). Personalized learning objects recommendation based on the semantic-aware discovery and the learner preference pattern. *Educational Technology & Society*, *10*(3), 84-105.

Zouaq, A., Nkambou, R., & Frasson, C. (2007). An integrated approach for automatic aggregation of learning knowledge objects. *Interdisciplinary Journal of Knowledge and Learning Objects*, *3*, 135-162. Retrieved from http://ijello.org/Volume3/IJKLOv3p135-162Zouaq.pdf

Biographies

Dr. Yevgen Biletskiy received his Ph.D. degree from Kharkiv State Technical University of Radio-Electronics, Ukraine in 2000. He is currently an Associate Professor at the Department of Electrical and Computer Engineering, University of New Brunswick, Canada. He is currently conducting research in the field of extracting, building and integrating ontologies with the purpose to enable semantic interoperability between heterogeneous information sources and users involving the Semantic Web infrastructure, in particular two Semantic Web layers: ontologies (OWL) and rules (RuleML). His research is extensively applied the domains of e-Learning and e-Business.



Mike Wojcenovich received his B.Sc. degree in Software Engineering at the University of New Brunswick, Canada in 2009. During his studies he was interested in development of web and Java based applications in the field of information retrieval and extraction. He worked at Chalk Media in Fredericton (Canada). He is currently participating in developing a web application which parses information from various video sources such as YouTube and Google Video, and this work has already won prizes in the CIBC Business Plan Competition and the Atlantic (Canada) Engineering Competition.

Hamidreza Baghi received his B.Sc. degree in Computer Engineering with specialty in software from Amirkabir University of Technology, Iran in 2006. He received his M.Sc. degree in Computer Science from the University of New Brunswick, Canada in 2008. During his studies he has been working on topics such as web search, text mining, knowledge engineering, and learning objects. His fields of interests include information retrieval, text mining, social network analysis, and automated software engineering. He is currently continuing his research with Dr. Biletskiy.