INTELLIGENT TAGGER FOR E-LEARNING

EECS 891 Project Report
Fall 2003

By
Ubayeedurrahman Syed

Advisor
Dr. Susan Gauch

Committee Members
Dr. Costas Tsatsoulis
Dr. Arvin Agah

Table of Contents

1. Introduction

1.1 Overview

Traditionally, due to language differences and a lack of standards, Information Systems were built as stand alone applications that could not inter-operate. Today, the Web makes communication between these systems far easier, and this makes XML (eXtensible Markup Language) technology important. XML represents a powerful way to overcome semantic barriers to information exchange. It allows the author to describe the data of virtually any type in a structured manner. With *XML*, data can now be stored in separate *XML* files or in databases, allowing developers to concentrate on using HTML for formatting and display.

The discipline of computer-assisted instruction in the field of Educational Technology has always been at the vanguard of Knowledge Management Engineering, and has, in general been an early user of a wide range of learning environments. Practitioners in this area quickly realized the importance of personal computers and today they are leading the education technology field with some very interesting uses of XML in practice. In these initiatives, the basic unit of content is called a Learning Object.

A Learning Object is defined as any entity, digital or non-digital, which can be used, re-used or referenced during technology-supported learning. Examples of technology-supported learning include computer-based training systems, interactive learning environments, intelligent computer-aided instruction systems, distance learning systems, and collaborative learning environments. Examples of Learning Objects include multimedia content, instructional content, learning objectives, instructional software and software tools, and persons, organizations, or events referenced during technology-supported learning.

Although Learning Objects may encourage instructors to operate in a more disciplined way, their major disadvantage is that a considerable amount of effort is needed in providing a description of each Learning Object when it is created. This requires a lengthy, manual process.

1.2 Motivation (IKME)

The Intelligent Knowledge Management Environment (*IKME*) is an ongoing project at the *University of Kansas* aimed at assisting the *Defense Information Technology Testbed (DITT)/University After Next (UAN)* by providing an advanced reach-back capability for commanders, staff, and other users who have time-critical needs. The knowledge management environment facilitates the creation of extensible and reusable learning objects that would lead to faster delivery of content to knowledge users.

The project is based on the idea of using the Extensible Markup language as the data format for publishing. The environment facilitates the Knowledge creators to create learning objects, which are stored as *XML* documents. These learning objects are based on an *XML* schema developed by the "*Center for Army Lessons Learned*". These learning objects can in turn be reused to create lesson objects, which in turn are used to create a manual. This setup facilitates ease of creation and also faster delivery of content to the end users. Another main advantage of using *XML* as the data format is the separation of content and style. The same *XML* document can be represented in various styles and data formats using style sheets. For example the manual can be published online using an *XSLT* stylesheet and also converted to a *PDF* file (for printing) by using the *XSL-FO* sytlesheet.

1.3 Objective

The main objective of this project is to facilitate the creation of new Learning Objects by developing a mechanism to identify the user's preferences from the user's history. The current schema has a total of 28 attributes to create a Learning Object, of which there are seven enumerated ones. Currently when a user creates a Learning Object they must fill in enumerated fields each time, our enhancements help decrease the effort required to add new Learning Objects to the system, overcoming one of the major obstacles in the use of Learning Objects.

1.4 Approach

Users create Learning Objects, which are stored as XML documents. The options for enumerated fields in the Learning Objects are automatically generated from the

Learning Object Schema using a schema parser. The values for these enumerated fields are read from a sample template *XML* file. The template file initializes all enumerated fields to blank values, hence when the user each time creates a new Learning Object he/she has to manually select the values for the enumerated fields. This project is an effort to facilitate the user by providing the user with the values that closely match to his or her preferences, hence saving a valuable amount of user's time when creating a new Learning Object.

eXist, a Native *XML* Database (NXD), is used as the backend to store all the Learning Objects or *XML* documents. Tools are provided to store documents into the database. User can choose the number of latest documents to be considered for his history to determine the most preferred tag value.

Our technique is based on exploiting user history to automatically customize the create Learning Object form based on the user's most popular frequent field values. This approach can be seen in commercial websites. For example a user "John" requesting for a web server with a login system may find the value "John" already existing in the user name.

Cookies are a very popular technique for allowing web-servers to keep track of sessions, so that the server can easily identify the same user preferences from one request to the next. They can store login information, keep track of shopping patterns, record your preferences for that particular site, and more. Although cookies are useful, they are disliked by many people mainly because of privacy and security concerns. Another disadvantage is that they provide very little information about the user’s likings. We need a longer user activity history trend, so we did not choose to base our system on cookies instead we have designed a more sophisticated mechanism.

The login system of ChatTrack, another ongoing project at University of Kansas, has also been integrated with IKME to require users to be registered members of IKME in order to create Learning Objects.

2. Technical Background

2.1 Reusable Learning Objects

Traditional content development techniques face many problems in terms of extensibility or reusability. Any modification to, or reuse of the content requires effort and time from the publisher. Reusable Learning Objects represent an alternative approach to content development. In this approach, content is broken down into chunks of information. Object-orientation highly values the creation of components called “objects” that can be combined or recombined in multiple contexts to create the customizable learning tracks. This is the fundamental idea behind Learning Objects. Instructional designers can build small components that can be reused a number of times in different learning contexts. One main advantage of following the Learning Objects approach is that when appropriate content already exists, they reduce development time considerably. Content creators can create customized manuals or lessons at much faster rate than using conventional techniques.

2.2 Native XML databases (NXD)

Native XML databases are databases designed especially to store *XML* documents. They also provide support features commonly found in other databases like transactions, security, multi user access, programmatic API’s, query languages etc. The main difference from other databases is that their internal model is based on *XML*. The

XML:DB Initiative offers the following formal definitions for a *Native XML database*:

a) Defines a (logical) model for an *XML* document and stores and retrieves documents according to that model. Examples of such models are the *XPATH* data model, the *XML* Infoset, and the models implied by the *DOM* and the events in *SAX 1.0*.

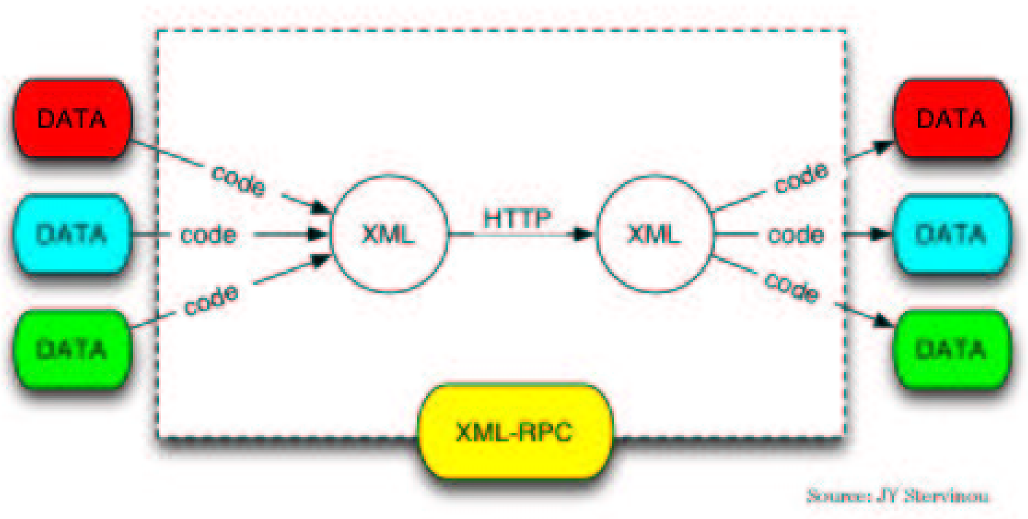
b) Has an *XML* document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage.

c) Is not required to have any particular underlying physical storage model. For example, it can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed, compressed files (source: http://www.xml.com/pub/a/2001/10/31/nativexmldb.html).

NXD's do not really represent a new low-level database model, and are not intended to replace existing databases. They are simply a new tool intended to assist a developer by providing robust storage and manipulation of *XML* documents.

2.3 XML-RPC

XML-RPC is a specification and set of implementations that allow software running on disparate operating systems and in different environments to make procedural calls over the Internet. It's remote procedure calls uses *HTTP* as the transport protocol and *XML* as the encoding. *XML-RPC* is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned. Remote procedural calls and input parameters are serialized into *XML* before sending to the *RPC* server; at the server side the *XML* input is un-serialized into its local data structures. The output is again serialized from local data structures into *XML* before sending it back to the client. In this way disparate systems could communicate irrespective of their computing environment.



XML-RPC diagram (source: http://www.xmlrpc.com/)

3. Design Considerations

3.1 Process of Creating Learning Objects

Learning Objects may use different instructional strategies to promote learning. Instructional Strategy may be defined as a set of rules with which learning content may be structured or sequenced. One can apply different Instructional Strategies depending upon the task to be learned and the learner's style. Different templates/tools may be used to create the learning objects.

The sample template (*XML* file) has all the required tags for a Learning Object initialized with blank values. Whenever the user creates the Learning Object this template file is loaded and displayed on to the screen using the user's browser. After the Learning Object is created it is stored in the XML database. There are fields in the Learning Objects whose values must be assigned from an enumerated list, e.g., FORMAT can be assigned either MIME type or TEXT/HTML type depending upon the data representation. The enumerated values used frequently remain the same for a sequence of related Learning Objects created by a particular user. Yet, with the current system, the user must select the desired value each time. To customize the Web page to aid the user, we will adapt the sample template for the user based upon their history.

3.2 Why eXist?

To obtain the history of a user's choices for the enumerated values, one of the designs considered was a flat-file mechanism. In other words, a history-file is maintained for each user that stores the values for the enumerated tags in a specific format. Whenever the user creates new Learning Objects, the history file corresponding to the user is accessed and processed to determine the field value. The limitations of this approach are that the data is isolated and separated, and each user maintains his or her own set of data. The file structure has to be defined by the program code, at the time of creation the system must search the file for each value, and space is wasted. In addition, any schema change may need a new program and incompatible file formats could also be a problem.

We also considered using cookies to store the user's preferences. Cookies can be used as a trick or a simple technique to store user identification or other information for each visitor to a website. They allow user specific functions like personalization of web pages for each user, tracking of visitors, etc. One good use of cookies is that they gather statistics such as the number of unique visitors to the site, provide the information about the movement of the user through the site, provide information such as how long and how often a user viewed a particular Web page, etc. ProFusion is one of the best examples that use cookies. In our case, determining the user's preferences from the user's history is a key factor. Using cookies, one could get only the last preference of the user, hence knowing the user's history for the enumerated tags is not possible by using cookies.

Because of the limitations of the flat file system and cookies, a database approach was chosen. We found by experimentation that the *XML* files in the eXist database are stored in *chronological* order, i.e., querying the database returned the latest documents first. The knowledge of order of documents in the database allowed us to easily retrieve the most recent values from the database. The database can be queried from the application and the recent values for the enumerated fields can be retrieved from the database. Once the recent values are obtained, a selection process can be applied to determine the final value from the results. The user is also able to choose the number of recent documents to be considered for his or her history.

3.3 Determining the preference from the user's history

When the user creates a new Learning Object, our system sets default values for each of the enumerated fields based upon the most frequently used value for that field in the user's recent past. To do this, at first the system parses the comma delimited file 'table.txt' created by one of the code file *Parse_Schema.pl*. 'table.txt' contains formatted information for tag/value pairs in the Learning Object. The system parses this file to identify the enumerated tags in the schema and stores them in a one-dimensional array called enumerated-tag-array.

For each enumerated tag in *enumerated-tag-array*, the system queries the database for the most recent *N* Learning Objects. This returns the *N* Learning Object ids and the enumerated field in *XML* format. These results are parsed, the values of the enumerated tags are extracted and stored in a temporary one-dimensional value array.

The one-dimensional value array is passed to a sub routine that identifies the most frequently occurring value. This sub routine has a temporary one-dimensional count-array which is of the same size as the value array. The count-array stores the count for each value in the value array that appears in the next index locations, for e.g., the enumerated field ‘PORTIONMARK’ has the following value-array from the recent five Learning Objects ‘Value-Array [5] = {'T','TS','T','TS','S'}’. Its count array will be ‘Count-Array [5] = {'2','2','1','1','1'}’. The *index* with maximum count in the count array is noted as the index of the most frequently occurring value in the value array. When two or more indexes have the same maximum count in the count array, the smaller index is chosen to select the most recently used value. In the previous example, the index 0 would thus be chosen, resulting in ‘T’ as the selected value.

4. Implementation

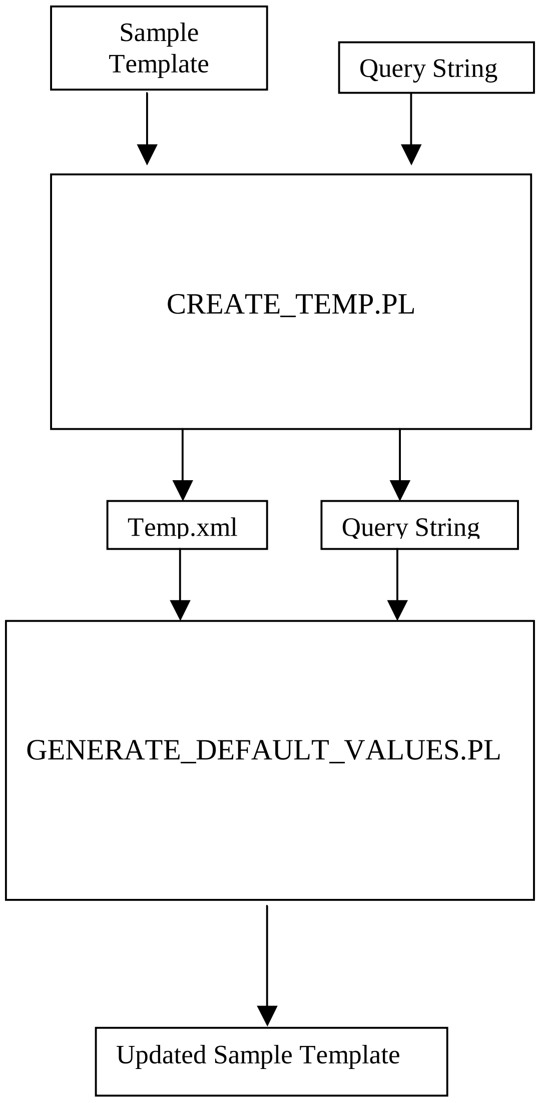
This chapter provides an overview of the code written for this project.

4.1 Software Requirements

- Perl Version 5.0 and later
- Apache Web Server
- Exist V0.9 and later
- Additional PERL Modules
    - CGI
    - RPC::XML
    - RPC::XML::Client
    - XML::Twig
    - XML::Sablotron
    - Data::Dumper

4.2 System Design

The inputs to the system are a Sample Template file and the Query-String. The system output is an updated Sample Template file. The two scripts in the implementation of this project are *create_temp.pl*, which takes in the system inputs and outputs a temporary XML file and *Generate_default_values.pl*, which takes in the temporary XML file from the output of *create_temp.pl* as input and produces the updated sample template file as its output.



Sample Template file, is an *XML* file that has all the mandatory tags of the XML Schema and Query String, is the array of attribute values obtained from the users Web page as a cgi query string when creating Learning Objects.

4.2.1 Parse_schema.pl

This script parses the XML Schema and produces a comma delimited flat file as output. This file follows a specific format with different characters in it. The tags that end with '$' can have text boxes in the XSL and only these values will be modified. The tags that end with '$$' are enumerated tags in the schema. One of the restrictions is that only the tags that are coded as <xs: element> in the schema will be processed. For example if we have an XML file.

```
<A>
  <B>
    <C type=""/>
        <D/>
   </B>
 <E/>
</A>
```

The output flat file (table.txt) would be as follows

/A	1	A # B # C ! type !! D # E ##	-
/A/B	1	B # C ! type !! D ##	-
/A/B/C	1	C ! type !!	$
/A/B/D	1	D	$
/A/E	1	E	$

The output file is called 'table.txt' and is loaded each time by the other script files. It contains special characters like '#', '!', etc which are useful in inserting the attribute values provided by the user to the right fields.

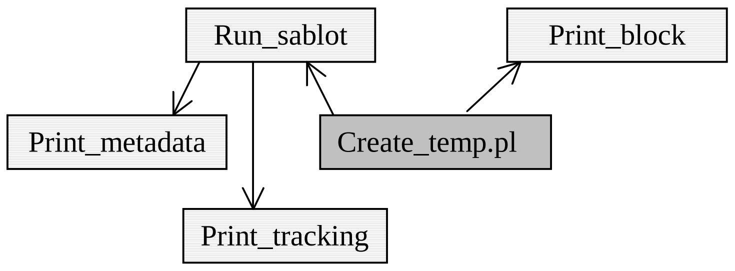
4.2.2 Create_Temp.pl

The purpose of this script file is to produce a temporary XML file depending upon the field to be modified. For example, in the XML file considered above, if field A is to have to its value modified, then the temporary file produced would contain field A and all its children. For IKME version 1.0, the XML-Schema has 5 main fields: 'metadata', 'tracking', 'content', 'views' and 'reconstruction'. The field and its children nodes are extracted from the sample template file which is also an XML file. This temporary file is overwritten whenever the user clicks a different field. It calls XSLT to display the temporary file

4.2.2.1 Algorithm:

```
Inputs – Sample Template, Query String
Outputs – Temporary file
  a. Load the Sample Template
  b. Extract the required field
  c. Print the field into the Temporary file
  d. Use XSLT, print the Sample Template (with default values)
     on to the screen
```

4.2.2.2 Module Design



4.2.2.3 Module Specifications

Title	Run_sablot
Purpose	Runs the sablotron to display the XML file
Returns	None
Input	None
Note	It displays the '/home/academy/htdocs/ikme/phoneix/temp.xml'

Title	Print_metadata
Purpose	To check if the required tags of metadata are filled
Returns	None
Input	None
Note	It's a java script function

Title	Print_tracking
Purpose	To check if the required tags in tag Tracking are filled
Returns	None
Input	None
Note	It's a java script function

Title	Print_block
Purpose	Prints the tag and all its sub tags into the temporary file
Returns	None
Input	None
Note	It is called while creating a new XML twig

4.2.3 Generate_default_values.pl

This is the main script file of the project. It modifies the values of the enumerated fields by reading from cgi query string. It loads the output of *Parse_Schema.pl* to insert the contents into the Learning Object (*XML* file). When modifying the tags, the enumerated tag contents are identified and stored in an array. It accesses the empty temporary file written by C*reate_Temp.pl*. Whenever the user clicks ‘done’ it updates the temporary file with all the contents from the user and then combines the updated temporary file with the original document.

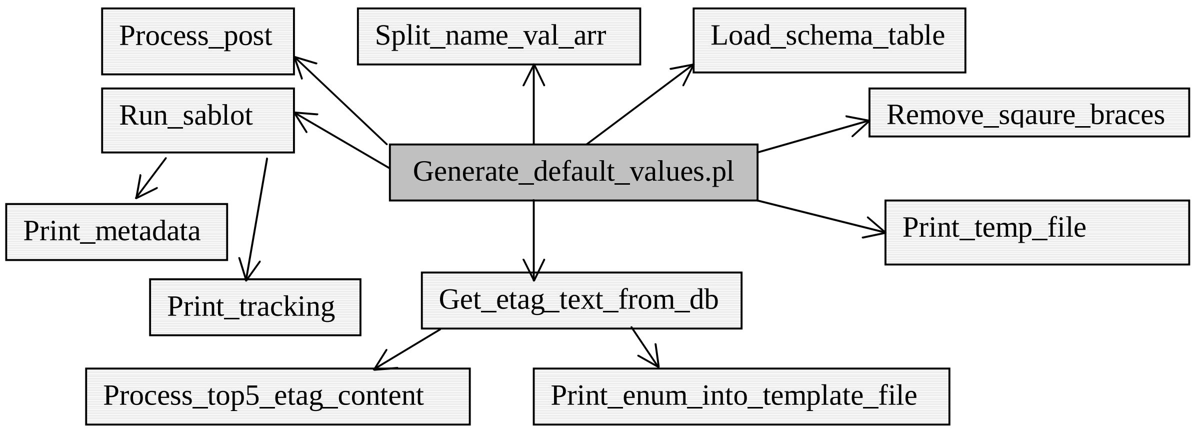
After the original document is written, it stores the document into the eXist database. It queries the database for each enumerated tag, processes the top ‘x’ number of results where ‘x’ is the number of recent documents to be considered for user’s history chosen by the user. For all the enumerated tag content, we pick the most common user’s preference from the retrieved results and insert these preferences into the template document. This CGI calls the XSL processor every time to display changes.

In the next login, when the user creates a new Learning Object, *create_temp.pl* displays the template file, the user can find some pre-existing values for these enumerated fields that closely match to the his or her likings. The restriction of this CGI is that all files being created or modified validate against the schema that was parsed into a table (output of *parse_schema.pl*).

4.2.3.1 Algorithm:

```
Inputs – Temporary File, Query String
Output - Updated Sample Template
 a. Store the Learning Object into the database
 b. Identify and store the enumerated fields in the Schema.
 c. FOR each enumerated field DO
      - Query the database and retrieve the content
 d. Process the top result to determine the default value
 e. Modify the Sample Template file with the default values.
```

4.2.3.2 Module Design



4.2.3.3 Module Specifications

Title	Process_post
Purpose	To get the values from the post method
Returns	An array – that stores post query
Input	An array
Note	None

Title	Split_name_val_arr
Purpose	Decode the Query String to split into name and value arrray
Returns	Scalar value
Input	3 arrays - name_val, name_arr, value_arr
Note	Refer the code for more information.

Title	Load_schema_table
Purpose	To load the o/p of parse_schema.pl into hash variable 'schema_table'
Returns	Hash table
Input	None
Note	This function access the o/p file "/home/academy/htdocs/ikme/phoneix/table_ubayeed" produced by parse_schema.pl. This o/p file has a specific format to know where to insert the post values when read.

Title	Print_temp_file
Purpose	To print the temp file
Returns	None
Input	None
Note	Temp file is at '/home/academy/htdocs/ikme/phoneix/temp.xml'

Title	Run_sablot
Purpose	Runs the sablotron to display the XML file
Returns	None
Input	None
Note	It displays the '/home/academy/htdocs/ikme/phoneix/temp.xml'

Title	Print_metadata
Purpose	To check if the required tags of metadata are filled
Returns	None
Input	None
Note	It's a java script function

Title	Print_tracking
Purpose	To check if the required tags in tag Tracking are filled
Returns	None
Input	None
Note	It's a java script function

Title	Remove_square_braces
Purpose	To remove square braces and number present in the path
Returns	None
Input	None

Note	None

Title	Get_etag_text_from_db
Purpose	To determine the value for each enumerated tag in the sample template a) query Database for each enumerated tag b) apply the selection process on (a)
Returns	None
Input	None
Note	a) E.g. of Xpath Query used “$query = 'xcollection('/db/ikme_ubayeed')/.portionmark” b) calls the function ‘process_top5_etag_content’ c) calls the function ‘print_enum_into_template_file’

Title	process_top5_etag_content
Purpose	It’s a selection process to determine the final value for the enumerated tag.
Returns	A string
Input	String – values from result of query
Note	None

Title	Print_enum_into_template_file
Purpose	To print the final enumerated values into the sample template file
Returns	None
Input	None
Note	Modifies the file /home/academy/htdocs/ikme/phoneix/sample_template_new.xml

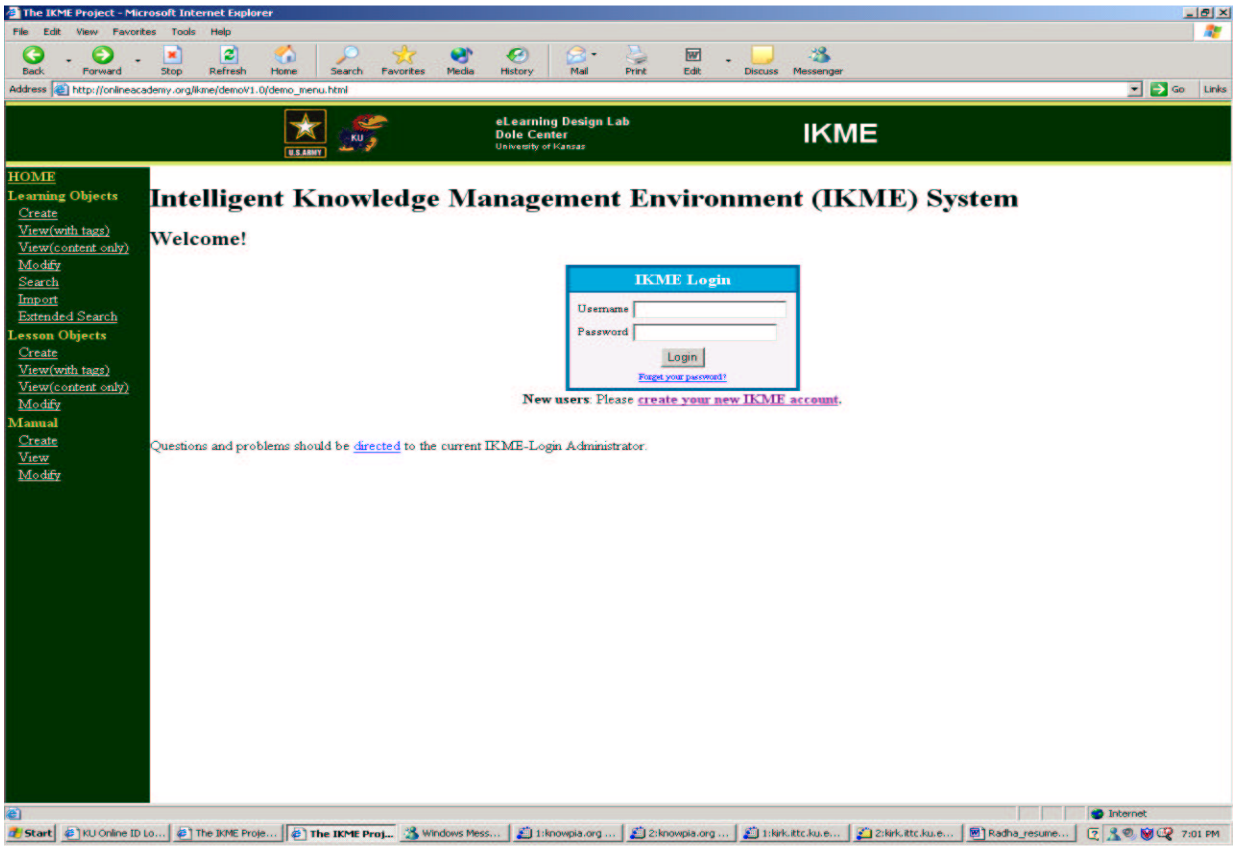
4.2.4 Print_html_file.pl and Print_html.pl

The first script prints out the html page to name the new Learning Object being created. The latter script prints out the HTML page with all the high level fields of the schema “Metadata, Tracking, Content, Views, and Reconstruction”. Clicking each field executes another script file (*create_temp.pl*), which produces the customized web page with all the sub-fields of the main field.

5. Evaluation

5.1 Sample Scenario

This section shows the login screen created for the IKME system that restricts access to the database for creating, viewing, or modifying the Learning Objects.



Intelligent Knowledge Management Environment (IKME) System

Welcome!

IKME Login

Username

Password

Login

Forget your password?

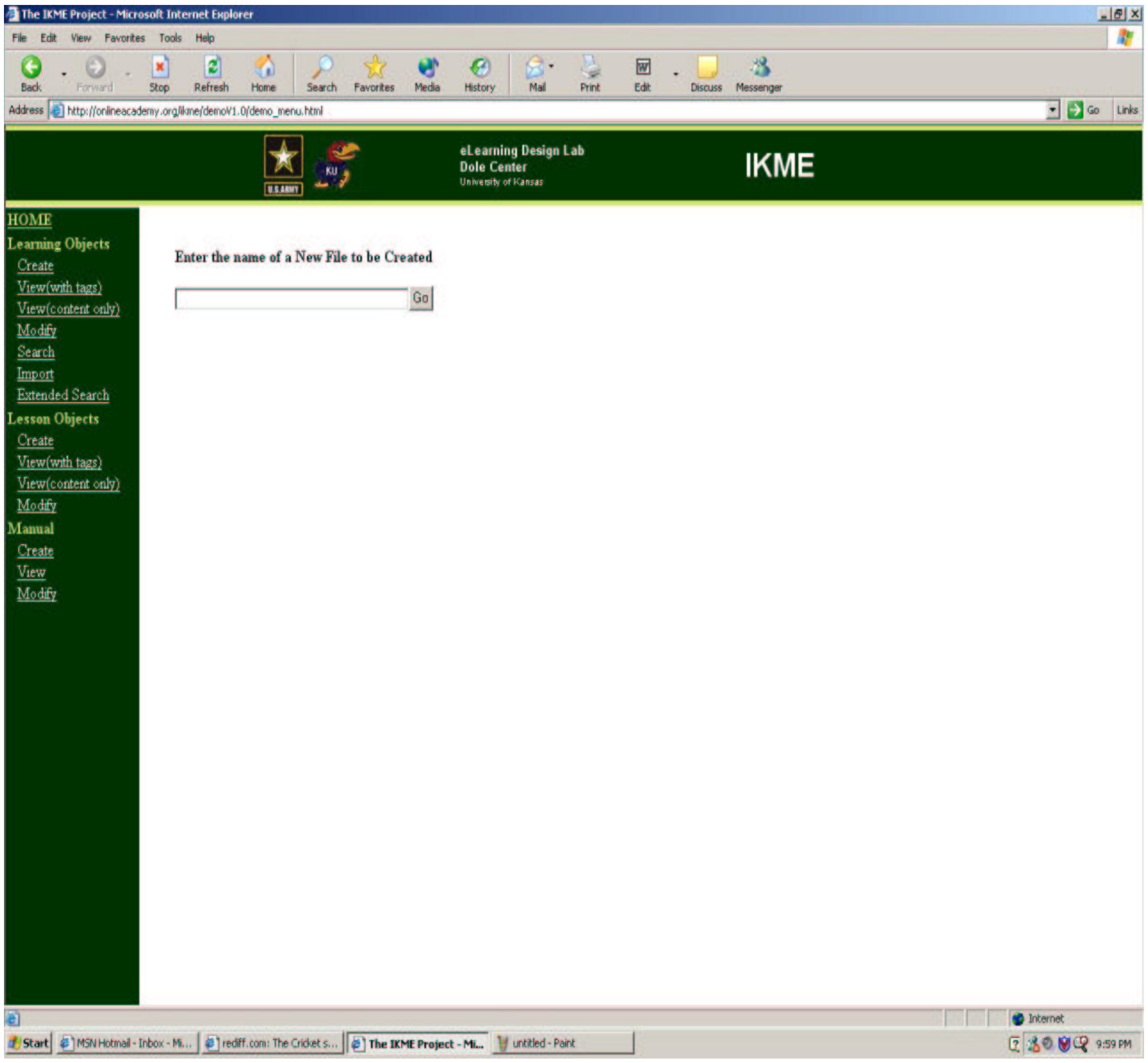
New users: Please **create your new IKME account**.

Questions and problems should be directed to the current IKME-Login Administrator.

Figure 5.1 The IKME login page

5.2 Name the Learning Object

The user begins by naming the new Learning Object



Enter the name of a New File to be Created

Figure 5.2 Naming the Learning Object

5.3 Highest Level fields in the Schema for IKME Version 1.0

The user is presented with the highest-level fields of the Learning Object schema.

Clicking each field produces all its sub-fields.

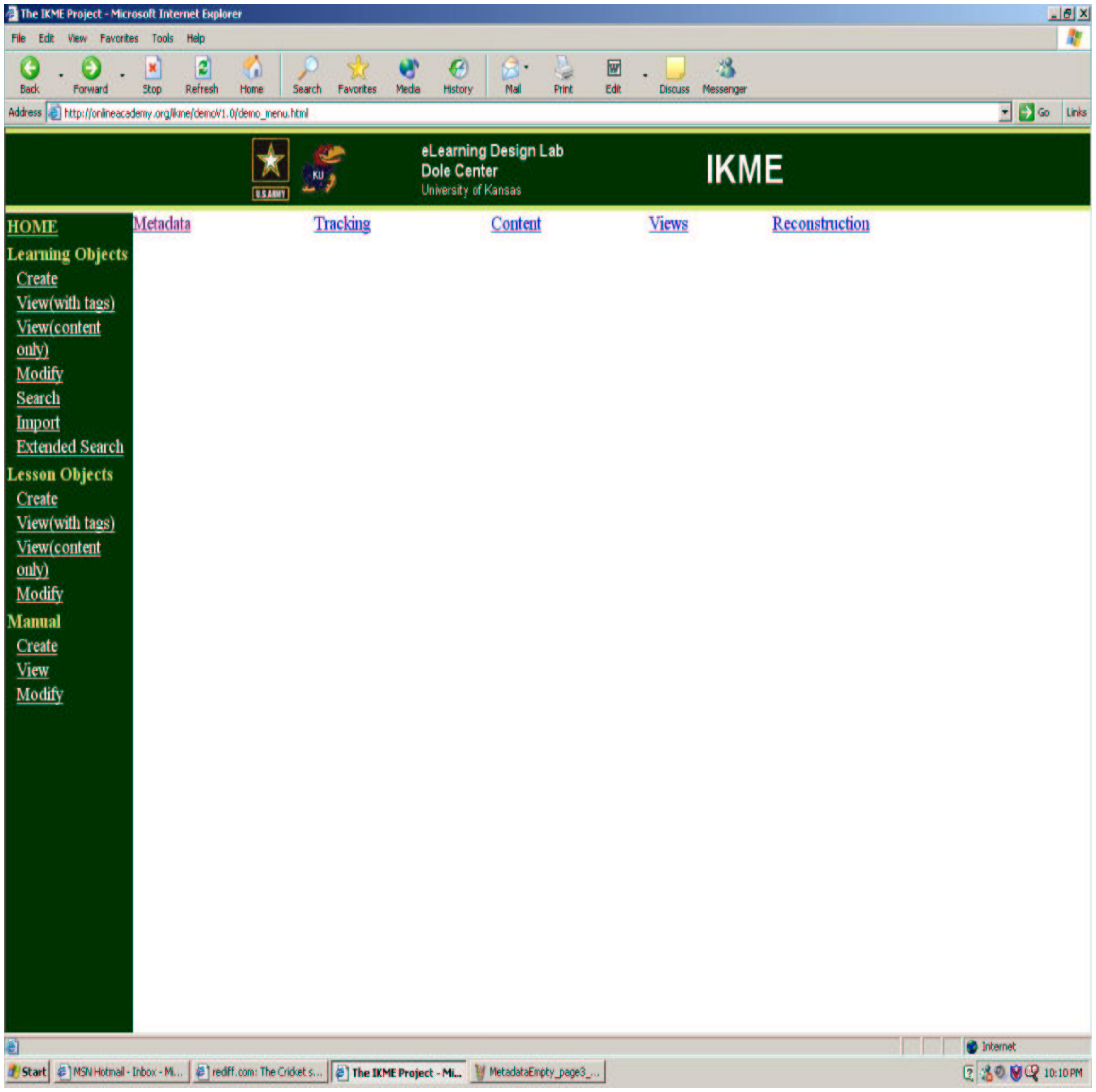
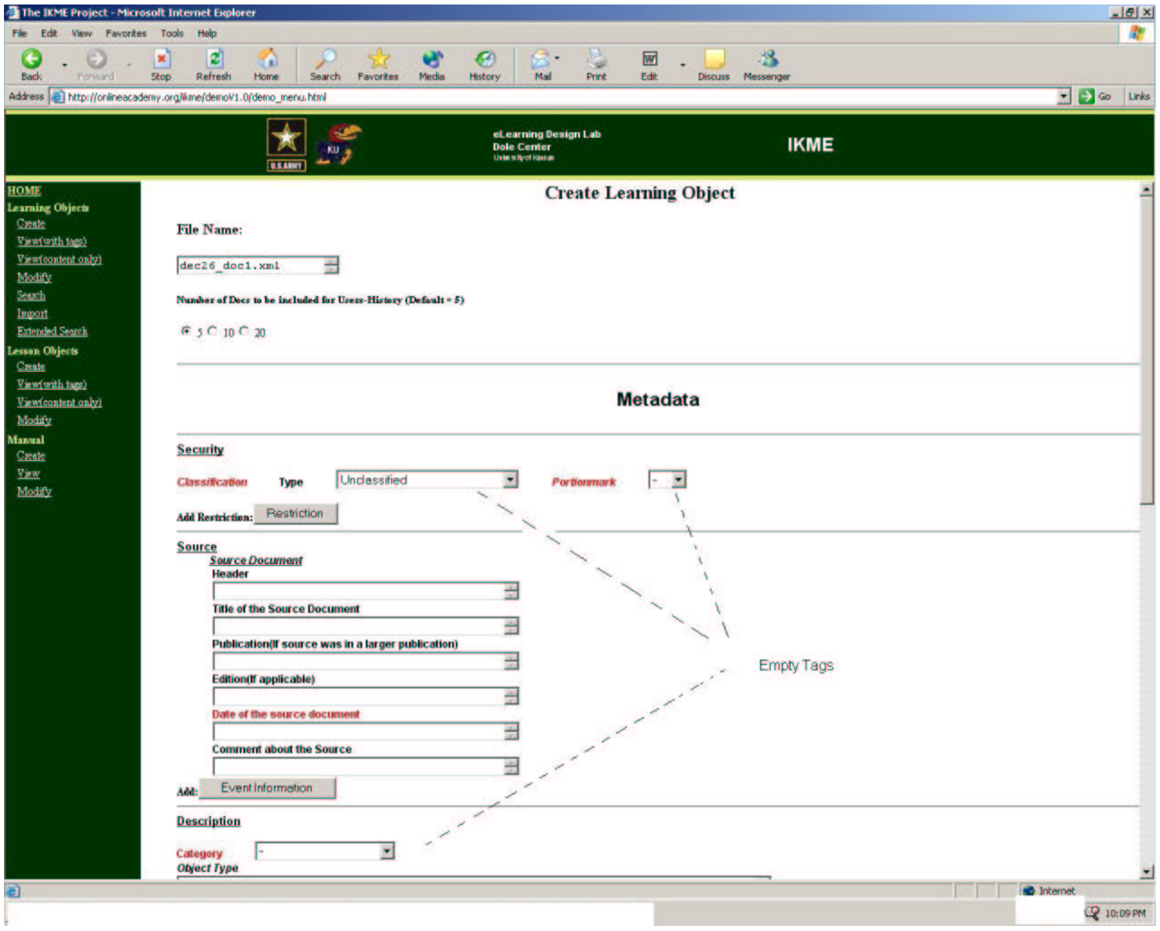


Figure 5.3 Access the high level fields of the Learning Object

5.4 Metadata – At first login, Sample Template spits out blank values

After the user selects the field, all its sub-fields are extracted from the Sample Template and displayed on to the screen (with the help of XSLT). At the first login, the user finds empty values for the enumerated fields, since the Sample Template has no default values assigned initially. The Schema in this version has a total of seven enumerated fields.



http://onlineacademy.org/ikme/demoV1.0/demo_menu.html

Create Learning Object

File Name:

dec26_doc1.xml

Number of Docs to be included for Users-History (Default = 5)

[✓ 5] 10 20

Metadata

Security

Classification Type: Unclassified Portionmark: -

Add Restriction:

Source

Source Document

Header

Title of the Source Document

Publication(If source was in a larger publication)

Edition(If applicable)

Date of the source document

Comment about the Source

Add:

Description

Category: -

Object Type

Figure 5.4.1 blank enumerated fields of Metadata

More enumerated fields.

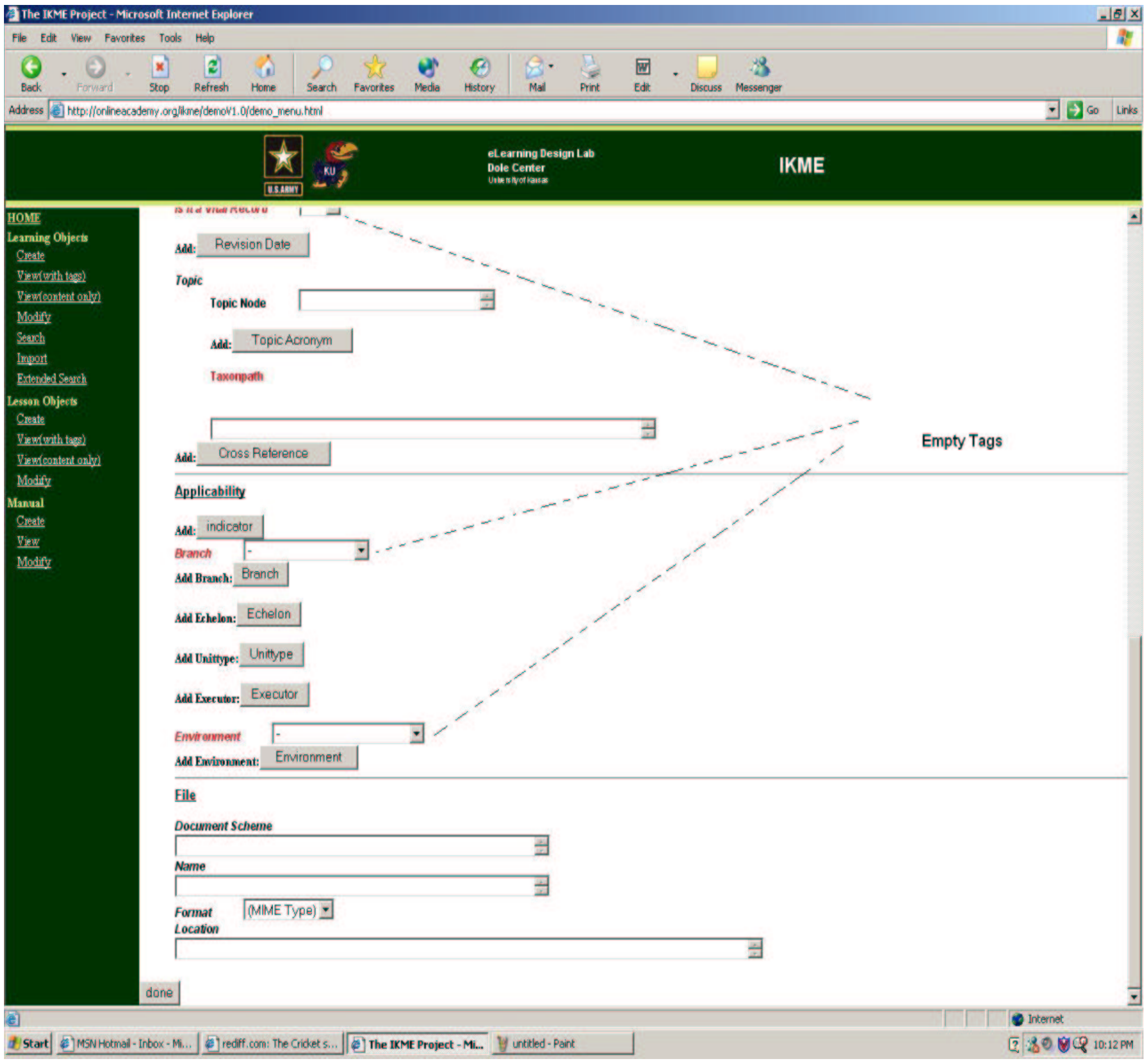
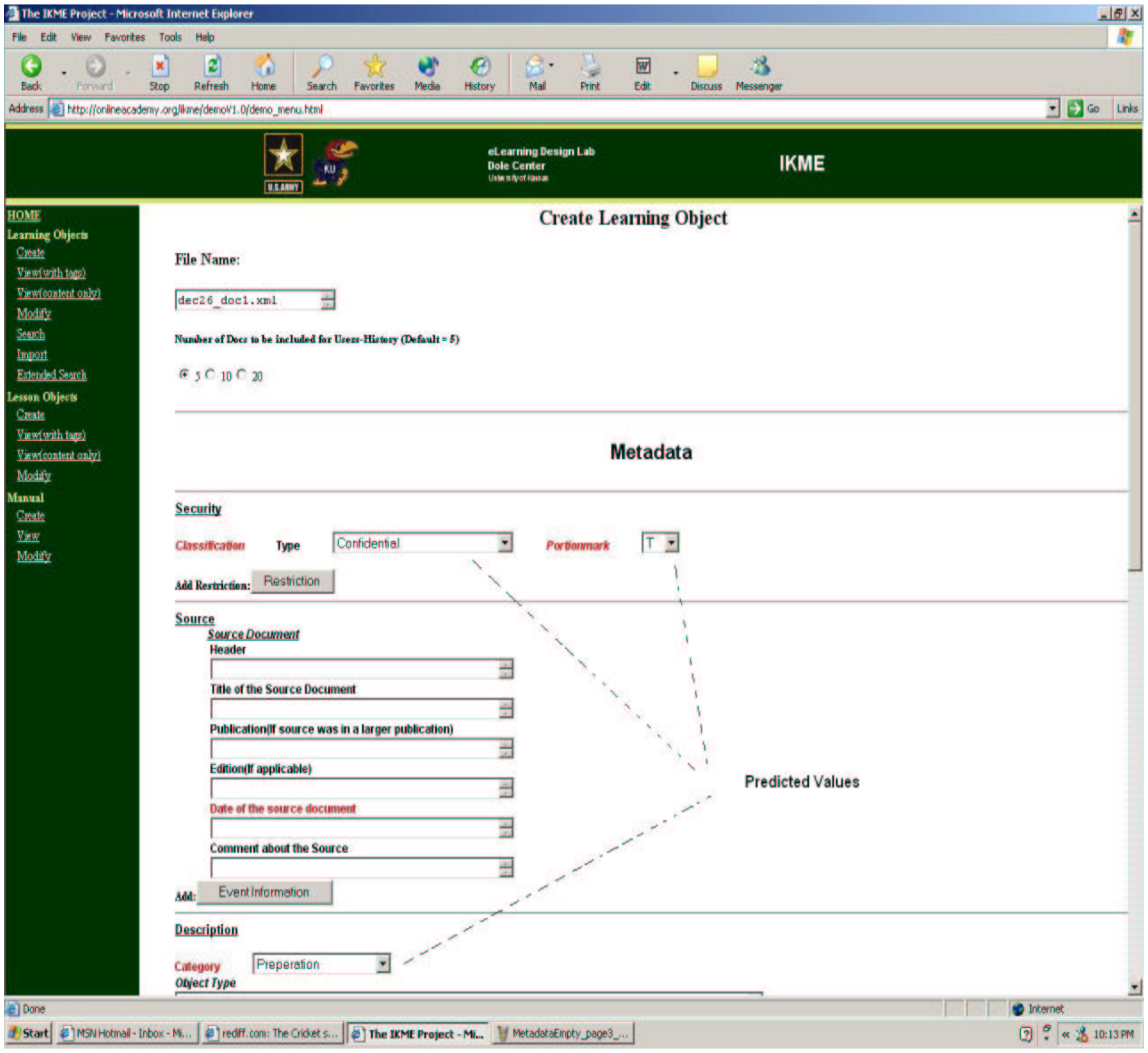


Figure 5.4.2 more blank enumerated fields of Metadata

5.5 Metadata – Next logins, Sample Template is customized to user's liking

The next time they create a Learning Object, the user will find default values for the enumerated fields based on their previously created Learning Objects. It can be noted that enumerated fields like TYPE, PORTIONMARK etc are already filled in.



Address http://onlineacademy.org/ikme/demoV1.0/demo_menu.html

Create Learning Object

File Name: dec26_doc1.xml

Number of Docs to be included for Users-History (Default = 5)

[✓ 5] 10 20

Metadata

Security

Classification Type Confidential Portionmark T

Add Restriction: Restriction

Source

Source Document

Header

Title of the Source Document

Publication(If source was in a larger publication)

Edition(If applicable)

Date of the source document

Comment about the Source

Add: Event Information

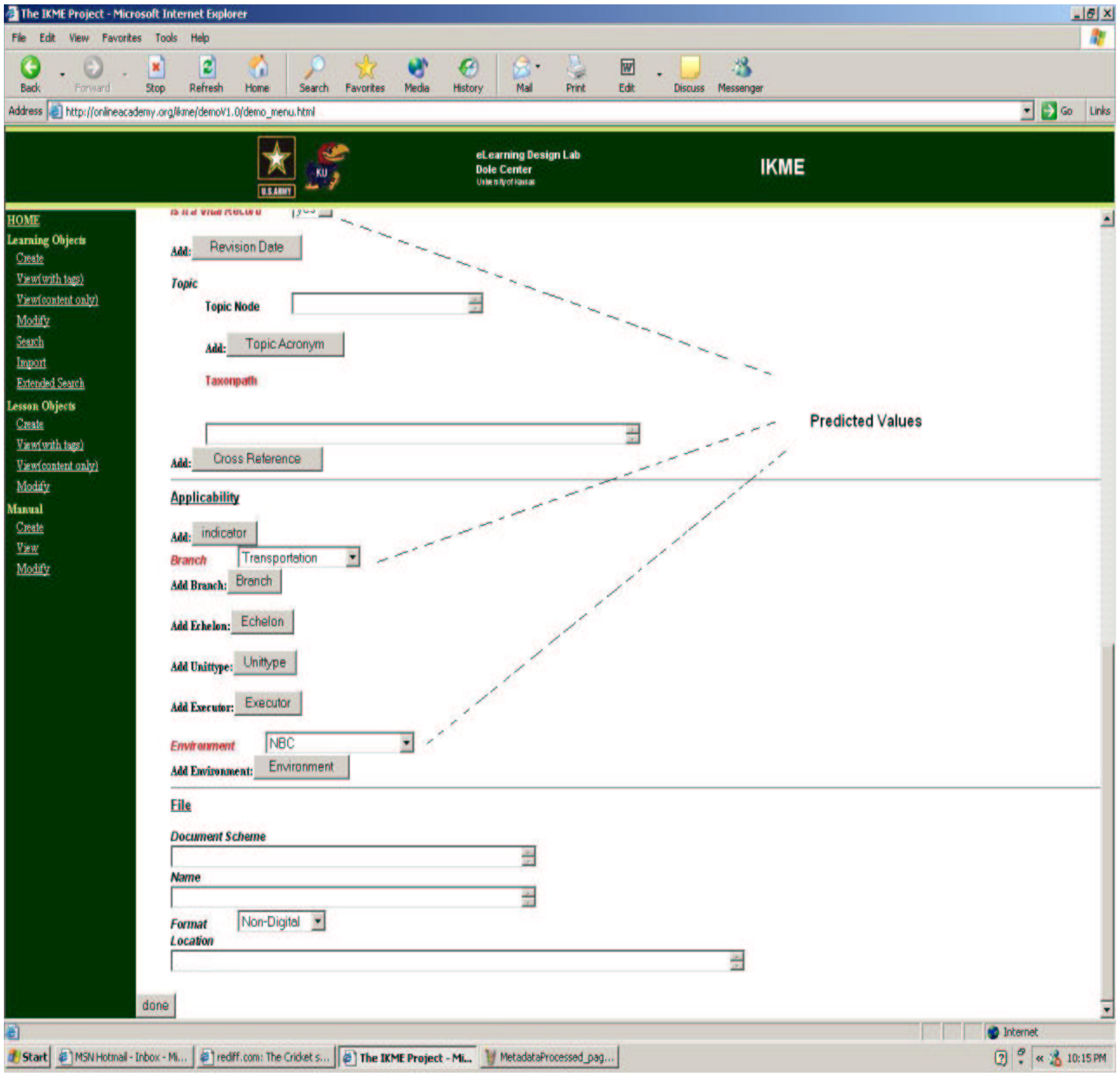
Description

Category Preperation

Object Type

Figure 5.5.1 enumerated fields of Metadata with default values

More enumerated fields with default values



Branch: Transportation

Environment: NBC

Format: Non-Digital

Figure 5.5.2 Sub-Fields of Metadata with default values

6. Conclusions and Future Work

We have developed a system that learns a user's preferences. All the initial requirements of the project have been successfully fulfilled. This project can be used as a platform for building more sophisticated statistical patterns for learning the user's preferences. Below some advantages, limitations, and the future work of this approach are discussed.

6.1 Benefits and Costs

The benefits of this approach are that, in determining the history of the user, nothing has to be put on the client's machine. The user can thus work from any machine. There is no redundancy in storing and defining data, hence there is no wastage of space. Moreover users have the flexibility to choose the number of recent documents to be considered for their history.

In this approach, the user must login each time to create Learning Objects. The main cost comes from accessing and updating the database. Each time the user creates a new Learning Object, the database has to be queried M times, where M is the number of enumerated fields in the XML Schema. However, in our case, since there are only seven enumerated fields, there is no difference observed in the performance time.

6.2 Future work

One goal would be to extend the functionality of the login system to support multiple users with individual logins. Doing so requires a mechanism that makes use of the login information and information in the ‘actor tag’, a child tag of ‘tracking’. We should also extend the functionality of this version to learn the user’s preferences on the *optional* enumerated fields. Doing so requires a change in the template file.

The next step would be to investigate how much history is needed to get the best default values, i.e., what *N* is best, where *N* is the number of recent Learning Objects created by the user. We should also investigate methods of weighting recent values more highly than older values for quicker adaptation.

7. References & Resources

The following are the references for all the resources that were used in this project. The latest date that these websites were accessible was Friday, January 22, 2004.

- Demo *URL*
  http://onlineacademy.org/ikme/demoV1.0/demo_menu.html
- The Comprehensive Perl Archive Network – Excess of *PERL* modules
  http://www.cpan.org/
- *XML:DB* Initiative
  http://www.xml.com/pub/a/2001/10/31/nativexmldb.html
- eXist Home Page
  http://exist-db.org/index.html
- Paper on eXist: An Open Source Native XML Database
  http://exist-db.org/webdb.pdf
- Syntax of queries to the eXist database from applications
  http://exist.sourceforge.net/devguide.html
- *XML::Twig* Home Page
  http://xmltwig.com/xmltwig/
- *RPC::XML* Home Page
  http://www.blackperl.com/RPC::XML/
- *XML-RPC* Home Page
  http://www.xmlrpc.com/
- *Introduction to XML:: XML-Future of the web*
  http://www.acm.org/crossroads/xrds6-2/future.html
- Comparison between *XML-RPC* & *SOAP*
  http://weblog.masukomi.org/writings/xml-rpc_vs_soap.htm
- Intro to *Native XML databases*
  http://www.xml.com/lpt/a/2001/10/31/nativexmldb.html
- *XML* for Data: *Native XML databases*
  http://www-106.ibm.com/developerworks/xml/library/x-xdnat.html

- About Learning Objects
  http://www.uwm.edu/Dept/CIE/AOP/learningobjects.html
  http://www.eduworks.com/LOTT/tutorial/learningobjects.html