Bo Yao. Adding Vocabularies to the HIVE demo at RENCI.org: Step-by-step Processes. A Master's Paper for the M.S. in I.S degree. April, 2014. 36 pages. Advisor: Jane Greenberg

The demand for high quality metadata is growing in response to the massive growth of digital information and data. Traditionally, keywords and other metadata have been created by authors and professional indexers. However, manual metadata generation is increasingly insufficient, due to the digital information explosion and the growth of multidisciplinary digital environments. There is a growing need to develop, study, and advance knowledge about tools that automatically generate metadata, particularly tools that reach across different disciplines.

This study addresses this need by introducing "HIVE" which supports automatic metadata generation by drawing descriptors from multiple SKOS (Simple Knowledge Organization Systems) encoded controlled vocabularies. First, HIVE's core components and key concepts are defined and discussed. Next, this study describes the step-by-step process of adding new controlled vocabularies into the HIVE demo at RENCI.org. The paper concludes by showing the result of the step-by-step process and addressing further steps of the HIVE project.

Headings:

Controlled Vocabularies

Metadata

Automatic Keyphrases Extraction

RDF/SKOS

HIVE

ADDING VOCABULRIES TO THE HIVE DEMO AT RENCI.ORG: STEP-BY-STEP PROCESSES

by
Bo Yao

A Master's paper submitted to the faculty
of the School of Information and Library Science
of the University of North Carolina at Chapel Hill
in partial fulfillment of the requirements
for the degree of Master of Science in
Information Science.

Chapel Hill, North Carolina

April 2014

Approved by

_____

Jane Greenberg

Table of Contents

INTRODUCTION

According to the EMC-sponsored IDC Digital Universe study, "Extracting Value from Chaos" in 2001, the world's information is more than doubling every two years, with a colossal 1.8 zettabytes to be created and replicated in 2011, which is growing faster than Moore's Law. The high demand of information together with the information explosion heightened the position of metadata. Today's metadata activities are unprecedented because they extend beyond the tradition library environment in an effort to deal with the Web's exponential growth (Greenberg, 2003).

Due to this information explosion, traditional ways of manual indexing and metadata generation by professional indexers or authors are to be described as costly, time-consuming, inefficient, and inconsistency. Furthermore, in today's world, more and more multidisciplinary digital information are generated, which makes manual metadata generation and indexing even more difficult. Metadata generation is inefficient, with automatic applications not being fully employed, and often the same metadata is being generated via humans in more than one setting (Greenberg, 2009). The need for developing the techniques to effectively automatically extract keywords and generate metadata across different discipline is increasing rapidly.

This study includes a literature review introduced some key concepts and existing researches on automatic metadata generation. Following that is an introduction about a

system called “Helping Interdisciplinary Vocabulary Engineer” or “HIVE”, which supports automatic metadata generation by drawing descriptors from multiple Simple Knowledge Organization System (SKOS) encoded controlled vocabularies, and also provide a step-by-step process of how to add new vocabulary into the HIVE demo hosted on RENCI.org. The study concludes by showing the result of the step-by-step process and addressing further steps of the HIVE project.

LITERATURE REVIEW

1.1 Semantic web

In addition to the classic “Web of documents” W3C is helping to build a technology stacks to support a “Web of data”. The ultimate goal of the “Web of data” is to enable computers to do more useful work and to develop systems that can support trusted interactions over the network. The term “Semantic Web” refers to W3C’s vision of the Web of linked data. Semantic Web technologies enable people to create data stores on the Web, build vocabularies, and write rules for handling data. Linked data are empowered by technologies such as RDF, SPARQL, OWL, and SKOS.

According to W3C, the Semantic Web is a Web of data — of dates and titles and part numbers and chemical properties and any other data one might conceive of. The collection of Semantic Web Technologies includes Resource Description Framework (RDF), Web Ontology Language (OWL), Simple Knowledge Organization System (SKOS), SPARQL Protocol and RDF Query Language (SPARQL), etc. provides an environment where application can query that data, draw interfaces using vocabularies, etc. (W3C Standards)

The Semantic Web Stack below illustrates the architecture of the Semantic Web.

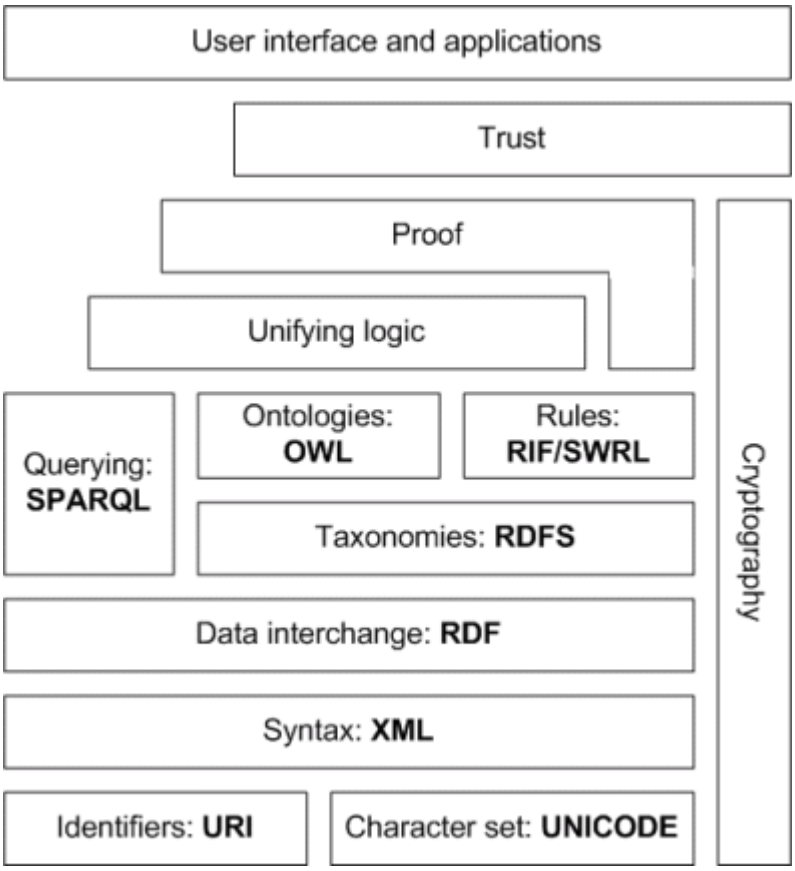

Figure 1 *Semantic Web Stack (from semanticweb.org)*

1.2 Controlled vocabulary

According to Sarah, Campbell and Beetham (2005), a controlled vocabulary is a vocabulary consisting of a "prescribed list of terms or headings each one having an assigned meaning". The way a controlled vocabulary defines the relationships between these terms or headings will vary in degree of complexity according to the purpose of the vocabulary, from simple alphabetically arranged flat lists to ontologies with richly defined relationships.

One example of controlled vocabularies is MeSH which stands for Medical Subject Headings. It was created by the National Library of Medicine to index biomedical

journals and books. There are about 25,000 subject headings arranged in a hierarchy. And a heading can appear in multiple locations in the hierarchy.

1.3 RDF

According to W3C, Resource Description Framework (RDF) is a standard model for data interchange on the web. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed. RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a "triple"). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications (W3C, 2014-02-25).

RDF is not designed for being displayed to people, instead, it is designed to be read and understood by computers. RDF documents are written in Extensive Makeup Language (XML), and the XML language used by RDF is called RDF/XML. By using XML, RDF information can easily be exchanged between various types of computers using different operating systems and application languages.

1.4 SKOS

SKOS, Simple Knowledge Organization System, is a formal language for representing a controlled structured vocabulary. By "controlled structured vocabulary" we mean to include (Miles & Pérez-Agüera, 2009):

• **Thesauri** broadly conforming to the ISO 2788:1986 guidelines such as the UK Archival Thesaurus (UKAT, 2004), the General Multilingual Environmental Thesaurus (GEMET), and the Art and Architecture Thesaurus (AAT) (ISO 5964:1985).

• **Classification schemes** such as the Dewey Decimal Classification (DDC), the Universal Decimal Classification (UDC), and the Bliss Classification (BC2).

• **Subject heading systems** such as the Library of Congress Subject Headings (LCSH) and the Medical Subject Headings (MeSH).

Those "controlled structured vocabularies" share a similar structure, and are used in similar applications. SKOS captures much of this similarity and makes it explicit, to enable data and technology sharing across diverse applications. The SKOS data model provides a standard, low-cost migration path for porting existing knowledge organization systems to the Semantic Web. SKOS also provides a lightweight, intuitive language for developing and sharing new knowledge organization systems. It may be used on its own, or in combination with formal knowledge representation languages such as the Web Ontology language (Miles & Bechhofer, 2009).

SKOS uses RDF to represent knowledge. By using it, the information can be passed between computer applications in an interoperable way. Using RDF also allows knowledge organization systems to be used in distributed, decentralized metadata applications. Decentralized metadata is becoming a typical scenario, where service providers want to add value to metadata harvested from multiple sources (W3C, 2012).

1.5 Keyphrase Extraction Algorithm (KEA)

Keyphrases are widely used in both physical and digital libraries as a brief, but precise, summary of documents (Medelyan & H.Witten, 2007). They describe the content

of single document and provide a kind of semantic metadata that is useful for a wide variety of purpose. The task of assigning keyphrases to a document is called keyphrase indexing. Keyphrase Extraction Algorithm (KEA) is an algorithm for extracting keyphrases from text documents. It can be either used for free indexing or for indexing with a controlled vocabulary. It is implemented in Java and is platform independent. It is an open-source software developed by University of Waikato at New Zealand and distributed under the GNU general Public License.

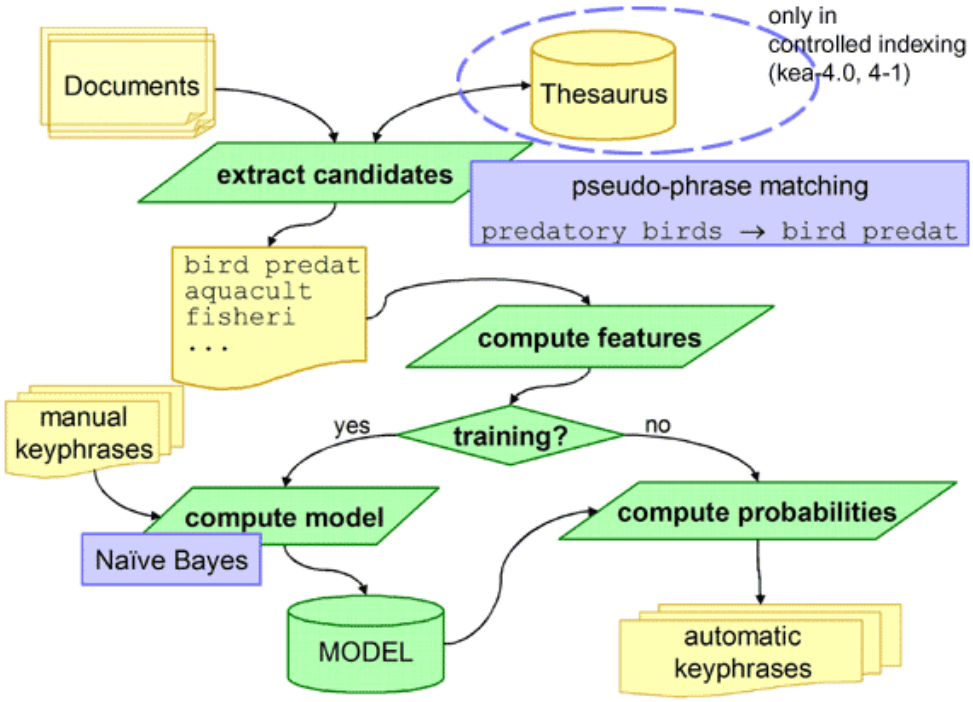

Figure 2 *KEA method diagram (from http://www.nzdl.org/Kea/description.html)*

KEA identifies candidate keyphrases using lexical methods, calculates feature values for each candidate, and use a machine-learning algorithm to predict which candidates are good keyphrases (Witten, Paynter, Frank, Guitwin and Newill-Maning, 1999).

Basically, KEA's indexing algorithm has two stages (Medelyan & H.Witten, 2007):

1. Candidate identification: identifies candidate terms that related to the document's content, including ones that appear verbatim as phrase in the document.

2. Keyphrase selection: uses a model to identify the most significant terms based on certain properties or "features". This involves first learning a model based on training data, and then applying it to test documents.

As Figure 2 shows, first, KEA gets a directory name and processes all documents in this directory that have the extension ".txt". The default language and the encoding is set to English, but this can be changed as long as a corresponding stopword file and a stemmer is provided.

Second, if a vocabulary is provided, Kea matches the documents' phrases against this file. For processing SKOS files stored as RDF file, KEA uses the Jena API, which is a free and open source Java framework for building Semantic Web and Linked Data applications.

Then KEA starts the Candidates Extracting process. It first cleans the input text, then identifies candidates, and finally stems and case-folds the phrases.

Input Cleaning

Modifications made here include:

- Punctuation marks, brackets, and numbers are replaced by phrase boundaries;
- Apostrophes are removed;
- Hyphenated words are split in two;
- Remaining non-token characters are deleted, as are nay tokens that do not contain letters

The result is a set of lines, each a sequence of tokens containing at least one letter.

Candidates identification

Here KEA extracts n-grams of a predefined length (e.g. 1 to 3 words) that do not start or end with a stopword. In controlled indexing, it only collects those n-grams that match thesaurus terms. If the thesaurus defines relations between non-allowed terms (non-descriptors) and allowed terms (descriptors), it replaces each descriptor by an equivalent non-descriptor.

Stemming and case-folding

The final step in determining candidate phrases is to casefold all words and stem them using the iterated Lovins method. This involves using the classic Lovins stemmer to discard any suffix, and repeating the process on the stem that remains until there is no further change.

After extracting candidates, KEA starts to calculate four features for each candidate phrase. The features are:

- **TF-IDF** (Term Frequency – Inverse Document Frequency) is a measure describing the specificity of a term for this document under consideration, compared to all other documents in the corpus, candidate phrases that have high TF*IDF value are more likely to be keyphrases.
- **First occurrence** is computed as the percentage of the document preceding the first occurrence of the term in the document. Terms that tend to appear at the start or at the end of a document are more likely to be keyphrases.
- **Length** of a phrase is the number of its component words. Two-word phrases are usually preferred by human indexers.

- **Node degree** of a candidate phrase is the number of phrases in the candidate set that are semantically related to this phrase. This is computed with the help of the thesaurus. Phrases with high degree are more likely to be keyphrases.

After features calculation, KEA starts to build the model. It needs to create a model that learns the extraction strategy from manually indexed documents. This means, for each document in the input directory there must be a file with the extension ".key" and the same name as the corresponding document. This file should contain manually assigned keyphrases, one per line.

Given the list of the candidate phrases, Kea marks those that were manually assigned as positive example and all the rest as negative examples. By analyzing the feature values for positive and negative candidate phrases, a model is computed, which reflects the distribution of feature values for each phrase. The technique KEA uses here is Naïve Bayes.

When the model was built, KEA can start to extract keyphrases from a new document. KEA determines candidate phrases and feature values, and then applies the model build during training. The model determines the overall probability that each candidate is a keyphrase, and then a post-processing operation selects the best set of keyphrases (Witten, Paynter, Frank, Guitwin and Newill-Maning, 1999).

1.6 MAUI

Another algorithm for topic indexing is called MAUI (Multi-purpose automatic topic indexing) which can be used for the same tasks as KEA, but offers additional features. Tasks MAUI performs include the following:

- Term assignment with a controlled vocabulary (or thesaurus)

- Subject indexing
- Topic indexing with terms from Wikipedia
- Keyphrases extraction
- Terminology extraction
- Automatic tagging
- Semi-automatic topic indexing

According to MAUI's wiki site, MAUI implements a two-stage algorithm for performing its tasks automatically. The first stage is called "candidate generation" which identifies candidate topics in a given document. The second stage is called "filtering", which analyzes the features of the candidate topics and filters out the most significant ones. Features MAUI utilizes are:

- Frequency statistics, such as TF, IDF, TF*IDF;
- Occurrence positions in the document;
- Keyphrasesness;
- Semantic relatedness;

All these make MAUI very similar with KEA. Actually, MAUI builds on KEA. Major parts of KEA become parts of MAUI without any further modifications. Other parts, such as feature computation, were extended with new elements.

Besides KEA, there are some other software inside MAUI. MAUI uses the machine learning toolkit Weka for creating the topic indexing model from documents with topics assigned by people and applying to new documents. MAUI also uses Jena library to make it applicable for topic indexing with many kinds of controlled vocabularies. In order to access data on Wikipedia, MAUI utilized Wikipedia miner which converts regular

Wikipedia dumps into MySQL database format and provides an object-oriented access to parts of Wikipedia.

ABOUT HIVE

Imagine we have some research data about how the solar variation influences the production of corn in different geographic regions, and we need to create metadata or indexing terms for this research data. Obviously, the data are interdisciplinary, covering astronomy, agriculture, biology, geography and environmental science. So, in order to index it, we need to draw standard indexing vocabularies from every discipline the data cover. The challenge here is, even though specialized vocabularies covering these fields are available on the Web, it is very time consuming and costly to search every vocabulary individually and finally generate the best indexing term.

This challenge simulates the advent of HIVE, which stands for Helping Interdisciplinary Vocabulary Engineering. HIVE is both a model and a system that supports automatic metadata generation by drawing descriptors from multiple Simple Knowledge Organization System (SKOS)-encoded controlled vocabularies (Greenberg, Losee, Perez Aguera, Scherle, White and Willis, 2011). HIVE allows a person to simultaneously search multiple vocabularies cross different disciplines. The project is led by a research team at the University of North Carolina at Chapel Hill's School of Information and Library Science (SILS) Metadata Research Center (MRC), in collaboration with the National Evolution Synthesis Center (NESCent) in Durham, N.C. The project is funded by the Institute of Museum and Library Services.

HIVE implements the technological infrastructure to store millions of concepts from different vocabularies and make them available on the web by a simple HTTP call.

Vocabularies can be imported in HIVE using SKOS RDF/XML format (Greenberg, Losee, Perez Aguera, Scherle, White and Willis, 2011).

HIVE features can be used in three ways:

- HIVE Web: Google Web Toolkit (GWT)-based web application
- Core API: Java SE API for programmatic access
- REST Service: Representational State Transfer (REST)-Based service for programmatic access

HIVE support the following functions:

- Conversion of various vocabularies to SKOS
- Browsing and searching SKOS vocabularies (Concept Browser)
- Automatic controlled indexing using KEA++, MAUI, and simple Lucene-based indexers.

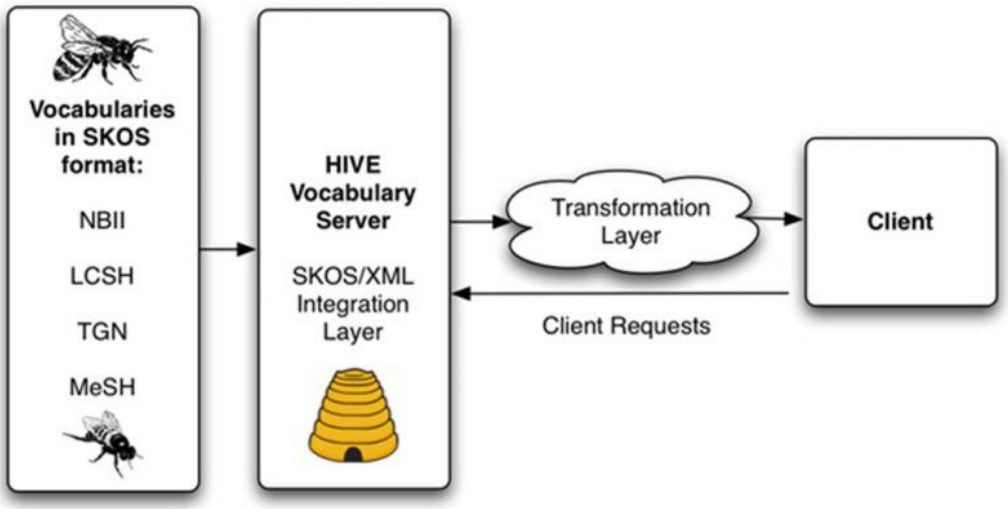

Figure 3 *The HIVE model (from http://sils.unc.edu/news/2011/hive)*

The HIVE model above well illustrates how HIVE works. First, HIVE need to accept requests from clients. After get clients' requests, the HIVE vocabulary server will go to different existing vocabularies to draw appropriate terms back to the HIVE. Then, HIVE

will transfer the well-formatted result to the client side. It is like bees gathering pollen from several flowers and bring it back to the hive. After some process, the hive finally returns honey to us.

RESEARCH OBJECTIVE

The research objectives for this study include:

1. To study and learn the process of preparing terminologies in SKOS for integration into HIVE.
2. To learn about the scope and extent of ontologies and the semantic web.
3. Develop a step-by-step process of loading new vocabularies into HIVE.

DESCRIPTION OF THE WORK

The HIVE demo this study worked on is hosted on Renaissance Computing Institute (RENCI)'s server. This demo can be accessed using any web browsers at http://hive.renci.org:8080/home.html.

The "Home" tab tells users some general information about the vocabularies exists in the demo, such as vocabulary name, number of concepts, number of relationships, and last update time. "Concept Browser" allows us to browse the terms in each vocabulary. Users can choose to view each term in SKOS format also. "Indexing" tab provides the function to automatically extract concepts from a document or URL based on selected vocabularies. The result will be concepts from the vocabulary source we selected.

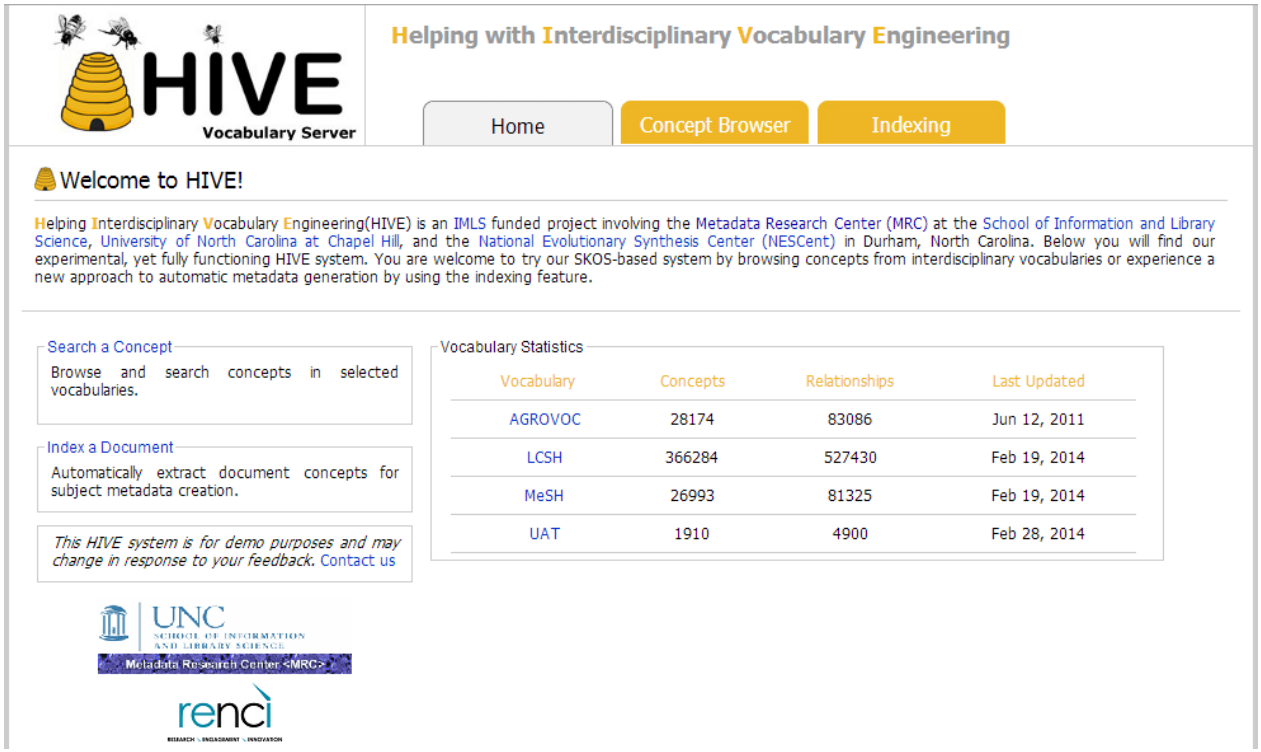

Figure 4 *HIVE.RENCI demo*

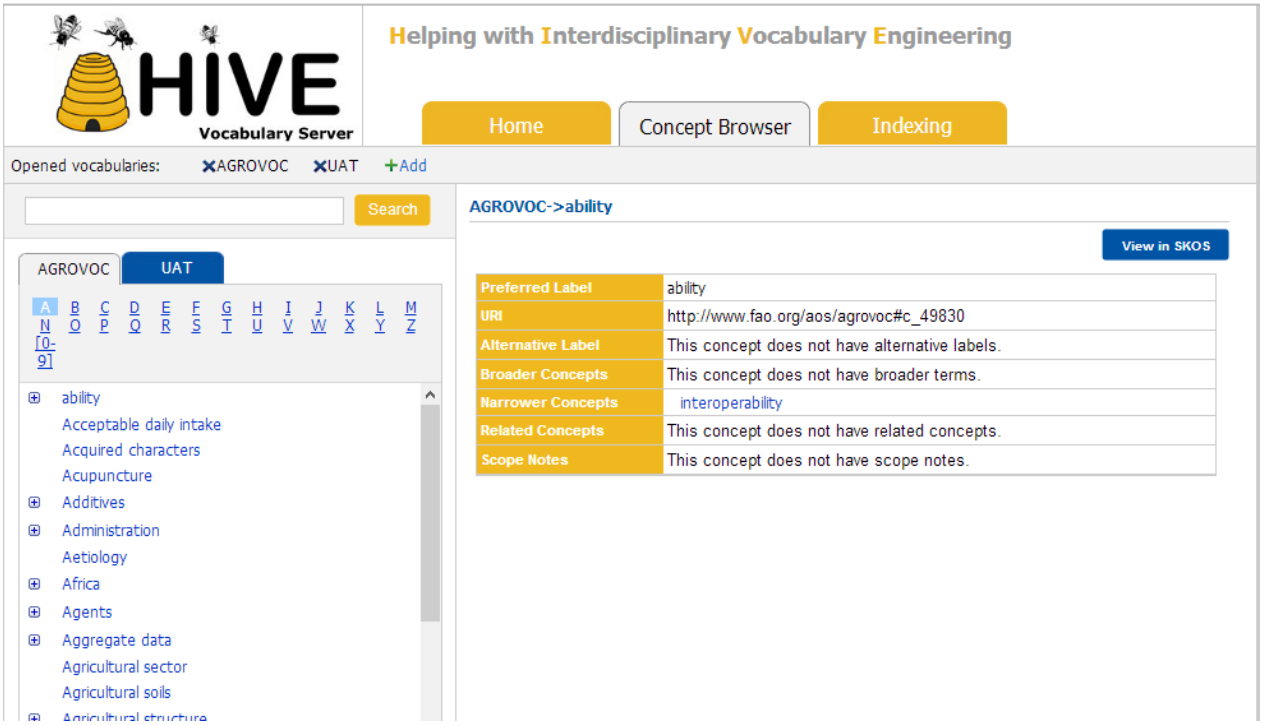

Figure 5 *Concept Browser*

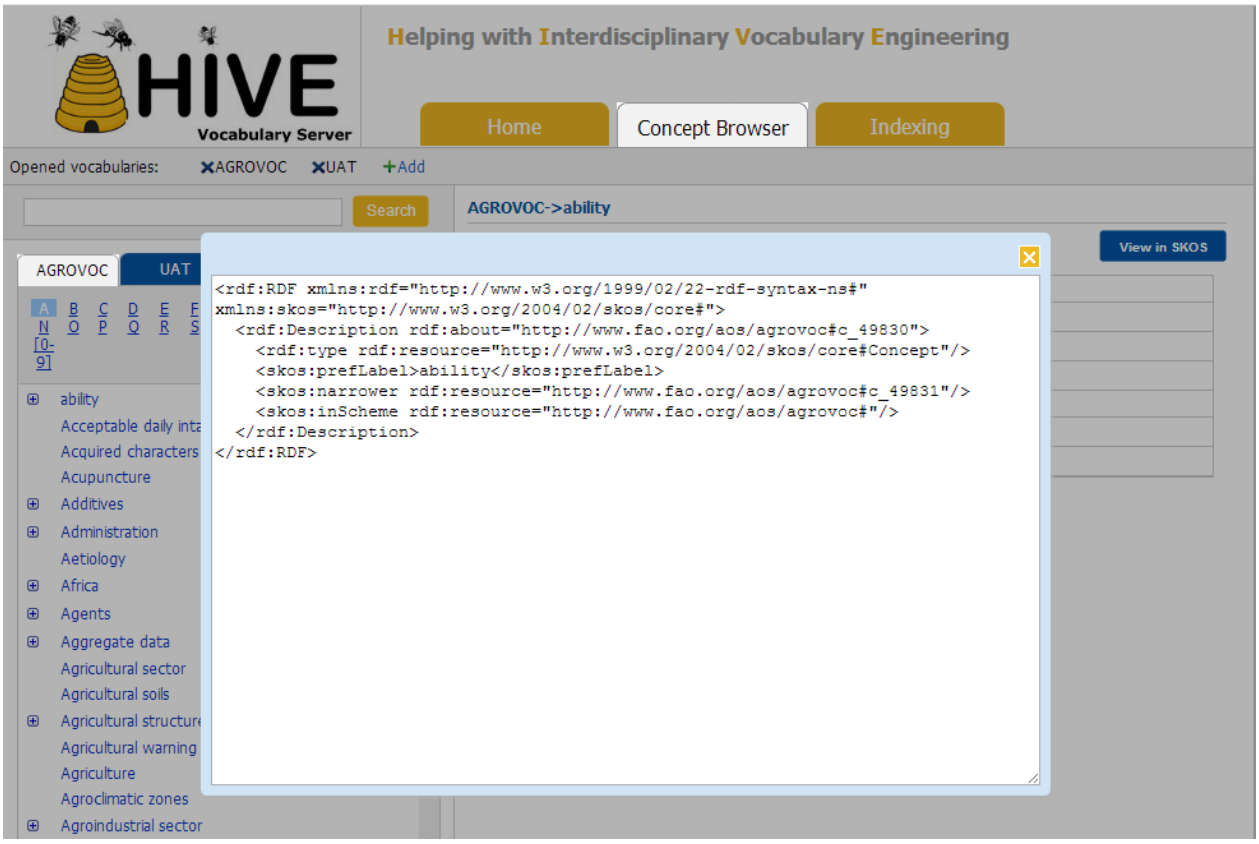

Figure 6 *View concepts in SKOS*

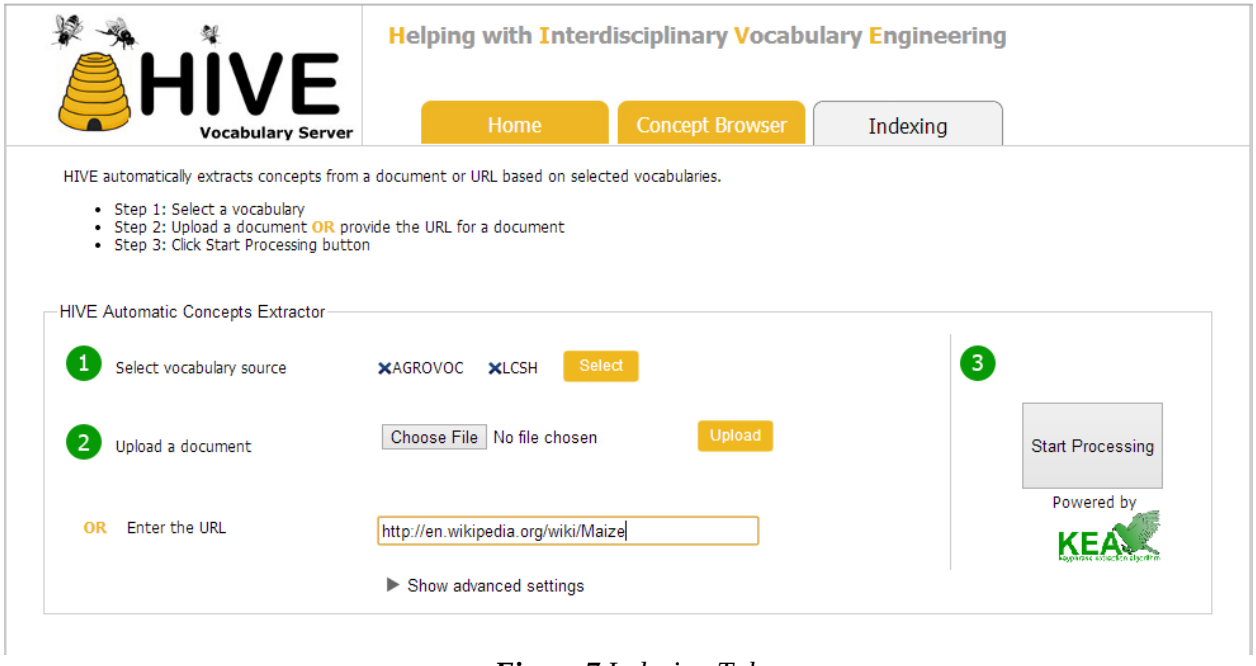

Figure 7 *Indexing Tab*

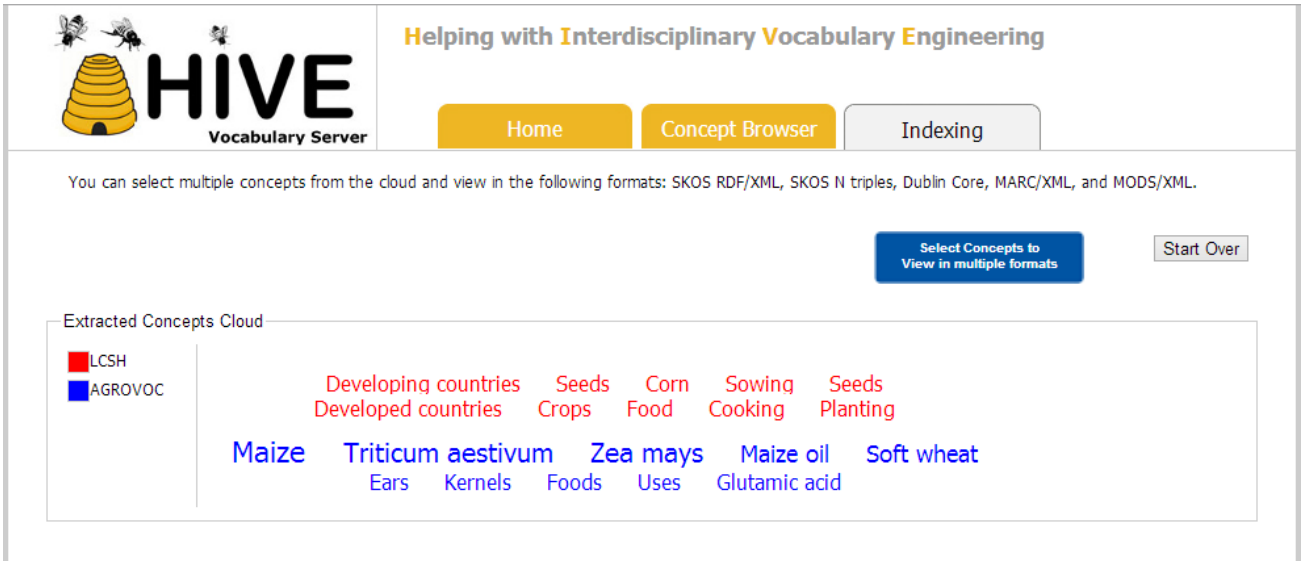

Figure 8 *Indexing result (From NESCent HIVE Demo)*

Before adding new vocabulary into it, there are already four vocabularies there. We are going to load another one into the server. The new vocabulary we are going to load is called "STW Thesaurus for Economics", which contains terms on any economics subject. It has more than 6,000 standardized subject headings and about 19,000 entry terms to support individual keywords.

In order to load new vocabularies into the demo, we use SSH Secure Shell Client to connect to the server and finish our job. Secure Shell is a cryptographic network protocol for secure data communication, remote command execution, and other secure network services.

After click the "Connect" button in SSH, a window will pop up asking for connection information. Fill in those fields with the information below and replace my username with yours.

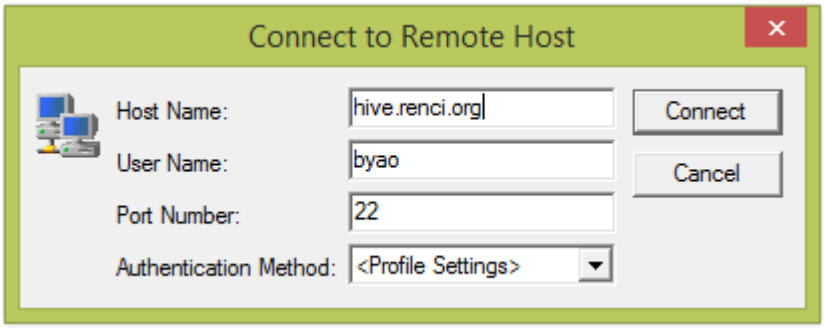

Figure 9 *Connect to Remote Host Window*

After correctly entering password, we are now on the server side of the HIVE.RENCI demo. "**`whoami`**" command will return our username to tell us who we currently login as.

The first step we should do is to stop the Tomcat server. This step is very important, we must stop Tomcat before running AminVocabularies tool. Otherwise, we will got an error when running the tool. We will introduce AdminVocabularies tool later. The command to stop Tomcat is "**`sudo service tomcat6 stop`**". "`sudo`" stands for "super user do". It will promotes you for your personal passwords and confirms your request to execute a command. When entering our password, we will not see any "•" or "*" like normal password text field. Just type in our password and press "Enter". In the console, it will look like this:

```
[username@hive ~]$ sudo service tomcat6 stop
[sudo] password for username:
Stopping tomcat6:
[  OK  ]
[username@hive ~]$
```

After we got "`[ OK ]`", we can refresh the tab of HIVE.RENCI demo in our browser. The browser will tell us it failed to connect to the URL, which means we successfully stopped Tomcat.

For the rest of the task, we must work as the Tomcat user instead of ourselves. Type "**`sudo –u /bin/bash`**" in SSH. This command will create a bash shell for the tomcat user. Also, it will promotes your for your password. After entering the password, if we type in "**`whoami`**", the system will tell us we are "tomcat" instead of ourselves. The command and result will be:

```
[username@hive ~]$ sudo -u tomcat /bin/bash
[sudo] password for username:
bash-4.1$ whoami
tomcat
```

Next, we should import RDF file which contains our new vocabulary into the right directory.

First, using "**`cd`**" command to change current directory to where hive data are. In this demo, all RDF files are under "/opt/hiveData/" folder. So we use "**`cd /opt/hiveData`**" to navigate to this directory. "**`ls`**" command will tell us what documents or directories are in our current directory. Currently, we have four folders "agrovoc", "lcsh", "mesh", and "uat" correspond to four vocabularies already exist in the demo. Like those four vocabularies, the new vocabulary should has its own directory also. The "**`mkdir stw`**" command will create a new directory called "stw" for our new vocabulary. After that, we can "**`cd`**" to "stw" folder. "**`pwd`**" command will return the "present working directory" for us. The commands and results will be similar to:

```
bash-4.1$ pwd
/home/username
bash-4.1$ cd /opt/hiveData/
bash-4.1$ ls
agrovoc         mesh            uat
lcsh
bash-4.1$ mkdir stw
bash-4.1$ ls
agrovoc       mesh            stw
lcsh            uat
bash-4.1$ cd stw
bash-4.1$ pwd
/opt/hiveData/stw
```

Then, we should import the new RDF file into "present working directory". The command we use here is "**wget**". Simply add the URL of the RDF file after "**wget**" command, the server will copy the file into current directory. If the vocabulary is in another format, it must be converted into SKOS before importing.

```
bash-4.1$ wget
http://urlofvocabulary.com/vocabulary/stw.rdf
--2014-03-25 00:14:22--
http://urlofvocabulary.com/vocabulary/stw.rdf
Resolving urlofvocabulary.com... 192.168.210.100
Connecting to urlofvocabulary.com |192.168.210.100|:80...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 15210220 (15M) [application/rdf+xml]
Saving to: stw.rdf?

100%[======================================>] 15,210,220
51.3M/s   in 0.3s

2014-03-25 00:14:22 (51.3 MB/s) - stw.rdf saved
[15210220/15210220]

bash-4.1$ ls
stw.rdf
```

Now, we have imported the RDF data file into the server space, but did not add it into the demo yet. The next step should be modify the default HIVE configuration and create configuration file for the new vocabulary. If we want to have KEA or MAUI indexing, we need create another directory called “stwKEA” under “stw”. Within “stwKEA”, there should be a “train” and a “test” directory containing a group of “.txt” and “.key” files which are used to train and test the algorithm. Another “data” directory should contain a list of stopwords. We can “cd” to “agrovoc” folder to see what those data look like. Currently, HIVE is transferring from KEA to MAUI and the process is not finished yet, and we do not have any training and testing data for the “stw” vocabulary, so there is no “stwKEA” directory here. Once we finished the transferring process and have the training and testing data, we can go through this process again to make the indexing work.

All configuration files are located in another directory so we should “**cd**” to there first. Type in “**cd /usr/share/tomcat6/webapps/ROOT/WEB-INF/conf**” to navigate to configuring files directory.

```
bash-4.1$ cd /usr/share/tomcat6/webapps/ROOT/WEB-INF/conf
bash-4.1$ ls
agrovoc.properties        lcsh.properties
lter.properties  tgn.properties
hive.properties           mesh.properties  uat.properties
itis.properties           nbii.properties
```

In this directory, we have a default HIVE configuration file called “hive.properties” and one “<vocabularyname>.properties” file for each vocabulary.

Type "**vi hive.properties**" to open "hive.properties" file in "vi editor". The file contains the name of the vocabularies the demo will load.

```
bash-4.1$ vi hive.properties
# Configured vocabularies
hive.vocabulary = agrovoc
hive.vocabulary = uat
#hive.vocabulary = itis
hive.vocabulary = lcsh
#hive.vocabulary = nbii
#hive.vocabulary = tgn
hive.vocabulary = mesh
hive.vocabulary = stw

# Path to location of scheme properties files. If not
set, defaults
# to the current path (where this properties file is
stored)
# hive.schemePath =

# Selected tagger. Possible values "kea" and "dummy"
#hive.tagger = kea
hive.tagger = maui
~
~
```

Any text after "#" will be regarded as comment and ignored when running this file. So, if we do not want to load a certain vocabulary, we can add a "#" at the beginning of its hive.vocabulary text line. For here, in order to let the system know we want to load "stw" vocabulary. We just need to add "**hive.vocabulary = stw**" into the file. After down with editing, press "Esc" button and then type "**:wq**" to save and quit editing. We'd better use "**vi <filename>**" command to open it again to make sure the change has been successfully saved. The "**:q!**" will exit the "vi editor" without saving any changes.

The "hive.properties" file is used by SKOSServer identify which vocabularies will be opened. When the system is running, HIVE will first read "hive.properties" file to

know what vocabulary it needs to load. Then, HIVE will read the configuration file for these vocabularies in the same directory as “hive.properties” file. So, we need to manually create a configuration file for the vocabulary with the paths to the files and indexes that will be generated by the HIVE import tools. The name of the configuration file for the vocabulary should be “<vocabulary name>.properties”.

Each configuration file must follow the format:

```
#Vocabulary data
name = <vocabulary name>
longName = <long vocabulary name>
uri = <url of the rdf file>
rdf_file = /opt/hiveData/<vocabulary name>/<vocabulary
name>.rdf

stemmerClass = kea.stemmers.PorterStemmer

#Sesame Store
store = /opt/hiveData/<vocabulary name>/<vocabulary
name>Store

#Lucene Inverted index
index = /opt/hiveData/<vocabulary name>/<vocabulary
name>Index

#H2 index
h2 = /opt/hiveData/<vocabulary name>/<vocabulary name>H2

#Autocomplete path
autocomplete = /opt/hiveData/<vocabulary
name>/autocomplete

#Dummy tagger data files
lingpipe_model =
/opt/hiveData/lingpipe/postagger/models/medtagModel

#KEA data
stopwords = /opt/hiveData/<vocabulary name>/<vocabulary
name>KEA/data/stopwords/stopwords_en.txt
kea_training_set = /opt/hiveData/<vocabulary
name>/<vocabulary name>KEA/train
kea_test_set = /opt/hiveDat/<vocabulary name>/<vocabulary
name>KEA/test
kea_model = /opt/hiveData/<vocabulary name>/<vocabulary
name>KEA/<vocabulary name>
maui_model = /opt/hiveData/<vocabulary name>/<vocabulary
name>KEA/maui
~
~
~
```

We can use “nano editor” to create the file and type in those texts. We just need to replace “<vocabulary name>” with “stw” and save this file. Or, if there already exists some configuration files for other vocabularies, we can take a shortcut by simply copy an existing file and modify it in “vi editor”. The command to copy a file is “**`cp <source filename> <target filename>`**”.

“Sesame Store”, “Lucene Inverted Index”, “H2 index”, and “Autocomplete path” indicate the path of those databases if we choose to create them when running the “AdminVocabularies” tool later.

“KEA data” indicates the location of “stopwords”, “kea training set”, “kea test set”, “kea model”, and “maui model”. Those information will be very important if we need to get the KEA or MAUI indexing working. Even if those directories does not exist now, we do not need to comment those lines out. Actually, if we comment them out, we will get an error when running “AdminVocabularies” tool. So, keep those lines, the system will skip those steps if they cannot find directories.

This is a relatively long file, so double check to make sure there is no typos before save and exit. If there is something wrong here when running the tool, we have to clean all files created by “AdminVocabularies” under “/opt/hiveData/stw” directory and run the tool again.

Now we successfully imported the RDF/SKOS file of our new vocabulary, modified “hive.properties” and created “stw.properties”. The next big step is to run the “AdminVocabularies” tool. The tool can run correctly only when Tomcat is shut down, that is why we use “**`sudo service tomcat6 stop`**” to stop Tomcat. Actually, we can import RDF/SKOS files, modify and create configuration files while Tomcat is

running, but we can only stop Tomcat service when we login as ourselves. We cannot stop it if we were Tomcat user. So, if we stopped Tomcat service at the very beginning before running "**`sudo –u tomcat /bin/bash`**" command, we do not need to log out Tomcat user to stop Tomcat and re-login again.

"AdminVaocabularies" takes three parameters:

1. Path to configuration directory
2. Name of the vocabulary
3. Active training option for KEA algorithm (optional, if we do not train our system, we cannot use automatic indexing classes)

The template of the command line to run "AdminVocabularies" is

```
java  -Djava.ext.dirs=<path to HIVE lib dir>
edu.unc.ils.mrc.hive.admin.AdminVocabularies -c <path to
directory with hive.properties> -v <vocabulary name> [-a
| -sldktmx]
```

Flags:

```
 -c <path>  Path to directory that contains
hive.properties
 -v <name>  Name of vocabulary to be initialized (e.g.,
agrovoc)
 -s  Initialize Sesame index
 -l  Initialize Lucene index
 -d  Initialize H2 database
 -k  Initialize KEA database
 -t  Train KEA
 -m  Train Maui
 -x  Initialize autocomplete
 -a  Initialize everything (equivalent of -sldktmxa)
```

For our demo, "<path to HIVE lib dir>" should be replaced with "/usr/share/tomcat6/webapps/ROOT/WEB-INF/lib ", "<path to directory with

hive.properties>" should be "/usr/share/tomcat6/webapps/ROOT/WEB-INF/conf", "<vocabulary name>" is "stw". For other flaggers, since we do not have training and testing data for KEA or MAUI, we can just initialize "autocomplete", "Lucene index", "Sesame index", and "H2 database". We can add more if we need. So, in this case, the command we will run is:

```
java -Djava.ext.dirs=/usr/share/tomcat6/webapps/ROOT/WEB-
INF/lib edu.unc.ils.mrc.hive.admin.AdminVocabularies -
c /usr/share/tomcat6/webapps/ROOT/WEB-INF/conf -v stw -x
-l -s -d
```

Before running this command, we must make sure we are currently in the right directory. This command can only run in the same directory as "perfStaes.log" and "hive-core.log" files. When running the tool, system will need these two .log file. If they are not in currently directory, the system will try to copy them and get a "permission denied". Both files are under "/usr/share/tomcat6/ROOT/WEB-INF/", one level higher than configuration files. Thus, we need "cd" to this directory and then run the "AdminVocabularies" command. The command lined and result will be:

```
bash-4.1$ cd /usr/share/tomcat6/ROOT/WEB-INF/
bash-4.1$ ls
classes                  hive-core.log.2013-05-22  lib
conf                     hive-core.log.2014-02-10
perfStats.log
deploy                   hive-core.log.2014-02-11
perfStats.log.2013-05-21
hive-core.log            hive-core.log.2014-02-17  tmp
hive-core.log.2013-05-21  hive-core.log.2014-02-19
web.xml
bash-4.1$ java -
Djava.ext.dirs=/usr/share/tomcat6/webapps/ROOT/WEB-
INF/lib edu.unc.ils.mrc.hive.admin.AdminVocabularies -c
/usr/share/tomcat6/webapps/ROOT/WEB-I
NF/conf -v stw -x -l -s -d
```

```
3/29|00:24:38 INFO
[edu.unc.ils.mrc.hive.admin.AdminVocabularies] - Starting
import of vocabulary stw
3/29|00:24:38 INFO
[edu.unc.ils.mrc.hive.api.impl.elmo.SKOSSchemeImpl] -
Loading vocabulary configuration from
/usr/share/tomcat6/webapps/ROOT/WEB-
INF/conf/stw.properties
3/29|00:24:38 WARN
[edu.unc.ils.mrc.hive.api.impl.elmo.SKOSSchemeImpl] -
Missing or invalid creationDate
3/29|00:24:38 WARN
[edu.unc.ils.mrc.hive.api.impl.elmo.SKOSSchemeImpl] -
atomFeedURL property is empty
Using kea stemmer l
Using maui stemmer maui.stemmers.PorterStemmer
3/29|00:24:40 INFO
[edu.unc.ils.mrc.hive2.api.impl.HiveVocabularyImpl] -
importConcepts /opt/hiveData/stw/stw.rdf
3/29|00:24:40 INFO
[edu.unc.ils.mrc.hive2.api.impl.HiveVocabularyImpl] -
Importing /opt/hiveData/stw/stw.rdf to Sesame store
3/29|00:24:47 INFO
[edu.unc.ils.mrc.hive2.api.impl.HiveVocabularyImpl] -
Import to Sesame store complete
3/29|00:24:47 INFO
[edu.unc.ils.mrc.hive2.api.impl.HiveVocabularyImpl] -
Initializing Lucene index
3/29|00:24:47 INFO
[edu.unc.ils.mrc.hive2.api.impl.HiveVocabularyImpl] -
Initializing H2 index
Mar 29, 2014 12:24:47 AM
org.openrdf.elmo.rolemapper.SimpleRoleMapper unregistered
WARNING: Unregistered type http://zbw.eu/namespaces/zbw-
extensions/Thsys
Mar 29, 2014 12:24:48 AM
org.openrdf.elmo.rolemapper.SimpleRoleMapper unregistered
WARNING: Unregistered type
3/29|00:24:40 INFO
[edu.unc.ils.mrc.hive2.api.impl.HiveVocabularyImpl] -
importConcepts /opt/hiveData/stw/stw.rdf
```

```
3/29|00:24:40 INFO
[edu.unc.ils.mrc.hive2.api.impl.HiveVocabularyImpl] -
Importing /opt/hiveData/stw/stw.rdf to Sesame store
3/29|00:24:47 INFO
[edu.unc.ils.mrc.hive2.api.impl.HiveVocabularyImpl] -
Import to Sesame store complete
3/29|00:24:47 INFO
[edu.unc.ils.mrc.hive2.api.impl.HiveVocabularyImpl] -
Initializing Lucene index
3/29|00:24:47 INFO
[edu.unc.ils.mrc.hive2.api.impl.HiveVocabularyImpl] -
Initializing H2 index
Mar 29, 2014 12:24:47 AM
org.openrdf.elmo.rolemapper.SimpleRoleMapper unregistered
WARNING: Unregistered type http://zbw.eu/namespaces/zbw-
extensions/Thsys
Mar 29, 2014 12:24:48 AM
org.openrdf.elmo.rolemapper.SimpleRoleMapper unregistered
WARNING: Unregistered type http://zbw.eu/namespaces/zbw-
extensions/Descriptor
3/29|00:24:56 INFO
[edu.unc.ils.mrc.hive2.api.impl.HiveVocabularyImpl] -
Initializing autocomplete index
3/29|00:24:58 INFO
[edu.unc.ils.mrc.hive2.api.impl.HiveVocabularyImpl] -
Autocomplete index initialization complete
3/29|00:24:58 INFO
[edu.unc.ils.mrc.hive.admin.AdminVocabularies] - Skipping
KEA H2 initialization
3/29|00:24:58 INFO
[edu.unc.ils.mrc.hive.admin.AdminVocabularies] - Skipping
KEA training
3/29|00:24:58 INFO
[edu.unc.ils.mrc.hive.admin.AdminVocabularies] - Skipping
Maui training
bash-4.1$
```

As we can see from the result, since we do not have KEA and MAUI flaggers in the command, the tool skipped KEA and MAUI training. The time HIVE spends to run this “AdminVocabularies” tool mostly based on the size of the RDF/SKOS file. “stw.rdf” is about 15MB, and it took around 20 seconds to run. Some large vocabulary such as

Library of Congress Subject Headings (LCSH) is about 300MB, which will take more than 5 minutes to finish the process. So, be patient if it looks like the system is not responding. Once we saw "`bash-4.1$`", that means HIVE has finished running "AdminVocabularies" tool.

If we "cd" to "/opt/hiveData/stw" we will see the directories and files the tool just created.

```
bash-4.1$ cd /opt/hiveData/
bash-4.1$ ls
agrovoc                          hive-core.log  mesh
stw
hive-agrovoc-sample-2.1.tar.gz  lcsh
perfStats.log  uat
bash-4.1$ cd stw/
bash-4.1$ ls
autocomplete  stwH2  stwIndex  stw.rdf  stwStore
bash-4.1$ cd stwH2/
bash-4.1$ ls
stw.h2.db  stw.lock.db
bash-4.1$ cd ..
bash-4.1$ cd stwIndex/
bash-4.1$ ls
_0.cfs  segments_2  segments.gen  write.lock
```

Now the new vocabulary has been added into the HIVE.RENCI demo, in order to make it available to the public we need to start Tomcat service.

```
bash-4.1$ exit
exit
[username@hive ~]$ whoami
username
[username@hive ~]$ sudo service tomcat6 start
 [sudo] password for username:
Starting tomcat6:
[  OK  ]
```

RESULT OF THE WORK

After that, we can refresh the tab of the demo in the web browser to see the result. It will take several seconds for the server to response after loading a new vocabulary.

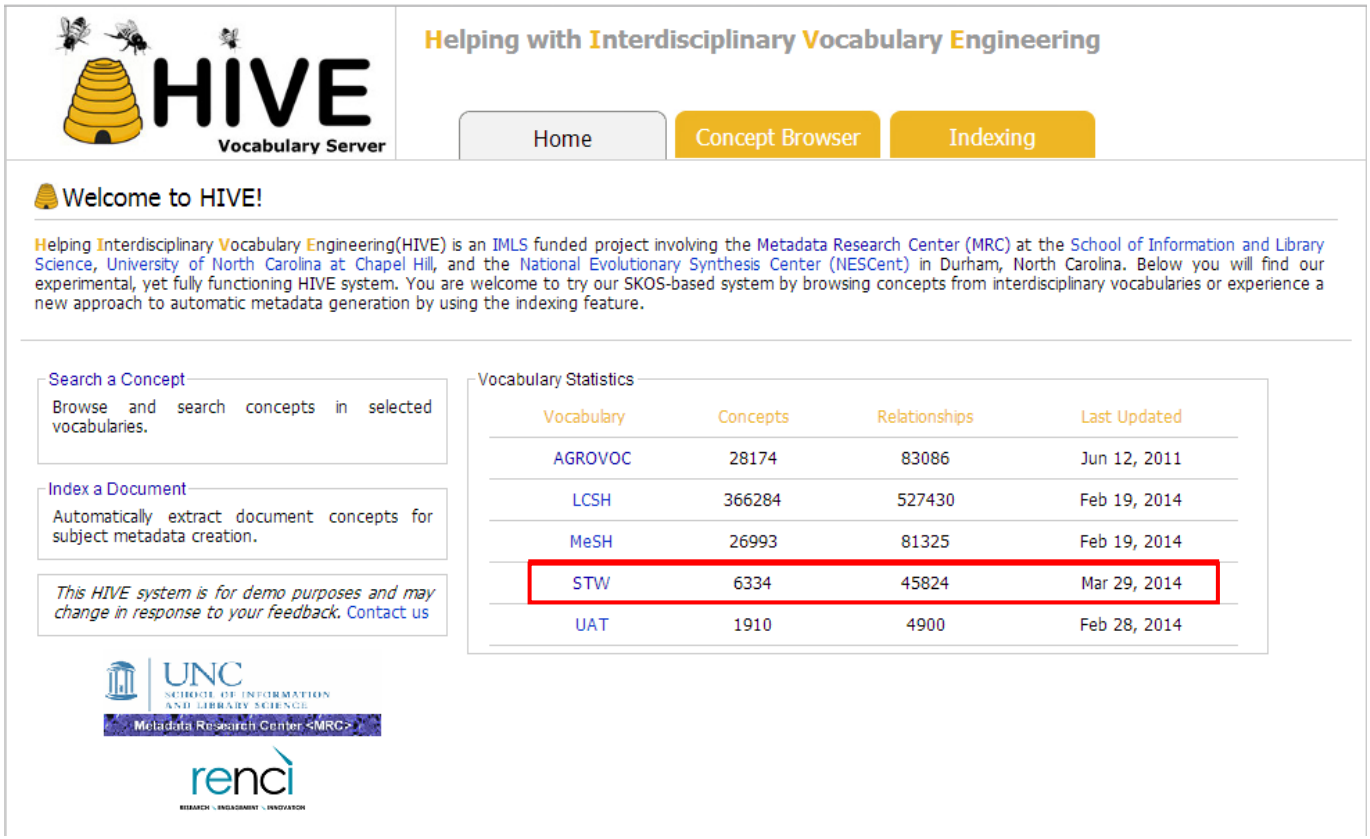

Figure 10 *HIVE.RENCI demo with newly added STW vocabulary*

We can see “STW” vocabulary is in the demo with 6,334 Concepts and 45,824 Relationships.

FUTURE STEPS OF THE STUDY

As mentioned earlier, HIVE is transferring its keywords extraction algorithm from KEA to MAUI that is why we did not include KEA or MAUI flaggers when running the “AdminVocabularies” tool. So the first big step the project should take is to finish the transferring process. After it is finished and training and testing dataset are

ready for each vocabulary, we re-run the “AdminVobucabularies” command to make indexing work with MAUI

Another step is try to develop the function to automatically update existing vocabularies to the latest version. For now, every time a new version of any existing vocabularies is published, we have to use “**wget**” command to manually copy the new RDF/SKOS file into HIVE, and also manually re-run the “AdminVocabularies” tool. So, in the future, the function should be developed to let HIVE automatically detect vocabulary update, copy it into server and run the “AdminVocabularies” tool.

REFERENCES

Greenberg, J. “Metadata and the World Wide Web”, The Encyclopedia of Library and Information Science, Vol.72, 224-261, Marcel Dekker, New York, 2003.

Greenberg, J. (2009). Theoretical considerations of lifecycle modeling: An analysis of the Dryad repository demonstrating automatic metadata propagation, inheritance, and value system adoption. *Cataloging and Classification Quarterly*, *47*(3–4), 380–402.

Greenberg, J., Losee, R., Pérez Agüera, J., Scherle, R., White, H., & Willis, C. (2011). Hive: Helping interdisciplinary vocabulary engineering. *Bulletin of the American Society for Information Science and Technology*, *37*(4).

Medelyan, O., & H.Witten, I. (2008). Domain-independent automatic keyphrase indexing with small training sets. *Journal of the American Society for Information Science and Technology*, *59*(7), 1026-1040. doi: 10.1002/asi.20790

H. Witten, I., W.Paynter, G., Frank, E., Gutwin, C., & G.Nevill-Manning, C. (1999). Kea: Practical automatic keyphrase extraction. *Proceedings of the fourth ACM conference on Digital libraries*, 254-255.

Miles, A & Pérez-Agüera, J (2007) SKOS: Simple Knowledge Organisation for the Web, Cataloging & Classification Quarterly, 43:3-4, 69-83, DOI: 10.1300/J104v43n03_04

Semantic web-w3c. (n.d.). Retrieved March 23, 2014 from http://www.w3.org/standards/semanticweb/

Currier Sarah, Lorna M. Campbell, Helen Beetham (2005). Pedagogical Vocabularies Review, JISC Pedagogical Vocabularies Project, Final Draft, 23rd December 2005, From http://www.jisc.ac.uk/whatwedo/programmes/elearningpedagogy/vocabularies.aspx

Harpring, P. (2010). *Introduction to controlled vocabularies: Terminology for art, architecture, and other cultural works*. (1st ed.). Los Angeles: the Getty Research Institute. Retrieved from http://www.getty.edu/research/publications/electronic_publications/intro_controlled_vocab/

RDF Working Group. (2014, 02 25). *Resource description framework (rdf)*. Retrieved March 24, 2014 from http://www.w3.org/RDF/

Miles, A., & Bechhofer, S. (2009, August 18). *Skos simple knowledge organization system reference*. Retrieved from http://www.w3.org/TR/skos-reference/

SKOS: Simple Knowledge organization for the web. (2012/01/01.). Retrieved March 24, 2014, from website: http://www.w3.org/2004/02/skos/intro

KEA: Keyphrase Extraction Algorithm. (n.d.).Retrieved March 24, 2014, from website: http://www.nzdl.org/Kea/description.html

Details about topic indexing with Maui. (2009/07/27).Retrieved March 25, 2014 from website: https://code.google.com/p/maui-indexer/wiki/InsideMaui