Development of a Classification Server for organizing data in a long term Preservation System

Dr. Sándor Kopácsi, Rastislav Hudak, José Luis Preza

Computer Center, University of Vienna

Universitätsstraße 7, A-1010 Vienna, Austria

{sandor.kopacsi, rastislav.hudak}@univie.ac.at, jl@preza.org

Abstract

In this paper we will describe the implementation of a Classification Server which helps organizing the data in a long term preservation system of digital objects. After a short introduction to classifications and knowledge organization we will set up the requirements of the system to be implemented. We will describe some appropriate SKOS (Simple Knowledge Organisation System) management tools that we have examined, including Skosmos, the solution we have selected for implementing the Classification Server. Skosmos is an open source, web-based SKOS browser based on Jena Fuseki SPARQL server. We will also discuss some crucial steps taken during the installation of the applied tools. Finally we will show some problems with the classifications to be used, and some possible solutions.

Keywords

long term preservation, metadata, classification, SKOS, Skosmos, Jena Fuseki, Skosshuttle

Introduction

Long term preservation of digital objects is a key issue today for libraries and research institutes, where we can ensure that digital content of books, documents, pictures, research data, etc. remains accessible and usable within a required period of time [1]. Digital preservation includes planning, resource allocation, and application of preservation methods and technologies [2].
When we store digital objects in an archiving system, it is very important to assign well-defined metadata, with which the objects can be easily found. Metadata can be considered as information about information, which can provide the title, the authors and keywords, and other information about a document. Metadata also stores the technical details on the format and structure, the ownership and access rights information, as well as the history of the preservation activities.

When the data provider of the digital object has to add standardized values as metadata, it is sometimes difficult to find the appropriate keywords, and sometimes requires guessing. If we want to avoid ambiguities, misspellings, etc. it is better to select the terms from pre-defined controlled vocabularies.
Controlled vocabularies, or rather classifications in many topics are available in several data sources, among which we can select. If we want to provide all relevant classifications to our archiving system, from where the user can select terms for adding metadata information during upload or search for data, a Classification Server, that handles our relevant vocabularies and classifications seems to be the ideal solution.
In a Classification Server the information should be stored according to classification- or knowledge organisation schemas, usually in the structure of Resource Description Framework (RDF) or as Simple Knowledge Organisation System (SKOS), and should be organized as Linked Data.
The University of Vienna has developed its own solution, called Phaidra[1], for archiving digital objects, that is also in use at several other institutions, and universities. To make the services of Phaidra more comfortable and more reliable, we are developing a Classification Server from which the user can select terms by accessing controlled vocabularies and classifications.

Classification

Classification is a form of categorization, that groups objects or items according to their subjects usually arranged in a hierarchical tree structure. This knowledge organization technique can take many forms that will be discussed below.
Controlled vocabulary is a closed list of words or terms that have been enumerated explicitly, which can be used for classification. It is controlled because only terms from the list may be used, and because there is control over who, when and how adds terms to the list.
Taxonomy is a collection of controlled vocabulary terms organized into a hierarchical structure by applying parent-child (broader/narrower) relationship. Each term in a taxonomy is in one or more relationships (e.g. whole-part, type-instance) to other terms in the taxonomy.
Thesaurus is more structured, much richer taxonomy, that uses associative relationships (like "related term") in addition to parent-child relationships.
Ontology is a more complex type of thesaurus expressed in an ontology representation language that consists of a set of types, properties and relationship types. In an ontology instead of having simply "related term" relationship, there are various customized relationship pairs that contain specific meaning, such as "owns" and its reciprocal "is owned by".

Knowledge Organisation Systems and Linked Data

Classifications can be considered as a collection of organised knowledge, therefore the technical background of classification is based on knowledge organisation systems. In

[1] https://phaidra.univie.ac.at/

knowledge organisation systems we usually store the knowledge in form of triplets, as object-predicate-subject, or object-attribute-value threesomes.
Classifications can be represented in Simple Knowledge Organisation Systems (SKOS) [3] as a Resource Description Framework (RDF) vocabulary. Simple Knowledge Organization System (SKOS) is a W3C recommendation designed for representation of thesauri, classification schemes, taxonomies, subject-heading systems, or any other type of structured controlled vocabulary.
Using RDF allows knowledge organization systems to be used in distributed, decentralised metadata applications. Decentralised metadata is becoming a typical scenario, where service providers want to add value to metadata harvested from multiple sources. [4]
Each SKOS concept is defined as an RDF resource, and each concept can have RDF properties attached, which include one or more preferred terms; alternative terms or synonyms; definitions and notes with specification of their language. Established semantic relationships are expressed in SKOS and intended to emphasize concepts rather than terms/labels. [5]
It is clear, that SKOS - as a modern well established standard - can (potentially) support formal alignments and hierarchical grouping of concepts using different SKOS relations (e.g. *skos:exactMatch, skos:closeMatch, skos:narrower, skos:broader, skos:related),* translation of concept labels, and URI-based mapping to similar concepts in other KOS.

Application of the Classification Server

The Classification Server will be a component of Phaidra[2], the Digital Asset Management System with long-term archiving functions developed by the University of Vienna. Phaidra is an acronym for Permanent Hosting, Archiving and Indexing of Digital Resources and Assets. It is implemented at several local Austrian institutions, and also internationally, including universities in Serbia, Montenegro and Italy. Phaidra gives educational, research- and management staff the possibility to archive objects and to permanently secure them, to create them systematically and to supplement them with metadata, as well as to archive objects for an unlimited period of time - and provides world-wide access to them.
We are going to apply the Classification Server when the user of Phaidra ingests new items to the archiving system, and wants to assign metadata to it from controlled vocabularies, or when the user searches for items supplied with terms from existing classifications. We also need it for resolving the terms saved in objects when displaying them.

Requirements of the Classification server

General requirements

Generally, we develope a Classification Server for Phaidra that supports classifications and controlled vocabularies. It should resolve the URIs of the terms, support multiple

[2] https://phaidra.univie.ac.at/

languages (the “official” languages in Phaidra are English, German, Italian and Serbian), support multiple versions of classifications, return the list of subterms (narrower concepts), and it should be Phaidra independent.
Phaidra should have no assumptions about the contents, which means that the set of classifications can differ on instances that are locally managed.
We were looking for solutions that do not require too much development efforts and have lower costs.

Technical requirements

Technically we expect the Classification Server to return the terms in multiple formats (such as XML, JSON, RDF, TTL). It should supports standard import formats for vocabularies (e.g. SKOS/RDF, TTL, N-TRIPLES), and should support Linked Data (in SKOS/RDF/XML formats). The Classification Server should provide SPARQL endpoint, and support search, that is needed for Phaidra. It should also support classifications/vocabularies that do not yet support linked data (do not have URIs).
It would be nice if it would be possible to use external terminology services, e.g. dewey.info, so that we do not necessarily have to import it locally.

Testing some available tools for classification

In this section we introduce some relevant tools that we have evaluated for implementing the Classification Server. We have also collected information about other tools (like HIVE, iQvoc, CATCH), but they did not appeared as candidates for our development, therefore we do not describe them in this paper.

ThManager

ThManager[3] is an open source tool for creating and visualizing SKOS RDF vocabularies. It was developed by the Advanced Information Systems Laboratory of the University of Zaragoza. It was implemented in Java using Apache Jena, and facilitates the management of thesauri and other types of controlled vocabularies, such as taxonomies or classification schemes. ThManager allows for selecting and filtering of the thesauri stored in the local repository. Description of thesauri by means of metadata is in compliance with a Dublin Core based application profile for thesaurus.
ThManager runs on Windows and Unix with the minimum requirement of the installation of the Java virtual machine. The application is multilingual, currently Spanish and English versions are available, but with little effort, other languages can be implemented. The main features include the visualization of thesaurus concepts (alphabetically, in hierarchical structure, properties of selected concepts); search for concepts ("equals", "starts with" and "contains"); edition of thesaurus content (creation of concepts, deletion of concepts, and update of concept properties); export of thesauri (including thesaurus metadata) in SKOS format.

[3] http://thmanager.sourceforge.net/

Available vocabularies in ThManager provide among others the vocabulary of AGROVOC, DCType, GEMET, ISO639, UNESCO.
Unfortunately the latest version of ThManager was launched in 2006, and we cannot expect any updates. Another drawback of ThManager is that it does not provide SPARQL endpoint for accessing the managed vocabularies.

TemaTres

TemaTres[4] is an open source vocabulary server that has been developed in Argentina. It includes a web application to manage and exploit vocabularies, thesauri, taxonomies and formal representations of knowledge, that are stored in a MySQL database, and provides the created thesauri in SKOS format. It requires PHP, MySql and a HTTP Web server. TemaTres provides SPARQL endpoint. Exporting and publishing controlled vocabularies are possible in many metadata schemas (SKOS-Core, Dublin Core, MADS, JSON, etc.). It can import data in SKOS-Core format and has a utility to import thesauri from tabulated text files.
It has an advanced search feature with search terms suggestions, and a systematic or alphabetic navigation. TemaTres has a special vocabulary harmonization feature where it can find equivalent, no equivalent and partial terms in other vocabularies.
It supports multilingual thesaurus, multilingual terminology mapping, as well as multilingual interface. It exposes vocabularies with powerful web services. TemaTres displays terms in multiple deep levels in the same screen. It also provides quality assurance functions (duplicates and free terms, illegal relations). The main problem with TemaTres is that its documentation is just partly available in English.

SKOS Shuttle

SKOS Shuttle[5] is an online Thesaurus Service, developed by Semweb LCC (Switzerland). It supports building, maintaining and operating SKOS thesauri. SKOS Shuttle allows operating on internal RDF repositories, and on SESAME compliant external RDF repositories. It also allows direct editing of RDF statements (triples).
It integrates a full REST API to create, manage and navigate thesauri that provides a full range of selections and commands that can be embedded into any application using different output formats (JSON, XML and YAML). It accesses securely all information through SSL transported authentication. It provides industrial security (Rights, Groups, User and Project Management) and a smart Orphan Concept Analysis together with an assistant for direct “deorphanization” without using SPARQL code. RDF Import/Export is possible in different formats (N3, N-Triples, TRIG, Turtle, NQuads, RDF/XML). The API access requires the same secured authentication as the application to provide online services.
SKOS Shuttle seems to be a very promising tool, but it is under development at the moment. It is available as a service and it is already used by several universities. SKOS

[4] http://www.vocabularyserver.com/
[5] https://ch.semweb.ch/leistungen/thesaurus-services/en-thesauri/?ucl=en

Shuttle is not an open source product, and the pricing is not yet known, but it will be provided as a free service for universities (up to a larger extent).

Poolparty

PoolParty[6] is a commercial semantic technology suite, developed by Semantic Web Company that offers solutions to knowledge organization and content business problems. The PoolParty Taxonomy & Thesaurus Manager is a powerful tool to build and maintain your information architecture. The PoolParty thesaurus manager enables practitioners to start their work with limited training. Subject matter experts can model their fields of expertise without IT support.
PoolParty taxonomy management software applies SKOS knowledge graphs. With PoolParty, you can import existing taxonomies and thesauri (e.g. from Excel) and export them in different standard formats. In addition to basic SKOS querying, the API also supports the import of RDF data, SPARQL update and a service to push candidate terms into a thesaurus.

Protégé

Protégé[7] is a free, open-source ontology editor and framework for building intelligent systems. It was developed by the Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine. Protégé is supported by a strong community of academic, government, and corporate users, who use Protégé to build knowledge-based solutions in areas as diverse as biomedicine, e-commerce, and organizational modelling.
With the web-based ontology development environment of Protégé, called WebProtégé, it is easy to create, upload, modify, and share ontologies for collaborative viewing and editing. The highly configurable user interface provides suitable environment for beginners and experts. Collaboration features abound, including sharing and permissions, threaded notes and discussions, watches and e-mail notifications. RDF/XML, Turtle, OWL/XML, OBO, and other formats are available for ontology upload and download. Although it is an efficient tool, it is too complex for editing and visualizing such a simple model as SKOS, and provides too many options not specifically adapted to the type of relationships used in SKOS.

Skosmos with Jena Fuseki

Skosmos[8], developed by the National Library of Finland, is an open source web application for browsing controlled vocabularies. Skosmos was built on the basis of prior works (ONKI, ONKI Light) on developing vocabulary publishing tools in the FinnONTO (2003–2012) research initiative at the Semantic Computing Research Group.

---

[6] https://www.poolparty.biz/
[7] http://protege.stanford.edu/
[8] http://skosmos.org/

Skosmos is a web-based tool providing services for accessing controlled vocabularies, which are used by indexers describing documents and by users searching for suitable keywords. Vocabularies are accessed via SPARQL endpoints containing SKOS vocabularies.
Skosmos provides a multilingual user interface for browsing vocabularies. Currently supported user interface languages are English, Finnish, German, Norwegian, and Swedish. However, vocabularies in any language can be searched, browsed and visualized as long as proper language tags for labels and documentation properties have been provided in the data.
Skosmos provides an easy to use REST API for read only access to the vocabulary data. The return format is mostly JSON-LD, but some methods return RDF/XML, Turtle, RDF/JSON with the appropriate MIME type. These methods can be used to publish the vocabulary data as Linked Data. The API can also be used to integrate vocabularies into third party software. For example, the `search` method can be used to provide autocomplete support and the `lookup` method can be used to convert term references to concept URIs. [6]
The developers of Skosmos recommend using the Jena Fuseki[9] triple store with the jena text index for large vocabularies. In addition to using a text index, caching of requests to the SPARQL endpoint with a standard HTTP proxy cache such as Varnish can be used to achieve better performance for repeated queries, such as those used to generate index view.

Overall evaluation and tool selection

All of the tested tools have advantages and disadvantages, but the most important selection criteria for us were to find a tool which is open source and which is based on the stable and widespread Jena technology that can also provide a SPARQL Endpoint and access via REST API.
For this selection criteria Skosmos with Jena Fuseki seemed to be the best solution, therefore we have selected them for implementing our Classification Server.

Implementation of the Classification Server

The classification server will be implemented using Skosmos as a frontend for handling SKOS vocabularies, and Jena Fuseki as a SPARQL Endpoint storing the SKOS vocabulary data (see Fig.1.).
Alternatively, we could use any other SPARQL 1.1 compliant RDF store, but the performance will likely not be satisfying with large vocabularies, since there is no text index support in generic SPARQL 1.1.

[9] https://jena.apache.org/documentation/serving_data/

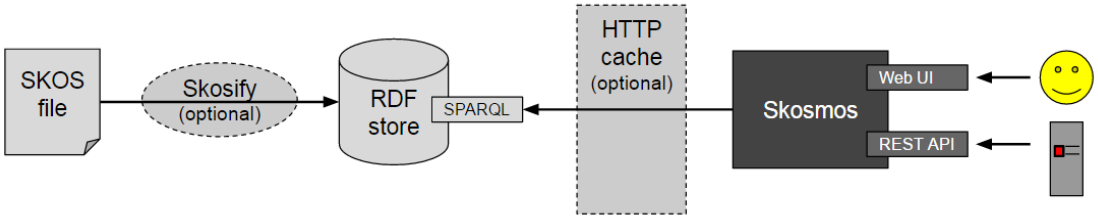

Fig. 1. System architecture (Source: [7])

Installation of Skosmos and Jena Fuseki

Skosmos and Fuseki require Apache and PHP running on the server. We have installed them on a Windows 7 environment (Professional 64 bit, Service Pack 1, Intel Core i7-56000 CPU, 2.6 GHz, 16 GB RAM) using Java 1.8 (jre1.8.0_40), with installed XAMPP (xampp-win32-1-8-3-4-VC11) , as well as on a CENTOS 6.5 virtual machine (Intel Xeon CPU E5-2670 0 @ 2.60GHz), but we are going to implement the final version on Ubuntu 16.04.

A detailed installation guide can be found on Github[10] for the Linux version, but there are some deviations on the Windows version, as well as there are some important issues that are worth highlighting.

Before installing Skosmos, it is advisable to make sure that Apache works and PHP is enabled. After installing PHP, you may need to restart Apache. If you are using Windows you need to install git for cloning, pulling as well as for the setting the dependencies for PHP.

You can either clone the code of Skosmos from Github[11] or download the zipped version. It is worth using the current stable version (maintenance branch), and better to clone from Github, because you can then easily upgrade to newer versions using `git pull`.

Skosmos requires several PHP libraries which will be installed using a dependency manager, called Composer[12]. For this, first you have to download the Composer (on Windows you have to set it up, too), and then install the dependencies with `install --no-dev` option. After downloading a new version of Skosmos you may need to update the dependencies with the `update --no-dev` option.

In Linux it may be required to set up Apache to access Skosmos under http://localhost/skomos by adding a symbolic link to the skosmos folder (e.g. opt/skosmos) into the DocumentRoot (e.g. /var/www/) and/or also you will need to give Apache permissions to perform network connections so that Skosmos can access SPARQL endpoints.

The default PHP configuration is probably appropriate for Skosmos, but you may check php.ini anyway. Make sure that the `date.timezone` setting is configured correctly, otherwise Skosmos pages displaying date values may not work at all. If you use

[10] https://github.com/NatLibFi/Skosmos/wiki/Installation
[11] https://github.com/NatLibFi/Skosmos
[12] https://getcomposer.org/

vocabularies with a potentially large number of triples, you may need to adjust the `memory_limit` setting. The default is usually 128M but the recommended setting is 256M.
JenaFuseki is a SPARQL server and triple store which is the recommended backend for Skosmos. The jena text extension can be used for faster text search. Simply download the latest Fuseki distribution and unpack the downloaded file to the intended folder of Fuseki.

Configuration of Skosmos and Fuseki

Configuration of Skosmos

Skosmos can be configured basically in two files, config.inc for setting some general parameters, and vocabularies.ttl is used to configure the vocabularies shown in Skosmos. In config.inc you can set the name of the vocabularies file, change the timeout settings, set interface languages, the default SPARQL endpoint, and set the SPARQL dialect if you want to use Jena text index.
Vocabularies are managed in the RDF store accessed by Skosmos via SPARQL. The available vocabularies are configured in the vocabularies.ttl file, that is an RDF file in Turtle syntax.
Each vocabulary is expressed as a `skosmos:Vocabulary` instance (subclass of `void:Dataset`). The local name of the instance determines the vocabulary identifier used within Skosmos (e.g. as part of URLs). The vocabulary instance may have the following properties: title of vocabulary (in different languages), the URI namespace for vocabulary objects, language(s) and the default language that the vocabulary supports, URI of the SPARQL endpoint containing the vocabulary, and the name of the graph within the SPARQL endpoint containing the data of the individual vocabulary.
In addition to vocabularies, the vocabularies.ttl file also contains a classification for the vocabularies expressed as SKOS. The categorization is used to group the vocabularies shown in the front page of Skosmos. You can also set the content of the About page in about.inc, and add additional boxes to the left or to the right of the front page in left.inc or right.inc.

Configuration of Fuseki.

Fuseki stores data in files. It is also possible to configure Fuseki for in-memory use only, but with a large dataset, this will require a lot of memory. The in-memory use of Fuseki is usually faster.
The jena text enabled configuration file specifies the directories where Fuseki stores its data. The default location will be /tmp/tdb and /tmp/lucene. To flush the data from Fuseki simply clear/remove these directories.
The jena text extension can be used for faster text search, and Skosmos simply needs to have a text index to work with vocabularies of medium to large size. The limit is a few thousand concepts, depending on the performance of the endpoint / triple store and how much delay is acceptable to the users.
If you start Fuseki in the TDB with `./fuseki-server --config config.ttl` then it will run using text index. For that you have to configure the TDB location and for jena text index the lucene text directory in config.ttl. If you start fuseki in the memory with `./fuseki-server --update --mem /ds`, then there is no text indexing by default.

It is also possible to use in-memory TDB and text index, but you need a Fuseki configuration file (config.ttl) with special "file names" that are actually in-memory (for TDB: tdb:location "--mem--"; and for jena text: text:directory "mem";)

Timeout settings

If there is more data than Skosmos is able to handle, some queries can take very long time. The slow queries are probably the statistical queries (number of concepts per type, number of labels per language) as well as the alphabetical index.
Short execution timeout for PHP scripts can trigger Runtime IO Exception. To change the timeout values it is suggested to check PHP and Apache's time out settings (e.g. in php.ini the max_execution_time). It is highly recommended to find this setting and change it to a higher value (say to 5 or 10 minutes)..
Skosmos also has a HTTP_TIMEOUT setting in config.inc, that should only be used for external URI requests, not for regular SPARQL queries, but there may be unknown side-effects. The EasyRdf HTTP client has a default timeout of 10 seconds. It is also recommended to change this value.
It is suggested to change the time out value, from the browsers where you are planning to access Skosmos. In Internet Explorer you can do it with regedit to change the Internet settings. You have to add a new DWORD value that is the KeepAliveTimeout and set the appropriate time-out value in milliseconds.
In Firefox there are two ways to change the timeout. You can extend it, or you can totally disable timeout. Depending on the way you use Firefox, either option may be helpful. Type `about:config` in your search bar on the top, and in the list of preferences you find Timeout, where you can either set the "enabletimeout" value to false, or you can enter a new value of "CountTimeout", which sets the timeout.
It is not allowed to change the timeout setting in Google Chrome.

Getting and setting vocabularies

The basic usage of our Classification Server is to store the classifications locally (if its access time is acceptable), and we also provide the links to the remote SPARQL endpoints of the classifications, if they are available.
If you want to use certain vocabularies locally, you have to get it in the right format, and upload it to the local SPARQL server, that is to Jean Fuseki.

Downloading and converting vocabularies.

Vocabularies can be downloaded from the original dataset provider (e.g. from Getty, COAR, Statistics Austria, etc.), or in case of a small dataset, it can be created manually. The vocabulary needs to be expressed using SKOS Core representation in order to publish it via Skosmos, but SKOS-XL representation or even files in Excel can be also easily converted to SKOS Core. For the SKOS-XL to SKOS Core conversion you can use for example the owlart converter[13]. You can also convert SKOS-XL labels to SKOS Core labels by executing SPARQL Update queries. If you have your classification in Excel you can write VBA macros to convert it to SKOS Core structures.

[13] https://bitbucket.org/art-uniroma2/owlart/downloads

The format of the file that is accepted by Fuseki can be rdf/xml (.rdf or .xml), turtle (.ttl) or N-Triples (.nt).
If either the SKOS file was downloaded from external resources or it has been converted from other formats, it is recommended that you pre-process your vocabularies using SKOS proofing tool, like Skosify[14]. This will ensure, e.g., that the broader/narrower relations work in both directions and related relationships are symmetric. Skosify will report and try to correct lot of potential problems in SKOS vocabularies. It can also be used to convert non-SKOS RDF data into SKOS. An online version of the Skosify tool is available, where after selecting the vocabulary to be checked, you can simply use the default options.

Uploading files to Fuseki.

If you want to use Skosmos for accessing classifications in your local SPARQL triple store you have to upload the datasets to Fuseki. First of all you have to consider if Fuseki will run either in the memory or in a predefined folder, usually called TDB. If you run Fuseki in the memory, then all uploads and updates (if you allow that) will be temporal. If you run Fuseki in the TDB, then the uploads and updates will remain there even if we exit from Fuseki and restart it.
You also have to consider that in a SPARQL triple store there is always a default (unnamed) graph, and there can also be multiple named graphs. In other words, there is only one default graph (with no name), but there can be any number of named graphs on a SPARQL endpoint/dataset. The URI namespaces can be used as graph names. E.g. http://vocab.getty.edu/tgn/ would store Getty's TGN data.
You can basically upload datasets to Fuseki online, when Fuseki is running, or offline, when Fuseki is not running. To upload online you can use the control panel of the web interface of Fuseki or you can use command line instructions. Offline you can upload datasets directly to the TDB.
When you upload datasets online to Fuseki through its control panel you can either set the Graph to “default” or you can provide a graph name. If you use a certain graph name when uploading a dataset to Fuseki, you have to take care of giving the same graph name to this dataset in `skosmos:sparqlGraph` (e.g. http://vocab.getty.edu/tgn/) by setting Skomos vocabularies in vocabularies.ttl.
If you want to upload datasets online to Fuseki from the command line, you have two options: the s-put and s-post utilities, which are part of Fuseki. You can use s-put, if you want to add a single data file. Note that s-put clears the dataset first, that is why it may happen, that you overwrite the previous data when loading a new file. If you want to add new data files to existing data without clearing it, you should use the s-post utility. In both cases (s-put and s-put) we have to provide the name of the SPARQL-server, the file name to be uploaded, as well as the name of the graph, where the dataset will be available (according to the `skosmos:sparqlGraph` name settings in vocabularies.ttl).
The Fuseki file upload handling is not very good at processing large files. It will load them first into memory, only then to the on-disk TDB database (and also the Lucene/jena text index). It can to run out of memory on the first step ("OutOfMemoryError: java heap space" is a typical error message when this happens). If you give several GB memory to

[14] https://code.google.com/p/skosify/

Fuseki (for example Setting JVM heap to 8 GB: `export JVM_ARGS=-Xmx8000M`) it should be able to upload large (several hundreds of MB) files, though it might take a while and you may want to restart Fuseki afterwards to free some memory.
If you have several and large files to upload to Fuseki it is worth using the offline data upload utilities. The dataset and the index can be built using command line tools in two steps: first load the RDF data, second create an index from the existing RDF dataset. You can load data with tdbloader which is part of the Jena distribution, but you may need to download it separately. Using the tbdloader you have to shut down Fuseki (since only one process can use the TDB at the same time). For tdbloader you have to provide a configuration file, the graph name with which you want access the vocabulary, and the file name you want to upload. You can use tdbloader for several files after each other, then you can build the text index as a separate step with the jena text indexer tool.
If you allow updates to the dataset through Fuseki, the configured index will be automatically updated on every modification. This means that you do not have to run the above mentioned jena.textindexer after updates, only when you want to rebuild the index from scratch.

Setting vocabularies.

To set the vocabularies you are going to use in Skosmos, you have to edit the vocabularies.ttl file which is an RDF file in Turtle syntax. First of all you have to create a vocabularies.ttl file in the Skosmos directory.
In this file after defining the prefixes (like rdf, skos, dc) you have to create new sessions for each vocabulary starting with the line `:id a skosmos:Vocabulary, void:Dataset;` where `id` is just an identifier, that will be used in the URL after /skosmos/.

In vocabularies.ttl you have to set the following required parameters:

1. the title of the vocabulary in different languages: `dc:title "title_of_the_vocabulary"@ language;` where language can be: en, de, it, etc.
2. the category of the vocabulary: `dc:subject : category;` where category can be: `cat_general,` or other defined categories in vocabularies.ttl, expressed as SKOS. The categorization is used to group the vocabularies shown in the front page.
3. the URI namespace for vocabulary objects (these may not overlap): `void:uriSpace "URI_namespace";` When you represent your data as RDF, the best practice is to coin a new URI namespace for your data set. Then use that as the value of the `uriSpace` setting.
4. the language(s) that the vocabulary supports: `skosmos:language " language ", " language ", ... ;` where language can be: : en, de, it, etc.
5. the URI of the SPARQL endpoint containing this vocabulary `void:sparqlEndpoint <URI_SPARQL_endpoint> ;`URI_SPARQL_endpoint can be http://localhost:3030/ if you want to use the vocabulary locally, or the URL of the SPARQL endpoint of the remote vocabulary

It is recommended to set the following optional parameters:

1. the default language of the vocabulary, if the vocabulary supports multiple languages: `skosmos:defaultLanguage " language ";` where language can be: : en, de, it, etc.
2. setting `skosmos:showTopConcepts true` should display the top level hierarchy - assuming that the dataset contains the `skos:hasTopConcept` and/or `skos:topConceptOf` relationships that are necessary for this to work. If you want to enable the *Hierarchy tab* showing top-level concepts on the vocabulary home page: `skosmos:showTopConcepts` should be set to `"true";`
3. *Group index* is meant for thematic groupsClass of resources to display as concept groups, or as arrays (subdivisions) of sibling concepts (typical values are `skos:Collection` or `isothes:ConceptGroup): skosmos:groupClass isothes:ConceptGroup;` If you do not need this tab, simply remove the `skosmos:groupClass` setting.
4. if you do not want Skosmos to query the mapping concept URIs for labels if they haven't been found at the configured SPARQL endpoint: `skosmos:loadExternalResources "false";`
5. URI of the main `skos:ConceptScheme` (instance of the current vocabulary) should be specified if the vocabulary contains multiple `skos:ConceptScheme` instances `skosmos:mainConceptScheme <main_Concept_Scheme_URI>.`
6. if the vocabulary is relatively small (e.g. 100 concepts) you can show the alphabetical index with all the concepts instead of showing only the concepts of one letter at a time: `skosmos:fullAlphabeticalIndex "true";.` It is not recommended to use `fullAlphabeticalIndex` for large vocabularies

Some examples and problems of adding individual vocabularies

In this chapter we are going to describe some examples for individual vocabularies that we are using in our Classification Server, that show typical problems and solutions.

Getty vocabularies

Getty vocabularies[15] contain structured terminology for art and other cultural, archival and bibliographic materials. They provide authoritative information for cataloguers and researchers, and can be used to enhance access to databases and web sites.
Getty has its own SPARQL endpoint, but it is not responding in the right way. There seems to be some incompatibility between Skosmos (in practice, the EasyRdf library which is used to perform SPARQL queries) and the Getty SPARQL endpoint.
Even if we could access the Getty SPARQL endpoint, it would most likely be extremely slow to use it with Skosmos, since it doesn't have a text index that Skosmos could use. The lack of a text index would most likely prevent any actual use of Skosmos with the Getty endpoint.

[15] http://vocab.getty.edu/

Therefore we have tried to upload Getty vocabularies to our own local Fuseki SPARQL endpoint, with the jena text index. But unfortunately Getty vocabularies do not work well in Skosmos due to their very large size.
There are two sets of each Getty vocabulary, the "explicit" set and the "full" set (Total Exports). With the "explicit" set, which is smaller, we had to configure Fuseki to use inference so that the data store can infer the missing triples. With the full set this is not needed, but the data set is much larger so we had difficulties loading it. We could finally upload the full set of Getty's vocabularies using the tdbloader utility.
The downloaded export file of the full set includes all statements (explicit and inferred) of all independent entities. It is a concatenation of the Per-Entity Exports in N-Triples format. Because it includes all required Inference, it can be loaded to any repository (even one without RDFS reasoning).
We had to download the External Ontologies (**SKOS**, SKOS-XL, ISO 25964), from http://vocab.getty.edu/doc/#External_Ontologies to get descriptions of properties, associative relations, etc. We have downloaded the GVP Ontology from http://vocab.getty.edu/ontology.rdf .
And finally we have loaded the full.zip export files (aat, tgn and ulan) form http://vocab.getty.edu/dataset/.
In this way we have made Getty vocabularies available in our Classification Server, but due to their huge size, they are extremely slow.

COAR Resource Type Vocabulary

COAR Resource Type Vocabulary[16] defines concepts to identify the genre of a resource. Such resources, like publications, research data, audio and video objects, are typically deposited in institutional and thematic repositories or published in journals.
The main problem with COAR is that it only represents labels using SKOS XL properties. Skosmos doesn't support SKOS XL currently. Unfortunately the remote endpoint of COAR[17] cannot be used, either, because the COAR endpoint data currently is not SKOS Core, but SKOS-XL. Since we wanted to use COAR data in our Classification Server, we have converted to SKOS Core labels using owlart (see Downloading and converting vocabularies).

ÖFOS

The ÖFOS[18] is the Austrian version of the Field of Science and Technology Classification (FOS 2007), maintained by Statistics Austria. The Austrian classification scheme for branches of science (1-character and 2-character) is a further development modified for Austrian data.

---

16 https://www.coar-repositories.org/activities/repository-interoperability/ig-controlled-vocabularies-for-repository-assets/deliverables/
17 http://vocabularies.coar-repositories.org/sparql/repositories/coar
18 http://www.statistik.at/

ÖFOS can be downloaded in PDF and CSV format, but neither in SKOS structure (in xml/rdf, turtle or N-Triples) format, nor Linked Open Data through a SPARQL Endpoint is available.
Since we have received it directly from Statistics Austria in Excel format, the simplest way of converting it to SKOS was using VBA macros. These macros simply reads the content of the Excel file, extend them with the appropriate RDF and SKOS labels, and writes the to the desired xml/rdf or ttl format.

Available services and usage of the Classification Server

Starting Fuseki and Skosmos

Before starting Fuseki it is worth extending the available JVM heap at least up to 8 GB by typing `$export JVM_ARGS=-Xmx8000M`. If you want to use the local vocabularies of the internal triplestore server, you have to start Fuseki first by typing the command line from the home folder of Fuseki "`java -jar fuseki-server.jar --config jena-text-config.ttl`", where jena-text-config.ttl is the configuration file of Fuseki. You can start Fuseki in the memory for making experiments with temporarily uploaded datasets, or you can use the TDB version where the triplestore will be stored permanently. (For details see the chapter Configuration of Fuseki). If you want to access external triplestores only, you do not have to start Fuseki at all.
You can start Skosmos from your web browser by typing in the address bar http://localhost/skomos or the domain name that you have assigned to Skosmos.

Available classifications and usage of the Classification Server

Currently we can access four general on-line classifications from external triplestores (AGROVOC, Eurovoc, STW, UNESCO), some other general local classifications (e.g. Getty, GND, ÖFOS, COAR Resource Type Vocabulary, etc.) and two local, Phaidra specific classifications. The local classifications have to be uploaded to our local triplestore before we want to access it from Skosmos.
The usage of the Classification Server is quite simple: from the opening page of Skosmos we simply have to click on one of the classifications, and the classification will be opened. We can see its vocabulary information, and we can select between alphabetical and hierarchical view. Depending on the configuration we can see the change history of the vocabulary or the group of concepts. We can also search specific contents directly in our entire triplestore server, or simply in the selected classification.

Connecting Phaidra to the Classification Server

Phaidra will require the Classification Server when the user ingests new items, and wants to add metadata to this from a controlled vocabulary, as well as when the user searches for some documents classified with some metadata from a controlled vocabulary and wants to display or resolve them.
The connection between Phaidra and the Classification Server will be realised using the REST API of Skosmos and/or Fuseki. Skosmos provides a REST-style API and Linked

Data access to the underlying vocabulary data. The REST API is a read-only interface to the data stored in the Classification Server.
REST URLs must begin with /rest/v1. Most of the methods return the data as UTF-8 encoded JSON-LD, served using the application/json MIME type. The data consists of a single JSON object which includes JSON-LD context information (in the @context field) and one or more fields which contain the actual data.

References

1. Digital Preservation Coalition (2008). “Introduction: Definitions and Concepts”. Digital Preservation Handbook. York, UK. Retrieved 24 February 2012.

2. Day, Michael. “The long-term preservation of Web content”. Web archiving (Berlin: Springer, 2006), pp. 177-199.

3. W3C Working Group: SKOS Simple Knowledge Organization System Primer, Note 18 August 2009 https://www.w3.org/TR/skos-primer/

4. W3C Semantic Web: Introduction to SKOS, https://www.w3.org/2004/02/skos/intro

5. Zeng, M. L., & Chan, L.M. (2015) Semantic Interoperability. In Encyclopedia of Library and Information Sciences 4th ed. p. 8.

6. Suominen, O., Ylikotila, H., Pessala, S., Lappalainen, M., Frosterus, M., Tuominen, J., Baker, T., Caracciolo, C., Retterath, A. (2015). Publishing SKOS vocabularies with Skosmos. Manuscript submitted for review, June 2015.

7. Osma Suominen. Publishing SKOS concept schemes with Skosmos. AIMS Webinar 6th April 2016, Slide 25.

Acknowledgements

We would like to express our special thanks to Osma Suominen, the main developer of Skosmos at the National Library of Finland, who was very helpful by answering any Skosmos or Jena Fuseki related question.