Joonas Laitio, Jussi Kurki

SAHA Metadata Management System Technical Report

Semantic Computing Research Group

Aalto University
School of Science
and Technology

AALTO UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY

Author: Joonas Laitio, Jussi Kurki

Title: SAHA Metadata Management System
Technical Report

Date: 21.3.2011 Language: English Number of pages:3+23

Semantic Computing Research Group

Department of Media Technology

Professorship: Media Technology Code: T-75

SAHA is a browser-based metadata editor and annotation tool for creating data in RDF format for the semantic web. Its main function is to provide a distributed environment where multiple distributed people can simultaneously annotate and edit RDF data through a web interface. The system is built paying special heed to scalability and concurrency. For example, the persistence is divided into two pieces to avoid the slowness of certain types of search through the whole RDF graph. The system also provides various ways to remotely query the data store, such as a SPARQL endpoint,. The aim of this document is to provide enough background information and instructions for adopting use and administration of SAHA in their own server environment.

Keywords: semantic web, metadata, annotation, indexing

Contents

1 General Information

The purpose of this document is to provide basic information and user instructions for the SAHA metadata management system. With this document it should be possible to adopt SAHA to one's own server environment and use all aspects of it successfully. This section describes SAHA's general information and development history.

1.1 Features

SAHA is a www-based tool for creating instance data for the semantic web in RDF (triple-based) form [2]. The base features include:

- Creating, deleting and editing of resources denoted by URIs in a robust distributed web environment
- Searching the model either by ontological class or with a global auto-completion search
- Lookup of ontology concepts for defined fields from external ONKI ontologies [8]
- Creating and editing point/area/route location data with a map interface
- Accessing the data from a SPARQL endpoint[6] or various other APIs
- Exporting the data as RDF/XML, Turtle (Notation 3)[1] or N-Triples
- Looking up data with a single object property based facet-search interface[5]

1.2 History

Browser-based annotation tools for the semantic web are not an entirely novel idea.[3][4] However, as the technology is still relatively new, implementing them can be difficult. SAHA has gone through multiple iterations throughout its history; these are listed below.

1.2.1 SAHA1

The first SAHA was created mainly by Onni Valkeapää in 2006 in the Semantic Computing Research Group[1]. The architecture was based on Apache Cocoon[2], and featured most of the very basic functionality still present. The backend was using JenaDB as the triple store, which is basically a Jena model stored in a relational database. [7]

1.2.2 SAHA2

There were some maintenance and scalability issues with the backend of SAHA1, which eventually led to changing the architecture from Apache Cocoon to Spring MVC and Freemarker templates in 2008. This was a direct file-by-file port of SAHA1, and it yielded some performance boosts all around the system. Still, functionality like text searching was very slow since the whole RDF model had to be iterated through.

1.2.3 SAHA3

SAHA3 is a complete rewrite of the metadata editor, implemented in 2009 mainly by Jussi Kurki. The basic architecture is quite similar to SAHA2, with the critical addition of a full-text index in the form of Apache Lucene[3] certain key functions like text searching and result sorting. The main triple store was also changed from JenaDB to TDB, which is a lot faster to setup and query for typical transactions in an editing environment. This is the current iteration and the subject matter in this report.

[1] http://www.seco.tkk.fi/
[2] http://cocoon.apache.org/
[3] http://lucene.apache.org/

2 System Overview

This section describes the technical functions of SAHA in more detail as the previous chapter. This includes the system architecture of SAHA and the various APIs it uses and offers.

2.1 Architecture

The general system architecture is shown in Fig. 1. The central component in the architecture is the MVC framework. It receives page load requests, queries the persistence store, acquires the correct template and fills it with the right data. In addition to normal requests, the MVC system handles asynchronous requests used throughout the system in the form of DWR calls. As the MVC system, the Spring Framework[4] is used, and Freemarker[5] is the template engine. Both these and the business layer code are written in Java.

Data access from the website is twofold. All the data is stored as triples in a triple store, Jena TDB RDF Database[6], but operations that require extensive searching and sorting, such as the autocompleting text search, use a separate full text index that is implemented with Apache Lucene[7]. The full text index is created from the data upon project initialization and updated as the data in the triple store changes.

An alternate way to access the data is through a SPARQL endpoint, described in more detail in section 2.3.1. The Joseki SPARQL server[8] is used here and it queries the triple store directly.

2.2 Authentication and User Management

SAHA does not natively support authentication or user roles. The URL paths used in the web interface are constructed in such a way that it is rather easily possible to use external authentication methods (such as those provided by the web server) to have certain URL patterns require authentication.

[4] http://www.springsource.org/
[5] http://freemarker.sourceforge.net/
[6] http://openjena.org/TDB/
[7] http://lucene.apache.org/
[8] http://www.joseki.org/

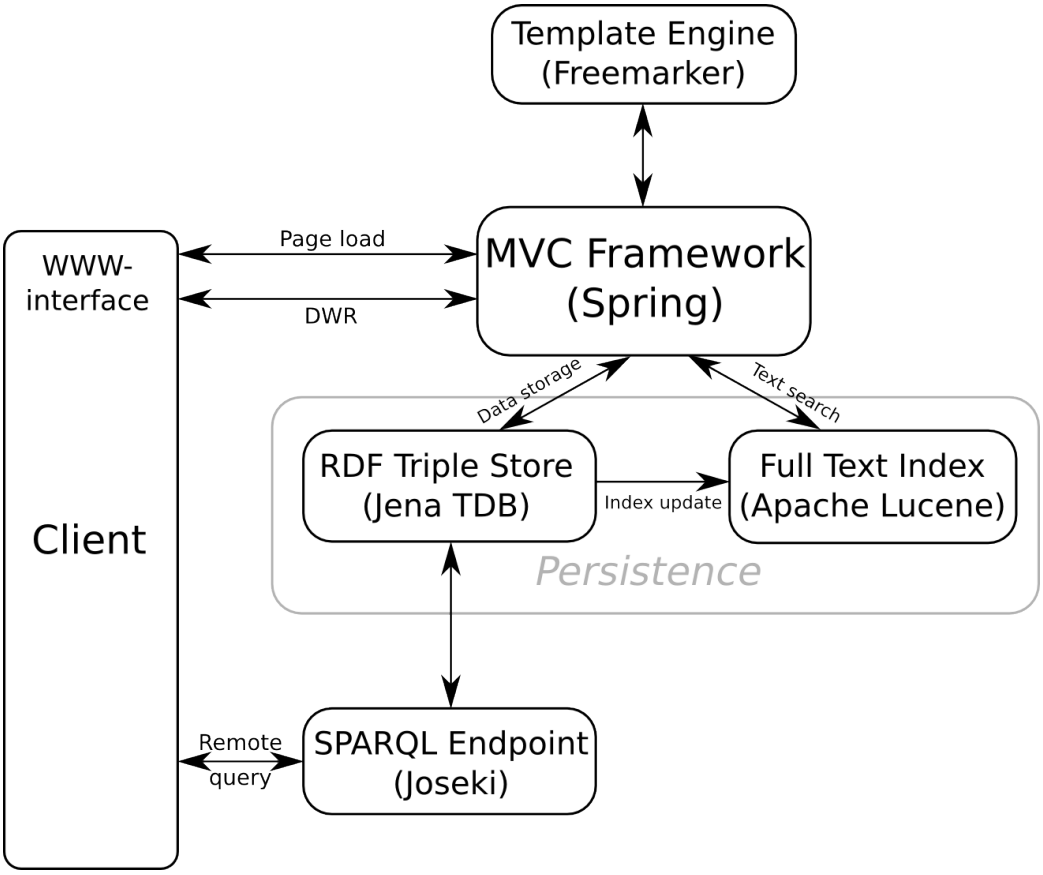

Figure 1: SAHA system architecture

2.3 SAHA API

2.3.1 SPARQL Endpoint

The main way to remotely query SAHA's RDF-based data storage system is through its SPARQL endpoint. SPARQL (SPARQL Protocol and RDF Query Language[9]) is a semantic query language designed for querying triple stores, roughly analogous to how SQL is used in relational databases. For example, to query all the resource types from a certain SPARQL endpoint, the following query could be used:

[9]http://www.w3.org/TR/rdf-sparql-query/

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?type
WHERE { ?resource rdf:type ?type . }
```

The actual URLs for the project-specific SPARQL endpoints are as follows:

```
http://www.example.org/saha/service/data/{project}/
sparql?query={query}
```

where {*project*} is the name of the SAHA project and {*query*} (both without the curly brackets) the SPARQL query. Note that because SPARQL queries use many reserved characters for URLs, the query needs to be URL-escaped. The result is given as SPARQL Query Results XML[10], for which the MIME type is `application/sparql-results+xml`. Here's an example of what a typical result for the above query could look like in a dataset only containing a schema but no data:

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="type"/>
  </head>
  <results>
    <result>
      <binding name="type">
        <uri>http://www.w3.org/2002/07/owl#DatatypeProperty</uri>
      </binding>
    </result>
    <result>
      <binding name="type">
        <uri>http://www.w3.org/1999/02/22-rdf-syntax-ns#Property</uri>
      </binding>
    </result>
    <result>
      <binding name="type">
        <uri>http://www.w3.org/2002/07/owl#Class</uri>
      </binding>
```

[10] http://www.w3.org/TR/rdf-sparql-XMLres/

```
    </result>
  </results>
</sparql>
```

NOTE: Because the SPARQL endpoint is a functional system from the content management, only the projects present on server startup are loaded as SPARQL endpoints. So any newly created projects cannot be queried with SPARQL until a server restart.

2.3.2 HTTP REST API

For getting info about a specific resource, one can use a simple HTTP request to get all of the triples of that resource in RDF format. The request URL is formed as follows:

```
http://www.example.org/saha/{project}/export.shtml?uri={uri}
```

where {*project*} is the name of the SAHA project and {*uri*} is the URI of the target resource, URL-escaped. If the results are wanted in human-readable HTML form, the address is similarly:

```
http://www.example.org/saha/{project}/resource.shtml?uri={uri}
```

2.3.3 Text Search API

It is also possible to make remote text searches with the same API that the web interface itself uses for its autocompletion text search. The output of this API is designed to be used with Dojo's FilteringSelect widget[11] but can be applied to other uses as well. The request is formed as follows:

```
http://www.example.org/saha/service/instance_search/
?model={project}&name={query}
```

The results given by the API are in JSON format with the following structure:

[11] http://docs.dojocampus.org/dijit/form/FilteringSelect

```
{"items":
[{"name":"{display HTML}",
  "uri":"{uri}"},
 {"name":"{display HTML}",
  "uri":"{uri}"}],
 "identifier":"uri"}
```

where {*display HTML*} is HTML designed to appear in the widget's combo box (containing info about why the search matched the hit) and {*uri*} is the URI of the resource that the search hit.

3 End User Instructions

3.1 Main View and Search

The main view is the front page of a project and gives a general overview of it. On the left side all the classes (as denoted by owl:Class) are listed, along with the count of how many instances of that class exist in the project. New instances can also be created from this list as can be seen in Fig. 2. The instances for any class can be browsed and filtered to find the desired ones.

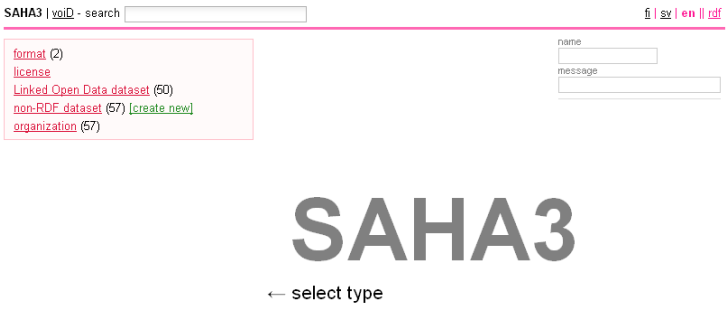

Figure 2: Resource view

The searchbar at the top of the screen is available through most views of SAHA, excluding HAKO. The search box there is a dynamic one that searches for the given string prefix in all the resource labels and any of their literal property values. By default the amount of search results shown is limited. There are two meta characters that can be used in the search box:

;
: The semicolon breaks the search so that the matches must be exact word matches and not prefixes.

+
: The plus sign removes the restriction on the result size - all hits are shown and not only the first few. This can be combined with the ; metacharacter above for a search of type `search;+`.

3.2 Resource View

The resource view, shown in Fig. 3, is a basic overview of a resource. There is a resource view for each resource in the model. All the property values of the resource are listed, except those that are configured to be hidden. The data cannot be edited here—to do that the `[edit]` button must be pressed, which takes the user to the Editor view, described in section 3.3.

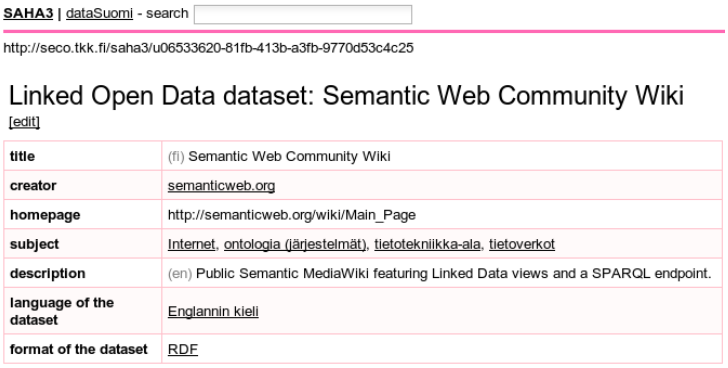

Figure 3: Resource view

3.3 Editor View

The editor view is where most of the actual model editing takes place. There is an editor view for each resource in the model. In the editor view of a specific resource, the properties of that resources can be added, modified and removed. As shown in Fig. 4 along with an overview of the editor view, there are two main types of properties—ones whose value is a literal, and ones whose value is another resource.

Editing literal properties (appointed by the higher red circle in Fig. 4) is straightforward. As the values are in most cases simple strings, they can easily be added or edited by modifying their respective text fields. The text field for editing existing properties is brought up by clicking the current value. Handling object properties—those that have a resource as their value—is a bit more complicated.

Object properties are added by writing parts of the desired object's label into the search field (appointed by the lower red circle in Fig. 4). This

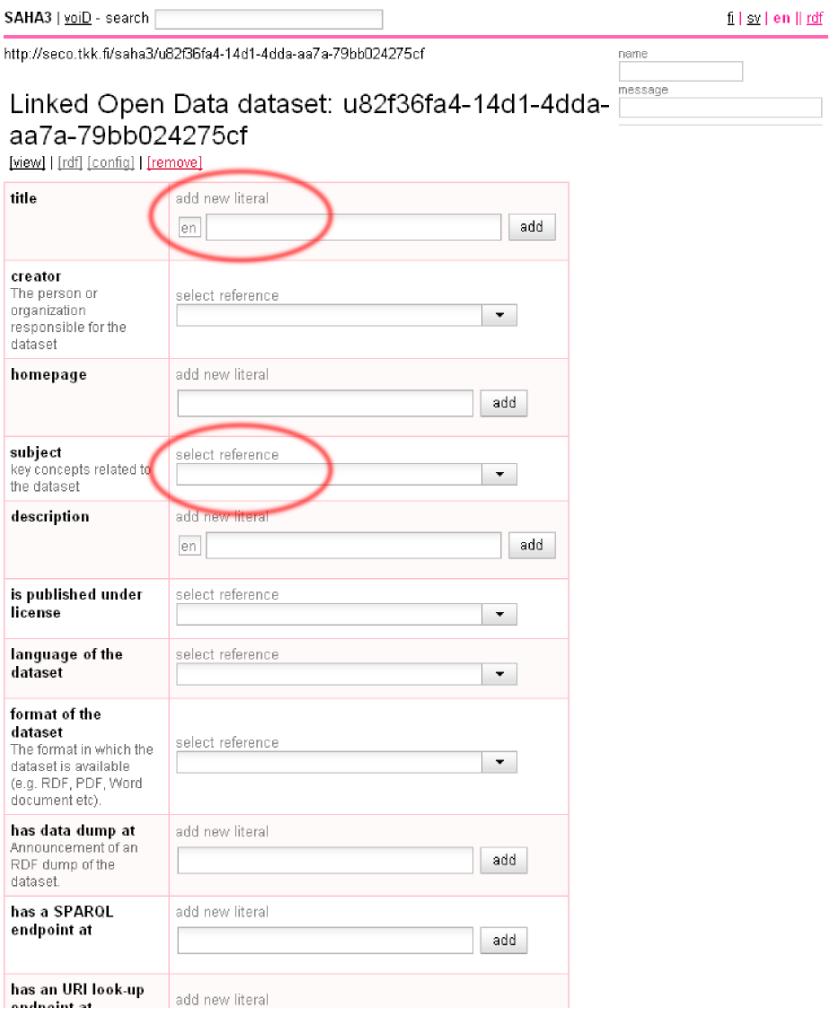

Figure 4: Editor view with literal and object properties

prompts an autocompletion search done (by default) both to the local model and any external sources configured for the property. The search functionality is very similar to the overall search, but typically the search set is only a subset of the whole model. The resources found are shown below the search field, as seen in Fig. 5. Choosing one of the results will put that resource as a value for the property.

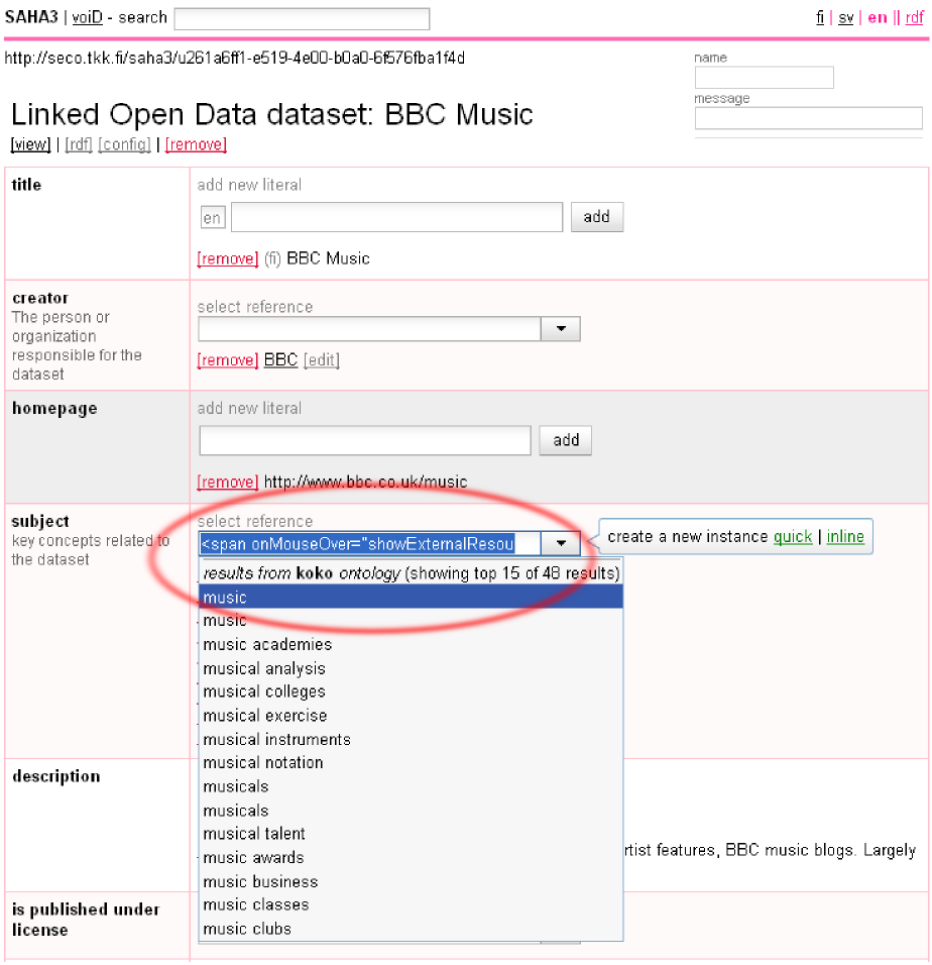

Figure 5: Search results for object properties

3.3.1 Inline Editor

The inline editor is a powerful tool available for adding and editing object properties. As shown in Fig. 6, the inline editor is effectively an editor within the editor. When adding or editing object properties, often there are some small edits that need to be done for the object resource as well, but leaving the current resource view and coming back would be cumbersome. Thus the resource that is the property's value can be edited in place, both when adding or editing object properties.

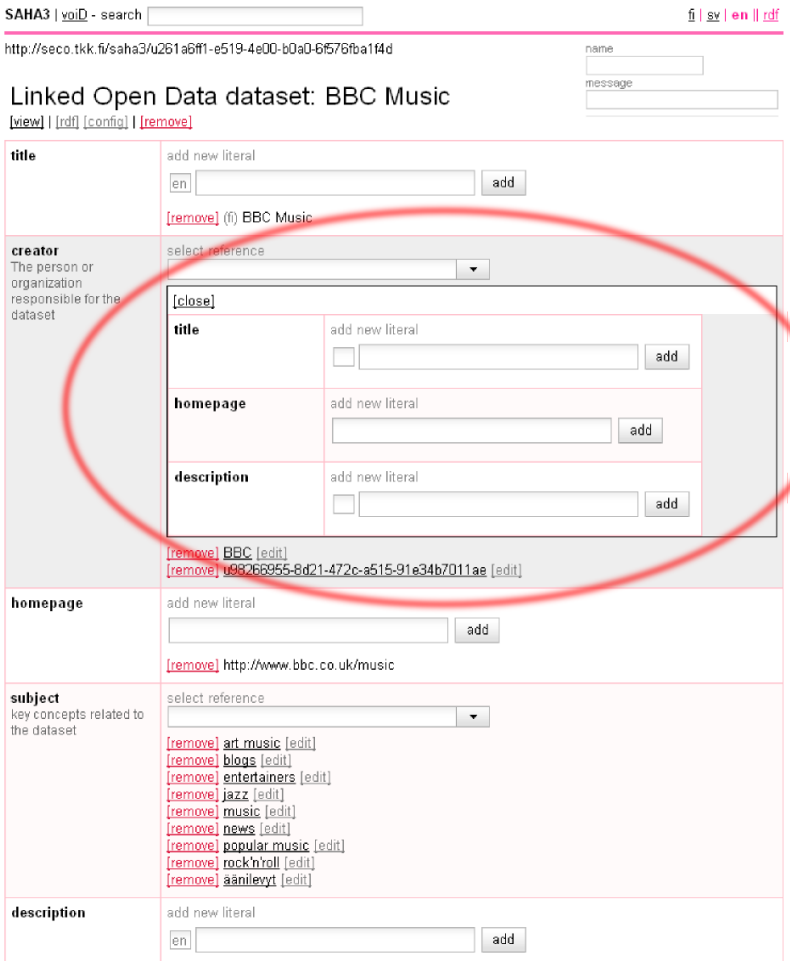

Figure 6: Inline resource editor

Inline editors can be nested; one can be opened within another inline editor to an arbitrary degree. They can also be closed to avoid view cluttering. Closing an inline editor also closes all inline editors nested within it.

3.4 Configuration View

The configuration view is for configuring how resources are edited. This mostly involves choosing and arranging the appropriate properties and their possible value sets. There is an editor view for each *class* in the model, denoted by being of type *owl:Class*[12] or a subclass of it. The changes affect all resources that have this class as type. The actual view, shown in Fig. 7, contains separate options for literal and object properties.

[12]URI: http://www.w3.org/2002/07/owl#Class

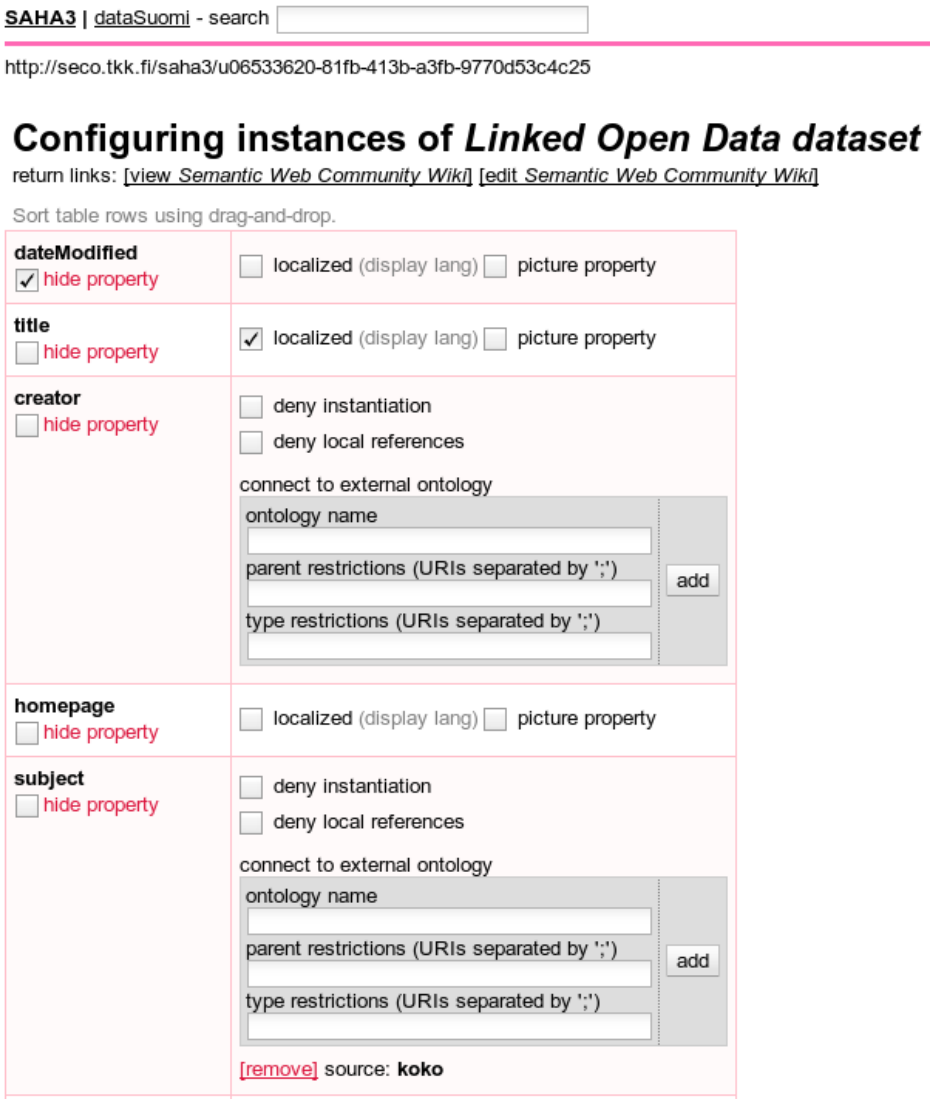

Figure 7: Configuration view

The functions of the different options are as follows:

Literal properties

localized
Controls whether or not to show the language definition for literals. If it is shown, the language definition can be modified and all new values for this property are set to the default language. If it is not set, new values will not have a language definition.

picture property
Properties with this option set are not treated like normal literal properties. Instead, there will be a file upload field in the editor view intended for uploading pictures. The picture will be saved into the data model folder of the project and a marker literal value will be saved to the property, which when viewed from the resource view is dereferenced to the actual image and it is shown. Fully supported image formats are JPG, GIF and PNG.

This setting can also be used to upload other file types than images such as audio data—in that case nothing is shown in the resource view except a link to the uploaded file.

Object Properties

deny instantiation
If this setting is checked, the user cannot create new resources when setting a value for this property. This essentially means that the user can only choose from a preset or separately defined group of values instead of being able to create new values on the fly.

deny local references
When searching for possible values for this property, the instances in the local model (that are of appropriate class as defined by the property's *range* definition[13]) are not included in the search. Only the external ontologies are searched.

adding external ontologies
The search set can be expanded beyond the local project data by configuring ONKI repositories. These repositories are searched in addition to the data that's locally present.

[13]URI: http://www.w3.org/2002/07/owl#range

ontology name
: Name of the ONKI ontology to be connected to. If the name set here is an URL, it is instead assumed that this is the URL to the WSDL definition of a web service that implements the ONKI web service interface[14].

parent restrictions
: URI or a list of URIs that specifies the ancestors for the search set. All search results have one of the defined URIs as a parent - the hierarchy is defined by the external ontology can be of any kind, such as one defined by *skos:broader*[15] or *dc:isPartOf*[16].

type restrictions
: URI or a list of URIs that specifies the root classes for the search set. More specifically, all search results are of a type that is one of the listed URIs or a subclass thereof.

3.5 HAKO

HAKO is a lightweight multifacet search system designed to function as a more powerful way to search and filter data in a SAHA project than the normal text search provides. It uses the same data stores and indices as SAHA and is essentially an integral part of the same system. HAKO can be accessed from a link in the upper right hand side of the SAHA main view.

3.5.1 Using HAKO

Using HAKO for searching instances is straightforward. At first, all possible results are shown. These are then filtered in two ways:

Text search
: The text search filters results by their label. The search is a prefix search, so it matches words starting with the search term(s).

Faceted search
: The main way to filter information is the facet search. Using preconfigured properties as facets, the search can be restricted to those resources

[14] http://www.yso.fi/onkiwebservice/
[15] URI: http://www.w3.org/2004/02/skos/core#broader
[16] URI: http://purl.org/dc/terms/isPartOf

that have certain values on certain properties. Fig. 8 shows a search with three facet restrictions (the format, subject and language of the items in the search set) active - only resources that meet all these three criteria are shown.

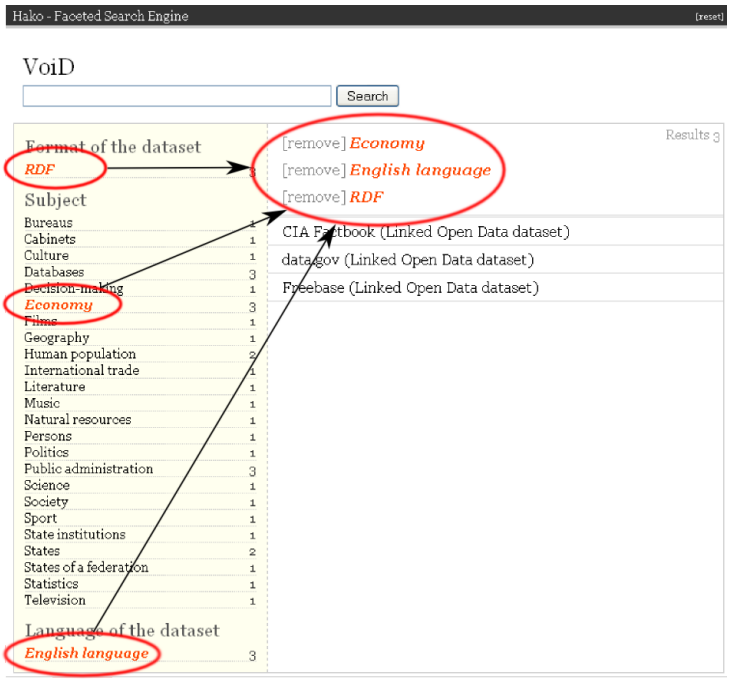

Figure 8: An example HAKO search

3.5.2 Configuring HAKO

Configuring HAKO is done by going to the HAKO view of a project through the HAKO link in SAHA with no HAKO configuration done. Instead of the regular HAKO view in a configured project, this brings up the configuration screen shown in Fig. 9. Once the configuration is done, the only way to bring up the configuration page again is to reset the configuration and set them up from scratch.

All that the configuration consists of is a list of the instance classes and facet properties.

Instance classes

Instance classes define the set of possible search results. All shown results, shown on the right side in the HAKO view in Fig. 8, must be of a type configured here.

Facet properties

Facet properties are the properties used for constraining the search. All shown search results must have the matching values for these properties. Fig. 8 shows three facet properties, circled in red on the left side.

Figure 9: HAKO configuration view

4 Administrational Instructions

This section describes tools and instructions for creating and administrating SAHA and SAHA projects. The section separately describes administrating existing projects and creating base models for new projects.

4.1 Environment setup

SAHA is implemented to be used as a web application on a Tomcat (5+) server. There are only a few environment-specific settings, and they are all located in the SAHA3 Spring bean definitions in (as of the current version) `/src/main/resources/saha3-beans.xml`. It defines the beans *SahaProjectRegistry* and *SahaChat*, both of which require the base directory for storing the projects as a parameter.

Further, there are two backup systems implemented. *SahaBackupManager* keeps three different backups for each model —one that is refreshed daily, one every week and one every month. The system keeps a manifest and timestamped backups in the specified `backupDirectory` folder. *SimpleBackupManager* is similar but only stores one backup per model, refreshed daily. It is less complex and thus more robust, but the backup could get corrupted as well as it is a day old at longest.

4.2 Administration Tools

There are two main ways to administrate SAHA in the web user interface: a general admin page and a project-specific management page. The general admin page is accessed from the URL:

`http://www.example.org/saha/saha3/admin.shtml`

By default the URL is not protected by authentication requirements and access should generally be controlled by e.g. a web server based authentication system. The admin view provides separate log info for the whole application and the backup system, along with general information about the application process such as uptime and memory usage. Specific projects can also be modified —they can be added/merged/rewritten with an RDF file or closed. Closing a project frees any memory used by the project and releases file channels so that the project info can be modified from the file system without shutting down the server.

The project-specific management page is accessed from the main view of a project (Sec. 3.1). It is protected by a password that can be changed from that management view - on project creation it's set as an installation-specific default. Features offered by the management page are project deletion, adding/merging/rewriting the project with a given rdf input file, and changing the management password.

4.3 Building a Basis Model for a Project

Because SAHA is designed to be an instance editor for generating metadata, it doesn't handle multiple meta-levels completely. Specifically, it is hard to alter the class hierarchy and certain class and property relations after the project is initialized. So the schema that is loaded in upon project creation should be finalized beforehand. This section describes which aspects of the schema control what behavior in the SAHA user interface: the ones directly related are listed below.

NOTE: This is separate from the SAHA-specific configuration done via the web UI as described in section 3.4. The web UI configuration is for settings that *only control the UI's and data's behavior in SAHA*, while the schema definitions below also control the data's behavior outside SAHA. As the configuration view settings (the ones listed at Section 3.4) are only interesting in the context of SAHA, they are stored in a separate XML configuration file instead of the data model.

rdfs:label, skos:prefLabel
: These are the basic label properties used: a label is shown instead of the resource's localname in every applicable spot in the user interface. For functions that don't call for a specific label property but instead an implicit ''label'' is created, by default `skos:prefLabel` is used.

owl:Class, rdfs:subClassOf
: The classes in the base model serve as the skeleton for the annotations. These are shown in the main view (Sec. 3.1) along with their respective instance counts. All the types of resources that are supposed to be creatable and editable in a SAHA model must have their class definition included in the base model as well. The class hierarchy used in the creation of the class tree in the main view is built from the `rdfs:subClassOf` triples in the base model.

owl:DatatypeProperty, owl:ObjectProperty, xsd:date
: All properties must be defined in the base model to be able to add new triples that have them as predicate. Object properties are defined with `owl:ObjectProperty` and literal properties with `owl:DatatypeProperty`. Datatypes for typed literals are ignored with the exception of `xsd:date`, which, when used as the range of a DatatypeProperty, prompts the use of a calendar widget instead of a regular text field.

rdfs:comment
: Any `rdfs:comment`s that a property has are displayed in the editor view (Sec. 3.3). This is intended to be a way to give instructions to users in how to use those properties when annotating.

rdfs:domain
: By default, only the properties that are in a class's domain (as denoted by `rdfs:domain`) are shown as options for new triples in the editor view (Sec. 3.3). Other properties must be explicitly added to the class's domain from the configuration view (Sec. 3.4).

rdfs:range
: When searching for values for object properties, all shown local hits are of a type that is in the property's range, as defined by `rdfs:range`. If the range is not set, all resources are searched. **NOTE**: this does not affect hits from an external ontology - those are controlled purely by their specific settings as set in the configuration view (Sec. 3.4).

wgs84:lat, wgs84:long
: The properties `wgs84:lat`[17] and `wgs84:long`[18] in a class's domain denote that it's intrinsitc for that type of resource to have a location in the form of latitude and longitude coordinates. A map interface is added to those resource in the editor to input and edit coordinate data.

sapo:hasPolygon, sapo:hasRoute
: Like the `wgs84` properties, `sapo:hasPolygon`[19] and `sapo:hasRoute`[20] in a class's domain denote that a resource can have an area or route, respectively. Appropriate map interfaces are added to those resources.

[17] http://www.w3.org/2003/01/geo/wgs84_pos#lat
[18] http://www.w3.org/2003/01/geo/wgs84_pos#long
[19] http://www.yso.fi/onto/sapo/hasPolygon
[20] http://www.yso.fi/onto/sapo/hasRoute

NOTE: A resource can only have one type of location info, and only one of that type at a time.

Most other schema definitions do not have an effect on SAHA functionality. Naturally, the triples themselves are preserved in the model and can be used in other applications should the model be exported from SAHA. They include, but are not limited to, the following:

owl:maxCardinality, owl:minCardinality, owl:cardinality
: SAHA does not keep track of or enforce cardinalities. This is problematic in the open world in the first place, so it's up to the user to ensure that there are no missing or extraneous resource values.

owl:allValuesFrom, owl:someValuesFrom, owl:oneOf
: SAHA does not support value restrictions. If you want to restrict a property to a certain set of values, this can be done by properly setting the range of that property to an appropriate class and disallowing instantiation.

5 Acknowledgements

The current implementation of SAHA is mainly developed by Jussi Kurki, with some work done by Joonas Laitio. Some of the design owns homage to the previous SAHA versions, implemented by Onni Valkeapää, Olli Alm, Joonas Laitio and Katariina Nyberg.

Developing SAHA has been a part of the National Semantic Web Ontology project in Finland[21] (FinnONTO, 2003–2012), funded mainly by the National Technology and Innovation Agency (Tekes) and a consortium of 38 organizations.

[21] http://www.seco.tkk.fi/projects/finnonto/

References

[1] T. Berners-Lee and D. Connolly. Notation 3 (N3) A readable RDF syntax. *W3C Submission, Jan*, 2008.

[2] D. Brickley, R.V. Guha, and B. McBride. RDF vocabulary description language 1.0: RDF schema. *W3C recommendation*, 10:27--08, 2004.

[3] S. Handschuh and S. Staab. Authoring and annotation of web pages in CREAM. In *Proceedings of the 11th international conference on World Wide Web*, pages 462--473. ACM, 2002.

[4] J. Kahan, M.R. Koivunen, E. Prud'Hommeaux, and R.R. Swick. Annotea: an open RDF infrastructure for shared Web annotations. *Computer Networks*, 39(5):589--608, 2002.

[5] J. Kurki and E. Hyvönen. Collaborative Metadata Editor Integrated with Ontology Services and Faceted Portals. 2010.

[6] E. Prud'Hommeaux, A. Seaborne, et al. SPARQL query language for RDF. *W3C working draft*, 4, 2006.

[7] O. Valkeapää, O. Alm, and E. Hyvönen. Efficient content creation on the semantic web using metadata schemas with domain ontology services (system description). *The Semantic Web: Research and Applications*, pages 819--828, 2007.

[8] Kim Viljanen, Jouni Tuominen, and Eero Hyvönen. Ontology libraries for production use: The finnish ontology library service onki. In *Proceedings of the 6th European Semantic Web Conference (ESWC 2009)*, May 31 - June 4 2009. Springer-Verlag.