Web Ontology Editor: architecture and applications

Dmitry Shachnev

Lomonosov Moscow State University, department of Mechanics and Mathematics
+7-916-7053644, mitya57@mitya57.me

Abstract. The paper presents the ontology editor which was built as part of ISTINA (Intellectual System for Thematic Analysis of Scientometrical Data), a system which is used in Lomonosov Moscow State University and a number of other institutions. The client side of the editor is based on web technologies (HTML 5, Twitter Bootstrap, JS), and allows one to enter concepts and relations between them in a web browser in an interactive mode. The server side of the editor is using the Django web framework like the rest of the ISTINA system, and implements a SPARQL endpoint, a version control system, users' personal ontology pages, collaborative editing and conflicts resolution mechanisms. The data is stored in a relational database, queries to the database are generated using the Django object-relational mapping system. This paper presents the editor architecture, database storage, details of the version control system and various applications, including automating the learning process.

Keywords: ontology, semantic web, ontology editor, scientometrics, version control, learning process

1 Ontologies in modern web

A popular definition of an ontology is "formal, explicit specification of a shared conceptualization" [1]. This definition specifies neither structure nor exact semantics of the ontology, so it is generally easier to use other models for the ontology. The *triplet model* says that ontology is a set of (subject, predicate, object) triplets, where the subject and the predicate are ontology concepts, and the object can be either an object or a literal. Any concept is determined by its universal r d identifier (URI), and sometimes the triplet itself can have a URI associated with it (thus forming a quadruplet). The *graph model* says that ontology is an oriented graph where all nodes and all edges are labeled with URIs. The *object-oriented model* says that ontology is a set of concepts, where every concept is an instance of a some class and has a set of properties according to its class. The properties can be either literal values or pointers to other concepts.

The World wide web consortium (W3C) has developed a set of standards which describe the key notions of ontologies and ways to represent them. Most importantly, the RDF and RDF Schema standards describe some common datatypes such as *rdf:Property*, *rdfs:Class* and *rdfs:Literal*, common property types such as *rdf:type* and

rdfs:subClassOf, and another standards in RDF family describe serialization formats such as XML syntax, N-triples, N-Quads, and the functional representation.

The RDF format only deals with representation of ontologies, it does not describe the semantics. To add semantics, the OWL family of standards has been developed. OWL (Web Ontology Language) extends RDF with logic, which allows one to write computer programs that can for example verify the consistency of the ontology or extract implicit knowledge ontology based on formulae which are part of it. For example, OWL supports: assertions about classes (e.g. whether some classes can intersect or should be disjoint), definitions of new classes using formulae, and kinds of properties: reflexive, symmetric, transitive, functional and others.

One more important standard of ontologies and the semantic web is the SPARQL query language. It is an equivalent of SQL language for relational databases and has the same set of common operations, but works in terms of triplets and makes it very easy to perform common operations on ontologies.

2 Introduction to ISTINA system

ISTINA (the Russian abbreviation for "Intellectual System for Thematic Analysis of Scientometrical Data") is the current research information system (CRIS) which is used in Lomonosov Moscow State University and in several institutes of the Russian Academy of Sciences. It stores various results of scientific and educational activity of employees, such as articles, conference talks, patents, research projects, lecture courses, students' guidance, etc. It is integrated with international systems such as Web of Science and Scopus. The system is described in detail in a book [4] published by the Lomonosov Moscow State University.

One of the key features of ISTINA is the ability to build a list of works for an employee using a certain formula, with every work having a weight according to that formula. A formula is a set of lines, where each line defines a set of works and a function to build the weight. For example, a line can say that articles in a journal having a Scopus impact-factor get a weight equal to $N \times$ *journal IF* $\div$ *number of coauthors*. It is possible to add multiple filters and restrictions, such as "select all oral talks on international conferences where the worker was the presenter". A work is included into the result if it matches at least one formula line; if it matches several lines the one which yields the maximum weight is used. Internally these formulae are stored as JSON, where each line is a tuple of (category of works, restrictions, parameter to use as initial weight, numeric multiplier, modifier functions to apply). A formula can be evaluated for a single employee, forming a set of ranked works, or for a department, which gives a ranked list of employees where for each employee a sum of their works' weights is given.

There are several ontologies that are used in ISTINA system. First, it is the internal ontology of the system which is used in the formulae evaluating system. A category in a formula defines a class of works, and a relation between a worker and a work, for example "course – author", "course – lecturer", or "project – responsible implementer". Each category has a set of properties, either numeric or boolean. For example, for

a course being read these are its duration in weeks, number of academic hours per week, and number of students. There is a JSON-like structure which maps the categories and the properties onto the underlying relational database structure. This mapping is used to generate the SQL queries for each line of a formula. The internal structure of the formulae and the algorithms used in the SQL generator are described in detail in [3].

Another type of ontology is the meta-ontology of scientific knowledge. It is closely related to the internal ontology and describes various concepts that are common to all branches of science, such as authors, works, events, facts and so on. Finally, there are subject area ontologies for every branch of science which are connected to the ontology of scientific knowledge and to each other. Each of these ontologies describes the corresponding branch of science, various facts and works in it. The key classes in every branch of science can be different: for example, one of the key class in biology would be 'species', it chemistry — 'substance' and in mathematics — 'theorem'.

In ISTINA, there are two ways of populating the ontology: semi-automatic population and manual population. The semi-automatic population happens when a user uploads some text to the system, for example a paper or a talk abstract. This text is then scanned using the Brainsterm and Sonmake algorithms [2], and based on the scan results suggestions for populating the ontology are presented to the user, which he/she can either accept or decline.

The manual population is done using the ontology editor which was developed specifically for the ISTINA system. The main goal when developing such an editor was to provide a means to edit the ontology for users who are not experts in information technologies or in computer science. Because of this goal, the most widely used solution, the Protégé editor, was not acceptable for us.

3 Server side of the editor

The ISTINA system is based on Django web framework (version 1.8 at the moment of writing this), and each subsystem is implemented as a Django application, one of which is the ontology editor. The editor is using the RDFLib library, which greatly simplifies work with RDF graphs and provides a SPARQL 1.1 endpoint with support for serializing the results as JSON, export and import to various formats (most importantly Turtle), and possibility to write a custom store backend. Some of the built-in stores in RDFLib include an in-memory store, a remote SPARQL store, a "Sleepycat" store using the Oracle Berkeley DB, and a relational database store which uses the SQL Alchemy framework.

The most common approaches for storing an ontology in a relational database are the following.

- Direct mapping, where a table in the database corresponds to an ontology class, and table fields correspond to properties of the class. This algorithm is described in detail in the "A Direct Mapping of Relational Data to RDF" standard of the W3C.
- Mapping using a mapping language. There is another W3C standard for that, "R2RML: RDB to RDF Mapping Language". It allows for indirect mapping,

where for example classes can be distributed across multiple tables or non-table constructs, such as SELECT queries.
- Storing everything in a limited set of tables. In the very simple case it can be one table (for triplets) or two tables (triplets and literals).

In ISTINA, a hybrid approach was taken. Because the ontology is large, and one of the goals is integrating it with the objects already existing in the relational database, we are storing the ontology itself in a limited set of tables, but provide a means to link with objects in the relational database. For the latter, we are using Django's "generic foreign keys" mechanism. A generic key is a (model id, primary key) pair and allows linking to any table that is registered as a Django model. These generic keys are integrated into Django object-relational model, which makes it very easy for us to use them in the RDFLib store implementation.
The main tables which we are using for ontology are listed below.

- CONCEPT table — maps URIs and internal numeric identifiers for concepts.
- CONCEPTPROPERTY table — stores literals: numeric values, dates and strings, optionally with language tags.
- TRIPLET table — stores a (subject, predicate, objects) triplets. *Subject* field can map to concepts and external objects, *predicate* field — only to concepts, *object* field — to concepts, external objects or literals.

4 Version control system

The version control system (VCS) for the ontology is similar to the non-distributed version control systems for files and directories, such as Subversion. So we will use the same terms for common operations as popular version control systems use (given below in italics).

The goals of a version control system are: grouping all changes to ontology into revisions (commits), checking all changes in a particular revision or between two revisions (*diff*), checking who and when created a given triplet (*blame*), checking all changes to a particular concept's relations (*log*), and viewing the state of ontology at a point in past (*checkout*).

As the version control is non-distributed and there is no need to have branches, we can use a linear history with revisions numbered using integers. There is a table named REVISION with three fields: identifier, Django user which created the revision, and date at which it was created. Then, every triplet has two fields: *fromRevision* and *toRevision*. The process for applying a change is as follows.

- First, a revision number is obtained. We use an Oracle DB sequence for that. A downside of this approach is that in case something goes wrong, the revision number will be lost. On the other hand, it is the only thing that will be lost, and we do not rely on continuity of the revisions numbers sequence anywhere. All the subsequent steps happen as part of a transaction in the database.

- A revision object is created. The date is assigned automatically, the user is set to the current user.
- For all concepts that are being created, *fromRevision* is set to the number of current revision, and *toRevision* is set to NULL. For all concepts that are being deleted, *toRevision* is set to the number of current revision.

This algorithm makes it very easy to do all of VCS operations. In particular, below are some rules in pseudo-SQL.

- Browse the current state of ontology: *toRevision* is NULL.
- Browse a state at revision N: $fromRevision \leq N$ and $\text{nvl}(toRevision, N) \geq N$.
- Check which concepts were created or deleted in revision N: $fromRevision = N$; $toRevision = N$.
- Compare revisions M and N (if $M < N$): $M \leq fromRevision \leq N$; $M \leq toRevision \leq N$.

5 Ontology pages

Because it is impossible to browse the whole ontology, every user is viewing or editing only a small piece of it at a time. These pieces are called “ontology pages” and are also stored in the relational database. Every page can be viewed by anyone, but the contents of it can be edited only by its creator. The pages are represented in the relational database as two tables: ONTOLOGYPAGE which stores the identifier, the name and the page owner, and ONTOLOGYPAGEMEMBER which links a page to the concepts it contains. On the client side, the relations which are drawn are the relations between the concepts included in the page, or between a concept and a literal.

6 Client side of the editor

The client side is using two big libraries: Twitter Bootstrap for widgets, styles and modal dialogs, and vis.js, in particular its Network component, for drawing the graph. The set of web pages available in the editor is the following.

- List of all revisions. This is a paginated list which shows who and when made which changes to the ontology.
- Viewing a particular revision. This page lists all added and all deleted triplets for a particular revision.
- Viewing a particular concept. This is what opens when one pastes a concept URI from ISTINA into a web browser.
- Viewing or editing a page. A page owner can edit it, all other users can only view it. When the page is editable, the vis.js data manipulation mode is enabled: buttons for adding a new concept, adding a new edge, and removing the selected concepts or edges are shown. When clicking on one of the two create buttons, a pop-up modal dialog is shown. For a new concept, one must select its class, enter a label

and optionally a URI (the prefix is fixed). When a URI is not entered, it is generated automatically. The client side uses the SPARQL JSON endpoint for performing various requests, such as retrieving the list of available classes. Making changes does not immediately save them to the server. Changes are stored locally on the client side and saved only when the user presses the "Save changes" button.

The changes are sent to the server in form of sets of commands, or "patches" which are represented in JSON format. There are several types of commands: "create a triplet", "remove a triplet", "remove a concept and all triplets associated with it", "add a concept to a page", "remove a concept from a page". Some commands take arguments. For example, the command to create a triplet accepts three URIs or two URIs and one literal value. So the JSON structure for a patch is an array of arrays.

When the "Save changes" button is pressed, the patch is sent to the ISTINA server using an asynchronous POST HTTP request. There it is processed in the version control system. If the transaction succeeds, the user gets the success message. If the transaction fails, the user gets a list of failed commands in their human readable representation.

7 Applications

Ontologies can be used for many different purposes in large scientific organizations. We will focus on the use cases which have been tested in the ISTINA system and proved their efficiency.

7.1 Data aggregation and manipulation

As we have described earlier in the "Introduction to ISTINA system" section, this system has its own SQL queries generator for extracting data from different tables and grouping it together, which can be used for preparing different reports and for analytic purposes. This generator works automatically in a sense that there is no need to write the whole SQL queries or templates for them. However, there is still a part which needs to be done manually: mapping the database structure to the internal data structure of the generator. When we have the ontology for the internal structure of the system, there is no need to maintain an additional internal structure; it can be extracted directly from the ontology. Also the ontology provides a much richer way to describe the classes and relationships between them, which opens the possibility for adding new features to the generator based on the ontology features, for example classes inheritance or reusing of the same property for different classes can reduce some duplicated code dealing with different types of objects with similar structure.

The ontology can be also used for manipulating data. In terms of SQL, we can build not only SELECT queries, but also UPDATE or INSERT queries in an automated way. For example, the system has a set of wizards for adding new works of different types. These wizards can have up to six steps: for example, for adding a conference talk one must enter the talk data, select the conference from the list of existing ones, if one does not exists, enter the conference data, enter the data about

conference organizers, select the profiles for authors from list of candidates, if one or more authors do not have a profile in the system yet, then enter their data. The number of steps in this case can get even larger if we allow for adding the published thesis of the conference talk, which will involve data about a journal, a collection, a publishing house etc. The ontology of the system allows us building such wizards dynamically based on relations between classes: for example, a conference talk is related to the conference and optionally to an article, which can be related to a journal or a collection, which in turn is related to a publishing house. This produces a graph, an iteration of which is done by the wizard. When the wizard is completed, we can automatically generate a set of SQL queries which will perform the needed changes in the database.

7.2 Data thematic classification

For the purpose of thematic classification of all data entered to the system, we will need the ontologies of subject areas. Every work in the system, for example an article or a conference talk, have a set of keywords associated with it. Every search query can be also split into keywords. We have an algorithm in place which maps the keywords to the closes terms in the ontology [5]. Then for each search query, its results will be the works which have the biggest amount of paths from the set of keywords to the work, and the lengths of the paths are shorter.

Based on the data aggregation system, we can also get sets of authors instead of sets of works, with support for formulae. For example, we can get an authors who have articles in journals with high impact factor on a certain topic. To do this, first every work is given a certain weight based on its thematic classification. Then a formula where different categories of works are having different weights is evaluated, but the final weights of all works according to a formula are multiplied by their weights according to their thematic classification. Finally, sum of all multiplied weights will give a weight for every particular author.

7.3 Using ontologies in the learning process

A lot of work has been done in the direction of using the ontology for automating certain tasks in the learning process.

- The ontology is used for automatically generating parts of some documents and reports. These parts include the course structure, the list of disciplines and topics, and the glossary (list of terms) for every discipline and topic. Not only there is no need to manually write such texts, but the data once entered into the ontology can be later reused for different courses which are being read in different departments of the organization.
- There is a module which can generate sets of exercises for students based on the terms and relations between them. The module supports various types of test questions, some of the basic examples are “select terms which have the given relation with the given object” or “order the terms by a certain relation”. There is also a possibility for interactive tests, when the student is given a set of terms and should

draw the edges between them and map these edges to the ontology relations. The advantage of such test tasks is that there is no fixed set of test variants, all variants are randomized and generated on the fly, so a student cannot just learn the correct answers, he/she should learn the actual terms and relations between them.

- Based on the tests' results, we can build cognitive maps for every student. A cognitive map is a fragment of the ontology where each term and relation is marked as one of the following: not tested, learned incorrectly, learned correctly, not learned. Given the cognitive maps, we can build the individual learning trajectories: topics where the number of correctly learned terms is lower should have higher priority in teaching, and can be tested again later to see if the coverage improves.

Illustration 1 shows how the different approaches to measuring a student's progress differ in the level of detail, from the very basic table which is part of the appendix to diploma, to the ontological approach to the right. It also shows how the two disciplines can intersect and reuse the same part of the ontology.

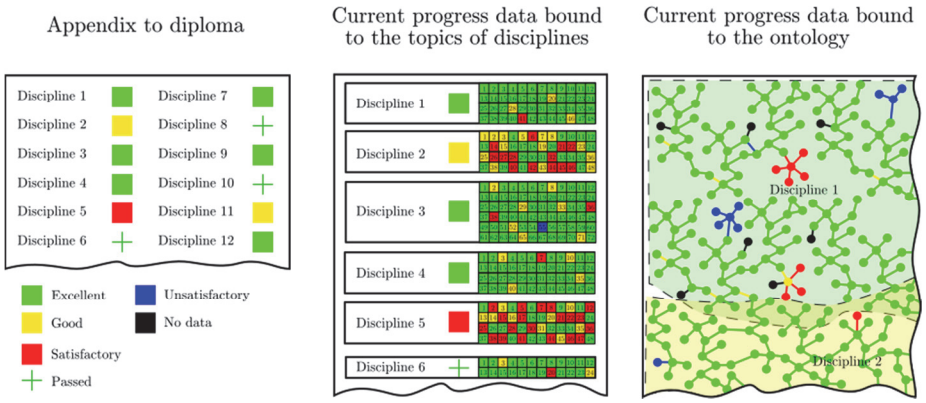

Illustration 1. Change in a detail of student's rating

8 Conclusion

The developed approach provides an editor for ontologies which can be used by many different users, who will not only help populating the semantic map of their subject areas, but also will have their own benefit from it. Many applications useful for high schools are presented, in particular the applications that help to automate the learning process. Ways of using the internal system ontology for aggregating and manipulating the data are also shown, and are useful not only for high schools, but for any system which works with large amounts of data.

References

1. Nicola Guarino, Daniel Oberle, Steffen Staab. *What Is an Ontology?* — In Handbook on Ontologies, 2nd edition — Springer, 2009.
2. Denis Golomazov. *Methods and means for managing the scientific information using ontologies.* — PhD thesis — Lomonosov Moscow State University, 2012.
3. Dmitry Shachnev, Sergey Afonin, Alexander Kozitsyn. *Software mechanisms for scientometrical data aggregation based on ontological representation of the relational database structure.* — Software Engineering, 7(9):408–413, 2016.
4. Viktor Sadovnichy, Valery Vasenin et al. *The Intellectual System of Thematic Investigation of Scientometrical Information ("ISTINA").* — Lomonosov Moscow State University, 2014.
5. Sergey Afonin, Kirill Lunev. *Detecting the thematic vectors in a collection of keywords.* — Software Engineering, (2):29–39, 2015.