A Synchronized Graphical and Source Code Editor for RDF Vocabularies

Alexandra Similea

Matriculation number: 2776909

October 27, 2016

Master Thesis

Computer Science

Supervisors:

Prof. Dr. Sören Auer
Niklas Petersen

Institut für Informatik III

Rheinischen Friedrich-Wilhelms-Universität Bonn

Contents

Chapter 1

Introduction

Chapter 2

Requirements

Chapter 3

Related Work

In this chapter, we will focus on previous work that is related to some extent to the problem stated in Introduction. We will first present a theoretical approach for creating a synchronized graphical and code editor. Next, we will introduce a list of existing tools for editing RDF vocabularies (visually, textually or both). Finally, we will show why visualizing semantic data can raise several problems and we will explain the solutions that were found.

3.1 Theoretical Approach

In the paper "Robust Real-Time Synchronization between Textual and Graphical Editors" [1], Oskar van Rest et al. present an approach which is meant to ease the work with languages that feature both textual and graphical syntax. This work also suggests ways to overcome common synchronization problems such as error recovery and layout preservation.

The main idea of this approach is having an underlying model as a common factor for the two views. Another concept involved in the synchronization process is the abstract syntax tree (AST), which can be regarded as a tree representation of the syntactic structure of the code written in the textual view. Abstract syntax trees are commonly used by compilers during semantic analysis, in order to verify that the elements of the programming language are correctly used.

Figure 3.1 presents a broad overview of this approach. For the textual to graphical view synchronization, the code is parsed into an abstract syntax tree, which is then turned into a model. The resulting model is merged with the one belonging to the graphical view and the this view is updated accordingly. The inverse synchronization process starts with pushing the graphical changes into the model, which is next transformed into an abstract

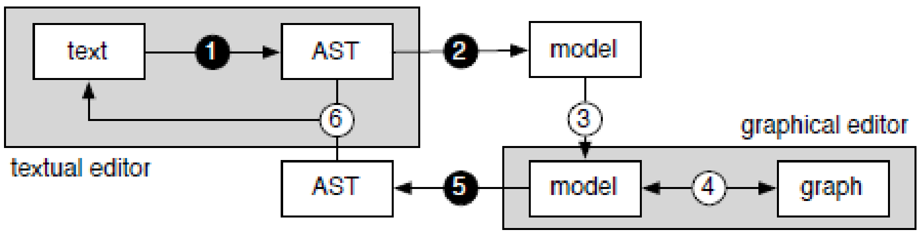


Figure 3.1: Steps involved in the synchronization process: 1) parsing, 2) tree to model transformation, 3) model merge, 4) graphical changes propagation, 5) model to text transformation, 6) tree to text printing.
Figure taken from [1].

syntax tree. The resulting tree is merged with the one belonging to the code view, which will be turned into text.

This approach is not feasible in our case as some steps do not need explicit implementation due to the numerous already existing frameworks for parsing and visualizing RDF data. Therefore, we are not bound to use abstract syntax trees, as RDF data can be easily parsed into an array of triples (one good example is the N3 Javascript library[1]). Moreover, we do not have to use one model for each view. In fact, we did implement the idea of having an underlying model, but, in our case, it is the same for the two views and represents the only connection between them. Both textual and graphical data is parsed directly into the common model and an update process is triggered. More details on this can be found in the Implementation chapter.

3.2 Vocabulary Editors

In this section, we will present a list of tools for authoring and editing RDF vocabularies. Some of them feature only textual editors with the possibility to visualize the result, while the others enable the users to also edit the graphical display. None of them, though, synchronize the changes without user interaction.

1. Vocabulary collaboration and build environment (VoCol)[2] offers a collaborative environment for building ontologies. Vocabulary files storage and versioning control are achieved through repository services like GitHub, GitLab and BitBucket. VoCol comes with a Turtle editor where files can be loaded from the repository and be modified. The changes can

[1]`https://github.com/RubenVerborgh/N3.js`
[2]`https://github.com/vocol/vocol`

be committed only when they pass certain rules of correctness, the editor featuring syntax validation done by tools like Rapper[3] or Jena Riot[4]. An important observation to be made at this point is that our implementation has as prerequisite the Turle editor offered by Vocol, together with its repository services. Furthermore, this environment offers the possibility of visualizing vocabulary elements using WebVOWL[5]. Figure 3.2(a) shows an example of the interface. The graphical view is not editale, though, and can be updated only after the changes were comitted to the repository.

2. Protégé[6] is an ontology and knowledge base editor created by Stanford University and actively supported by the Protégé community. The tool allows the definition of classes, class hierarchies, variables, variable-value restrictions, relationships between classes and the properties of these relationships [2]. Ontologies can be uploaded and downloaded in various formats such as RDF/XML, Turtle, OWL/XML and OBO. Protégé's functionality can be divided into three area: creating ontologies, creating data using the ontology and querying the data. Moreover, the software comes with visualization packages (OntoViz[7], EZPAL[8] and others) and it can create a graphical user interface from the ontology, that is, forms with fields corresponding to elements in the ontology [3]. An example can be viewed in Figure 3.2(b). Protégé features a web extension - Web-Protégé, which aims to better support the collaborative development process in a web environment. The tool provides support for simultaneous editing, where a change made by an user is immediately seen by the other users [4]. Both editors enable modifying the graphical view and the changes can be exported into text files having the previously mentioned formats. Therefore, there is no support for immediate synchronization.

3. IsaViz[9] is a visual environment for browsing and authoring RDF models as graphs. This tool is offered by W3C Consortium [2], but it has not been maintained since 2007. IsaViz comes with an user interface which allows creating and editing graphs, together with zooming and navigation into the model. Figure 3.2(c) shows an example of the interface. The changes occuring in the graphical view can be synchronized with text only through file export. The tool allows importing ontologies in formats like RDF/XML, Notation3 and N-Triple, while the export supports, besides the already men-

[3]`http://librdf.org/raptor/rapper.html`

[4]`https://jena.apache.org/documentation/io`

[5]`http://vowl.visualdataweb.org/webvowl.html`

[6]`http://protege.stanford.edu`

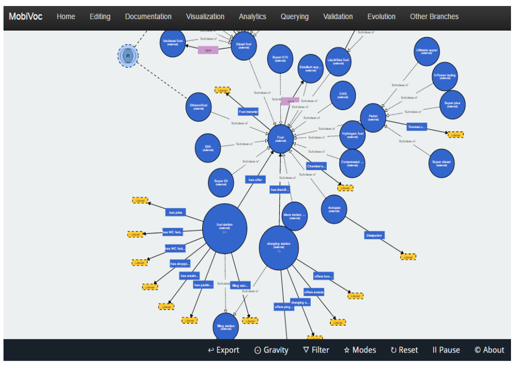
[7]`http://protegewiki.stanford.edu/wiki/OntoViz`

[8]`http://protegewiki.stanford.edu/wiki/EZPal`

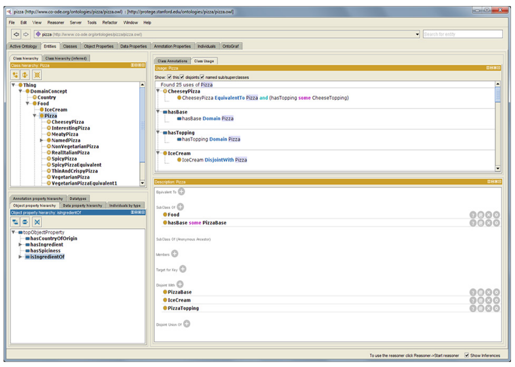
[9]`https://www.w3.org/2001/11/IsaViz/`

tioned formats, also SVG and PNG.

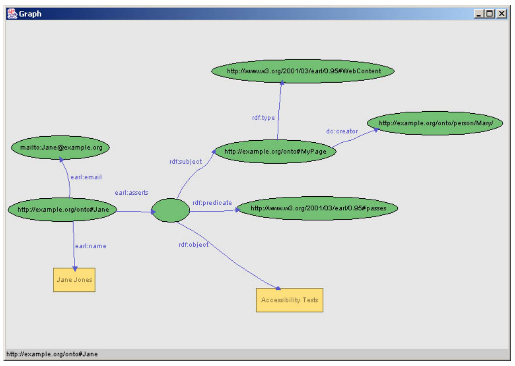
4. RDFauthor is a tool for viewing, creating and querying RDF instance data. It extracts structured information from RDFa-enhanced websites and creates an edit form based on this data. The RDF data model is presented graphically as a directed graph [3]. A visualization example is shown in Figure 3.2(d). In order to store the changes persistently in the triple store that was used to create the RDFa annotations, RDFauthor needs information about the data source (i.e. SPARQL endpoint) regarding the named RDF graph from which the triples were obtained or where they have to be updated [5]. An important contribution is that RDFauthor allows hiding the RDF and related ontology data models from novice users completely, thus allowing more people to author semantic content.



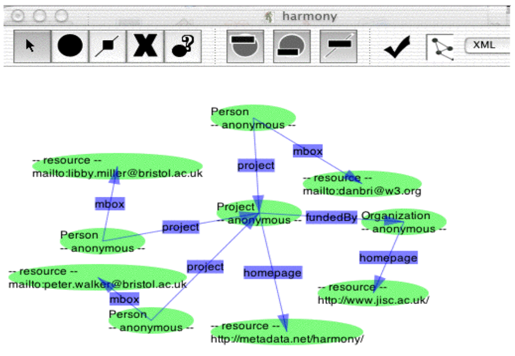
(a) Vocol



(b) Protégé


Graph
http://www.w3.org/2001/03/earl/0.95#WebContent
rdf:type
do:creator
http://example.org/onto/person/Mary
mailto:Jane@example.org
rdf:subject
http://example.org/onto#MyPage
earl:email
earl:asserts
http://example.org/onto#Jane
rdf:predicate
http://www.w3.org/2001/03/earl/0.95#passes
earl:name
rdf:object
Jane Jones
Accessibility Tests
http://example.org/onto#Jane

(c) IsaViz


harmony
XML
Person -- anonymous --
mbox
project
-- resource -- mailto:libby.miller@bristol.ac.uk
-- resource -- mailto:danbri@w3.org
mbox
Project -- anonymous --
fundedBy
Organization -- anonymous --
project
Person -- anonymous --
homepage
project
homepage
-- resource -- mailto:peter.walker@bristol.ac.uk
-- resource -- http://www.jisc.ac.uk/
mbox
Person -- anonymous --
-- resource -- http://metadata.net/harmony/

(d) RDFauthor

Figure 3.2: Graphical user interfaces of different tools for editing RDF vocabularies.

3.3 Visualizing Semantic Data

Most of the tools that realize RDF data visualization generally work well with small models. However, semantic data describing web resources can easily reach thousands of nodes and, at this point, special techniques are needed in order to display the RDF model in such a manner that it is easy to understand and navigate.

GViz [6] is a general purpose visual environment for browsing and editing graph-based data. Its main advantage, compared to most other graph visualisation tools, is that it is easily customizable [7]. Modifying the graph layout is highly flexible, the users being able to choose the shape, color, size and other attributes of the nodes and edges. One interesting feature is the possibility to define callbacks in the Tcl scripting language, in order to further customize nodes and edges depending on certain attributes.

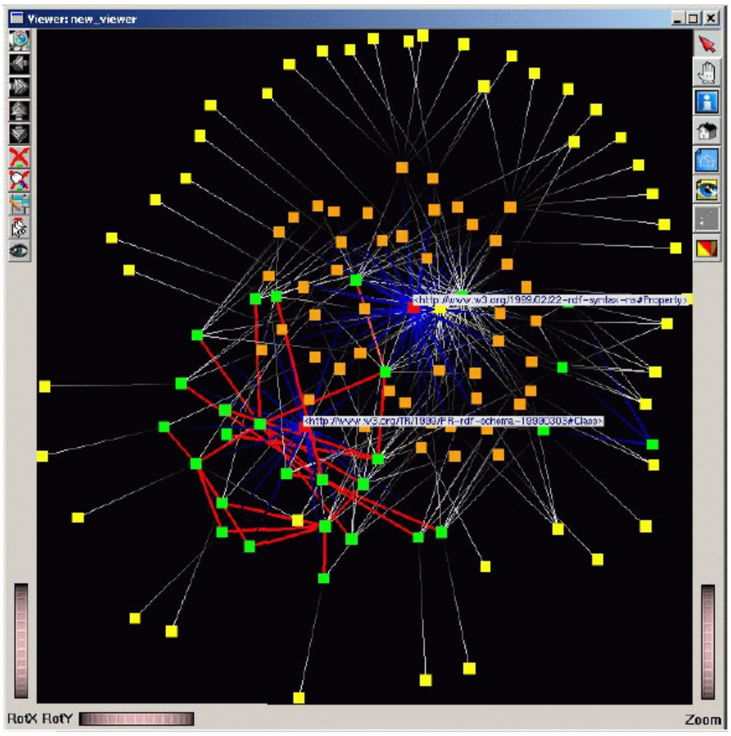

Viewer: new_viewer
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
<http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Class>
RotX RotY
Zoom

Figure 3.3: GViz ontology visualization. Figure taken from [7].

Figure 3.3 presents a visualization example, where choosing different colors makes the navigation more intuitive. Literals are depicted with yellow and displayed at the perifery as they are loose coupled, resources are green and nodes having an edge with the *rdf:Property* value are displayed in orange. Edges are also differentiated through colors, depending on their value:

rdf:type is blue, *rdfs:subClassOf* is red and the rest are white. Another important customization is not displaying the edges as arrows, but as lines fading towards the subject, in order to avoid the clutter produced by highly connected graphs. This approach proves itself very helpful when it comes to easily finding the interesting nodes, i.e. the nodes describing the web resources that the model defines, as they will always stand out due to their different color.

Another approach for intuitive graph visualization was investigated in [8]. The main idea is about using the properties between instances in order to place the related nodes near to each other, while keeping the other nodes evenly distributed. The resulting graph will give the user insight into the structure and relationships in the data model that are hard to see in text [8]. The drawing algorithm uses the spring embedding method [9], which distributes the nodes in a two-dimensional space and, at the same time, keeps the connected nodes reasonably close together. The graph is considered as a force system were each node simulates a charged particle, which causes a repulsive force, and each edge is modeled as a spring that exerts an attractive force between the pair of nodes it connects. Figure 3.4 shows an example of such layout where connected nodes are close together, yet no pair of nodes are too close to each other due to the repulsive forces acting between them [8].

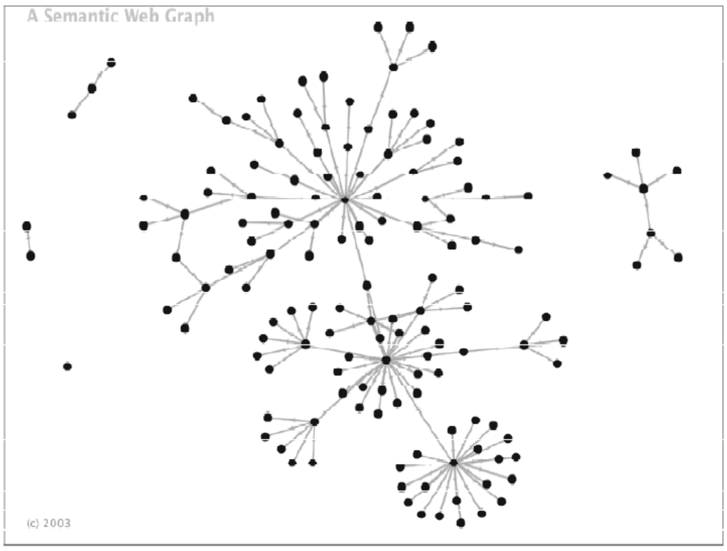


Figure 3.4: Graph layout using spring embedding. Figure taken from [8].

Chapter 4

Implementation

Chapter 5

Evaluation

Chapter 6

Conclusions

Bibliography

[1] O. van Rest, G. Wachsmuth, J. Steel, J. G. Süß, and E. Visser, "Robust real-time synchronization between textual and graphical editors," *Proceedings of ICMT '13*, vol. 7909, pp. 92–107, 2013.

[2] B. Kapoor and S. Sharma, "A comparative study ontology building tools for semantic web applications," *International Journal of Web and Semantic Technology (IJWesT)*, vol. 1, no. 3, pp. 1–13, 2010.

[3] D. Steer, "Meg client software review." `http://www.ukoln.ac.uk/metadata/education/regproj/review`. Accessed: 27-Oct-2016, Last updated: 07-Jun-2002.

[4] T. Tudorache, J. Vendetti, and N. F. Noy, "Web-protégé: A lightweight owl ontology editor for the web," *5th OWL Experiences and Directions Workshop (OWLED 2008)*, 2008.

[5] S. Tramp, N. Heino, S. Auer, and P. Frischmuth, "Rdfauthor: Employing rdfa for collaborative knowledge engineering," in *Knowledge Engineering and Management by the Masses; 17th International Conference, EKAW 2010, Lisbon, Portugal* (P. Cimiano and H. Pinto, eds.), October 2010.

[6] A. Telea, A. Maccari, and C. Riva, "An open toolkit for prototyping reverse engineering visualization," *IEEE EG VisSym '02*, pp. 241–250, 2002.

[7] A. Telea, A. Frasincar, and G.-J. Houben, "Visualisation of rdf(s)-based information," *Proc. of 7th Intl. Conf. on Information Visualization (IV 2003)*, pp. 294–299, 2003.

[8] P. Mutton and J. Golbeck, "Visualization of semantic metadata and ontologies," *Seventh International Conference on Information Visualization (IV03), IEEE*, pp. 300–305, 2003.

[9] T. M. J. Fruchterman and E. Reingold, “Graph drawing by force-directed placement,” *Software – Practice and Experience*, vol. 21, no. 11, pp. 1129–1164, 1991.