LODmilla: shared visualization of Linked Open Data

András Micsik, Zoltán Tóth, Sándor Turbucz

MTA SZTAKI, Budapest, Hungary
{andras.micsik,zoltan.toth,sandor.turbucz}@sztaki.mta.hu

Abstract. Most current visualizations for Linked Open Data are created for a single purpose or a single dataset. These ad hoc approaches can hardly exploit the linkedness of LOD, and we miss the tools for comfortable and enjoyable LOD browsing. On the other hand WWW has mature generic visualization: the web browsers. With the LODmilla browser we try to find the basic commodity features for generic LOD browsing including views, graph manipulation, searching, etc. With our browser users can navigate and explore multiple LOD datasets and they can also save LOD views and share them with other users. This feature set enables the visualization of provenance and context for LOD entities as well as to navigate on the links between entities such as documents or datasets.

1   Introduction

The Semantic Web initiative provided the framework and the tools for sharing machine-understandable data on the Web. A lot of semantic data appeared since then on the Web, but most of these data were segmented and there were no uniform method for connecting facts and resources from different data sets. In 2006 Tim Berners-Lee outlined a set of best practices for publishing and connecting structured data on the Web: the Linked Data principles [1]. This bases the identification and linking of semantic entities on so-called dereferenceable URIs, which can be used to retrieve more and meaningful information on the referenced object. On the other hand, Open Data refers to the open access to data in open (non-proprietary) formats. The merge of the two concepts became very popular in last years and was named as Linked Open Data (LOD) [2]. The LOD cloud diagram [3] recorded the growth of available LOD data, and counted 295 datasets containing more than 31,000 million triples.

Usually, these LOD datasets are built using RDF, which is a general method for conceptual description or modeling of information for web resources. The basic piece of RDF is a triple (or statement) which consists of three elements: a subject, a predicate, and an object. With this approach, we are able to store not only data, but statements about the resources. All the three parts of a statement are resources themselves as well.

Although the Semantic Web and LOD are meant for machine processable data, their use by humans cannot be avoided. Semantic data is often the only available place for the information sought, and furthermore it is usually more accurate and richer than any human-readable representation. Therefore, it is the task of the IT to provide nice and useful visualizations of Semantic Data for humans. The problem with these visualizations is that they are not generic, but ad-hoc; they are capable of presenting limited types of datasets only. While the World Wide Web had its generic visualization method, the web browser since the very beginning, the LOD cloud is still missing a generic visual browser. In this paper we investigate previous generic LOD visualization approaches and present our own ideas in this respect which were implemented in the LODmilla prototype.

2 Related work

There are quite a lot of approaches for presenting LOD data for humans. Most of these however are dedicated to specific purpose and specific datasets (see for example [4]). The obvious solution for all-purpose LOD browsing is a pure text-based approach (e.g. Virtuoso faceted browser or Graphity [5]). In this case, we usually see a single resource with all its referring triples listed. In this case we can read all data properties such as names, birthdate, etc., and we see all connected resources as links. Clicking on a connected resource brings up the tabular or list view of the selected new resource.

The disadvantage of the pure textual approach is that we see one resource only and the graph structure of connections cannot be seen. In case of hypertext, this is not a problem, but in RDF the connections carry much more information, for which users should have an overview. Therefore, the combination of graphical and textual browsers is a more popular approach, and in the rest of the section we provide an overview of these.

One thing that differentiates the applications using the semantic web is the level where it handles the data. As [6] points out, the grouping by the granularity of information can be the following:

- Collection level
- Resource level
- Intra-resource level

While the collection level approach focuses on providing a general overview of a set of data, and mostly used for predictions, the resource level shows the attributes of the individual resources, and visualizes the connections between them, hence it provides more details on individual resources. Intra-resource level approaches show the distribution of the topics and attributes in a single resource, and they are used mostly for deeper analysis. In this work we aim at the resource level and try to point out the strong and weak points of related other work.

LodLive [7] represents the LOD resources and their connections in a graph structure. The visual design here is plain and simple, so it is relatively easy to understand the whole concept. Even so, the resources, represented by circles, do not contain

enough information for the first sight. We only see a circle, with the resource's labels listed in different languages, plus some other circles around it. If we would like to know more about a resource, we have to open its detail box on the right. Here we can see the data properties attached to the selected node in a pre-processed format, for example image URLs are detected and shown, geographical location is extracted and put on map, etc. The major drawback of LodLive is the pure navigation on connections. Connections are grouped by property, and visualized as expanding small circles around the resource circle. This gives a limitation on the number of connections that can be shown, and in fact, LodLive truncates the shown connections to the first 30-40 for each connection type which results in information loss. It is also hard to see where the connection points, as only the resource URIs are shown as a hover for each small circle. Resource URIs can be quite cryptic for humans when they contain numeric identifiers. Therefore, in LODmilla we aim at showing the labels or titles of connection endpoint resources. LodLive has a nice design, but quite often the usability is sacrificed on the design. As an advantage, it is made as a pure HTML5 browser, which can be run in any modern browser.

OOBIAN [8] is a feature-rich, well-designed and useful LOD browser implemented in Silverlight. Technically this is a drawback as Silverlight is not available in all browsers. OOBIAN consists of several views: a graph view, a textual reader, a file explorer and a map. This application combines the visual and text based approaches, but it can't be used for advanced purposes. It is a good LOD browser to jump from one node to another or to filter properties. The main drawback is that we can see only a single resource and its connections in the graph view.

Microsoft Academic Search is a special tool for finding researchers, their publications, and the relations between these. It includes a Silverlight-based graph module named Visual Explorer [9] where we can visualize the connections between people, show the links between co-authors, and see the citations graph of a person. The main problem with this approach is that it is limited to a given scenario and requires the internal database of Microsoft which is not open for any other organizations.

The VisualDataWeb project produced a set of very interesting graphical user interfaces for the Semantic Web [10]. The RelFinder helps to find connection paths between selected resources. This is a very useful function if we want to know how two objects are related to each other. In LODmilla, we implemented a similar function, but with a different solution we are able to find longer paths and paths can also include nodes from different RDF stores.

gFacet and tFacet [10] are the graph-based and textual implementation of faceted browsing of RDF data. SemLens [10] provides tables and plots to analyse trends and correlations in RDF data. These tools cover specific needs for RDF data consumers, and may be applied as add-ons in future generic LOD browsers.

3   A generic LOD browser

Browsing the web is a commodity today with a number of web browsers on the market. These browsers share some default, fundamental controls and functions, which

provide the basic browsing experience, as a result of crystallization during the last two decades. The basic functions are:

- Opening documents,
- Following links in documents,
- Going back to the previously seen document in the history,
- Searching for text in a document,
- Saving a local copy of the document.

Although the LOD has a much shorter existence than WWW, we cannot find a tool that provides comfortable and visual browsing of semantic data and LOD resources. The tools we examined are either built for specific datasets or they are difficult to use and lack important visualization features. We believe that there is a need for generic LOD browsers, with a set of common basic features the users can learn and get used to. This would greatly increase the impact and usefulness of LOD.

The following basic actions for generic LOD browsers have been identified:

- Visual representation of multiple resources and properties (most probably as a graph),
- Opening resources, viewing object and data properties,
- Searching in the resources,
- Searching in the graph,
- Manipulating a set of documents,
- Saving current LOD view,
- Sharing LOD views with others,
- Undoing previous actions (replacement of web browser history).

We implemented LODmilla as our prototype solution for the above listed goals of generic LOD browsing. LODmilla is a graph based browser, running in conventional web browsers, developed using HTML, CSS and Javascript. While it is primarily visual, it also contains textual representations of resource properties in order to combine the best of both worlds. Its goal is to provide a simple, yet feature-rich application for the interactive exploration of LOD content residing in multiple knowledge bases.

By its design, LODmilla does not hide any information available in RDF from the users, but it tries to organize and pre-process presented data. For example, incoming and outgoing properties are grouped by property type, and for the data properties URLs are made clickable, image URLs are shown inline and geographic locations are shown on a map. The work in the field of user experience is still in progress, but our long-term approach is to extend the interface with more advanced operations in a palette-like fashion, which work similarly to usual image manipulation software (e.g. Gimp).

3.1 Frontend

As the goal was to implement a solution for a wide set of browsers, first we had to choose from available drawing options (Flash, Java and Silverlight were excluded because they need separate plug-ins): SVG (Scalable Vector Graphics), HTML5 canvas, or plain HTML+CSS based graphics. Each of these have their own advantages and disadvantages regarding printing, zooming and other features; so finally we found that a mixed approach fits our needs the best. We used jsPlumb [11] as the visual engine, along with jQuery [12] as the most popular Javascript library. JsPlumb uses pure HTML+CSS for visualizing the graph nodes, and SVG for the links between them. With this approach we lost the ability to use built-in zooming and panning, but we could substitute these with our own implementation in Javascript. On the other hand, handling buttons, text and events got much easier with the HTML and CSS basis of the implementation.
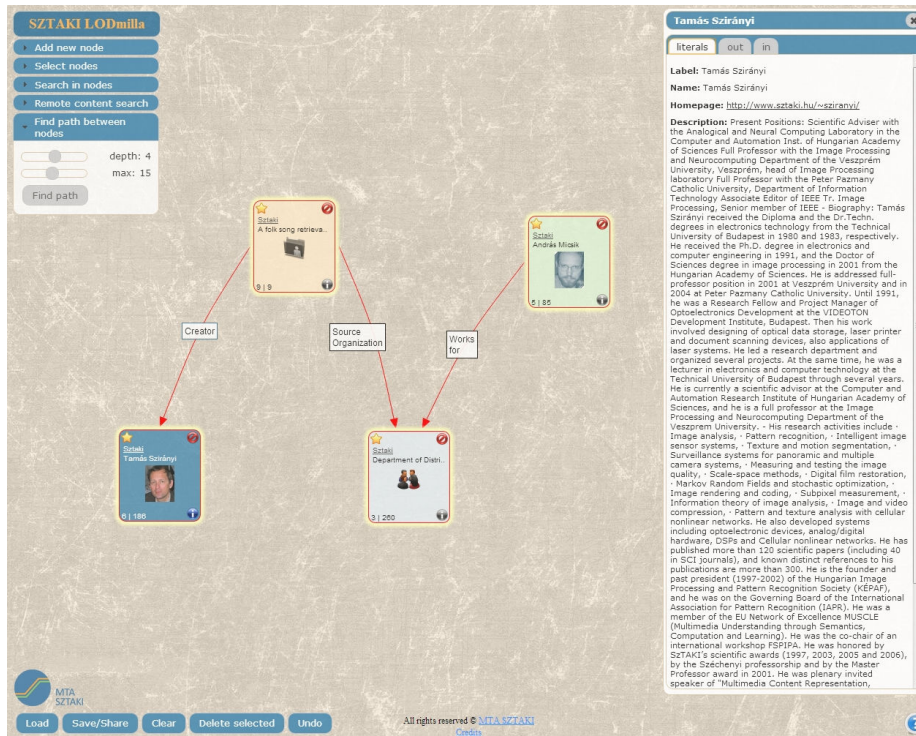

Fig. 1. LODmilla user interface

Figure 1 shows the four main parts of the web application:

- canvas
- palettes (top left)
- toolbar (bottom left)

- inspector (right)

The canvas is the background, on which the graph structure is drawn. The palettes contain various actions, grouped into "accordion" styled menu items. We have a toolbar at the bottom, with the standard operations like load, save, etc. Every node (LOD resource) has a lot of information; these can be viewed in the inspector window on the right, which opens by clicking the "I" (information) button in any node. The inspector window - unlike the other three main elements - can be moved, resized, and closed.

Complex operations can be started from the palettes, while simple ones can be found in every node. Each node has the following basic information in it: the source of the resource, its title or label, and the number of data properties and object properties associated with the resource. Additionally, an image representation (if found) or an icon based on the resource type is put in the middle of the node box. The selection of the image or icon to be shown is based on heuristics, which contain both generic and ad-hoc (dataset-specific) rules.

Nodes have three actions, which affect only the particular node:

- remove it from the canvas
- open details in the inspector window
- select/highlight it (with the star icon)

Furthermore nodes can be moved around, and we can zoom or pan the whole graph view. Actions in the toolbar affect the whole view, while palette actions are used to manipulate a set of (highlighted) nodes or to add new nodes.

Opening resources
The first step in using the LOD browser is to open some nodes. This can be done by pasting a resource URI, but for some datasets we offer an easier way: by typing the label of the resource, we can choose one with autocomplete. We can then start reading the content of the resource in the inspector window, and open some new nodes by clicking on selected connections in the list.

Searching in open resources
Resources may contain a lot of properties and long texts, which we don't want to read through. One palette item serves for searching text patterns in the content of all shown graph nodes. Figure 2 shows this menu in the left menu box, and the search result in the right inspector window, with red colored text.

Searching for new resources
When we don't find the requested information in open nodes, we can try to expand our graph view with new nodes. This is a different search function; it finds the resources which contain the given query word(s), and are connected to a selected resource. Figure 3 shows the use case of searching of the word "KOPI" (a plagiarism search service), from the starting document named "Máté Pataki", shown in the middle. All the resources surrounding are the search results containing the word "KOPI" in their content.

Saving/Sharing/Loading

The graph of currently open resources may demonstrate a new finding, or record a certain state of knowledge, which may be useful in the future for the user who created it. We offer the ability to fully save a graph state under user-given title, and load it later into the browser. Saved graphs can be shared via a unique URL (similarly to Google documents).
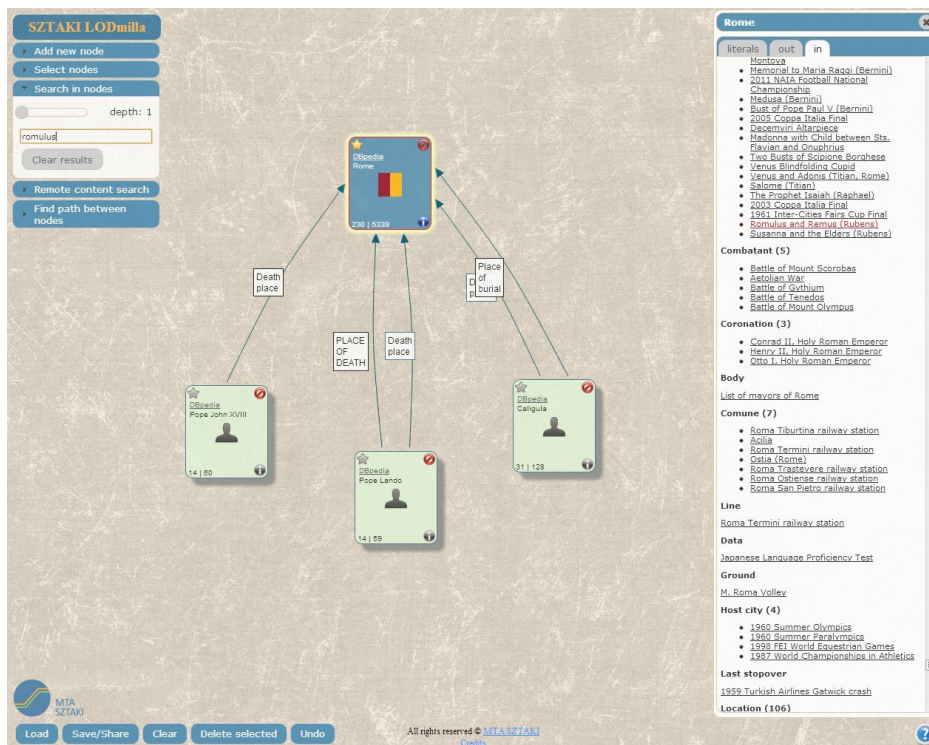

Fig. 2. Finding text in resources

Undo

As some actions may unexpectedly cover the canvas with many new nodes, an Undo function has also been implemented, which reverts the last action the user made.

Marking/highlighting (selection)

All resources on the canvas can be marked one-by-one or in groups. One method to achieve this is clicking on the star in the top-left corner of the nodes. Another method is using the "Select nodes" palette on the left, which can mark all nodes or nodes of the same type. Marking nodes provide the starting points for operations on node groups such as search. For example, we can select the person type resources, and perform a search with the word "KOPI" in those nodes, this way finding the people in

a department, who are somehow related to "KOPI", which is the name of a project running in our department.
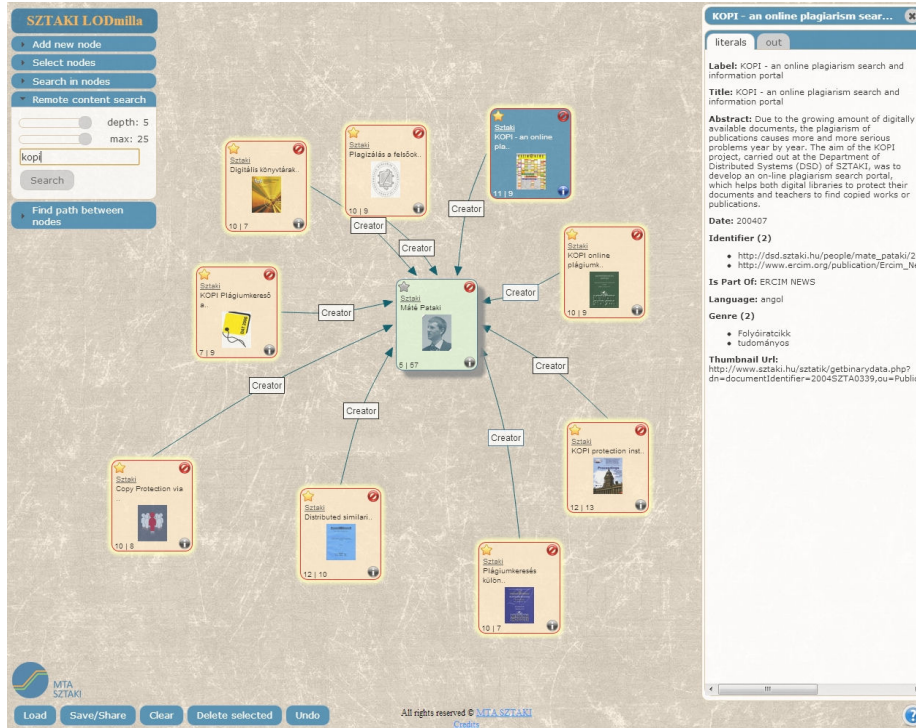

Fig. 3. Expanding the graph via text search

3.2 Backend

Most of the browsing functionality does not rely on a server, and thus our tool could have been run standalone in a browser without a dedicated server. In order to support saving and sharing users' views, we had to implement a server side component as well. The backend has an additional benefit for performance as it can load information faster and cache visited nodes. Finally, we moved most of the search operations to the backend because of these reasons. Search operations use both graph traversal and SPARQL querying, as we cannot always rely on a SPARQL endpoint for datasets.

We put a requirement that our solution should work on as many datasets as possible, and it should use the latest information available, so harvesting and pre-processing datasets was not an option unlike in approaches such as [13]. These presumptions lead us to a graph traversal which can use either a SPARQL query (DESCRIBE for example), or HTTP GET method and dereferenceable URI mechanism to fetch the connections of resources. By using the Jena toolkit, we can parse incoming RDF as Turtle, RDF/XML, JSON, etc.

Three variations of LOD graph search have been implemented:

- Content search: we are searching neighbour nodes with data properties containing the given search pattern,
- Connection search: the nodes are expanded via object properties with names matching the search patterns,
- Path finder: paths are sought between selected nodes.

In all cases we wanted to avoid solutions that work in single datasets only and solutions which use pre-processing of whole datasets. These requirements lead to several problems: first, the quality of the RDF stores is quite different in capabilities, availability and speed, which has big impact on the performance and quality of the graph traversing and building process. Some of the RDF nodes might not be available during the search process, or they can be slowly harvested. The second problem is that the world-wide LOD graph is huge: nodes may have 500 or 1000 connections, and a 2-step path may cover 3 different RDF stores. The third problem also comes from the heterogeneity of our data sources: links between graph nodes sitting at different RDF stores are known only by one of the nodes (i.e. incoming links are not stored).

Because of these limitations, we chose to generate our graphs dynamically. When a user explores a part of the LOD graph, several search operations may be started in a sequence. These queries may be slow at first due to dynamic loading of nodes, but will get faster and faster after the graph area is cached.

As we only see a part of the whole graph at search time, most of the well-known fast graph search algorithms are not applicable in our case. We have to go back to A* style traversal and adapt it to our needs. It is hardly possible to estimate the distance to the goal node, but we can use some heuristics based on connection types. As a specificity of this task, there are paths we are simply not interested in, for example Book resources are all connected to the Book RDF type. Therefore, we simply not follow a set of trivial links denoting type, language or format of nodes. The traversal of the remaining links may be ordered heuristically based on learning, this remains as future work in the project.

In the case of remote *content search* our task is to answer the question: does a node containing a given string in a data property exists in the neighbourhood of a given node? To answer this question breadth-first traversal in the RDF graph structure is applied. We have to limit this algorithm in several ways. We specified a maximum depth until the algorithm tries to find results. As the result set may easily grow to hundreds of nodes, which is incomprehensible for the user, we limit the number of result nodes as well. The result of the search is shown as highlighted nodes in the graph for which the shortest paths from the start node are also displayed.

In the case of *connection search* only those connection types are followed which are matching with the given search text(s). Multiple search items might be added divided by a separator. The traversal of links is also breadth-first. The result of the query is a set of nodes which are accessible via the matching connections. The relevant incoming connections, just like in the case of the content search algorithm, might not be discovered, as these can only be retrieved using SPARQL endpoints, and even

so, we cannot ask each SPARQL endpoint in the world for the existence of such connections.
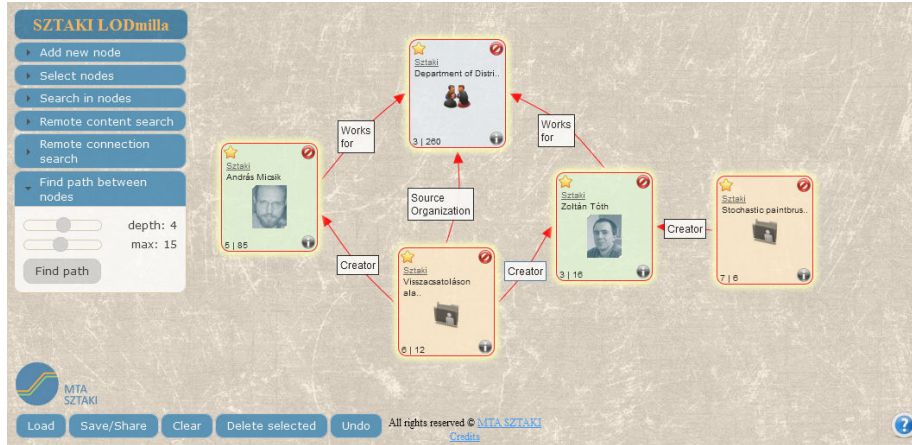

Fig. 4. Finding a path between two resources

For *path finding* the question is if a path exists between two RDF resources (Fig. 4). Our implementation is similar to Dijkstra's algorithm where all edges have the same weight and where the graph is produced on the fly. The starting parameters of our implementation are the two starting nodes, the maximum depth of the search, and the maximum number of nodes we can handle as a response. In the first iteration a breadth-first traversing starts from both endpoints. Their connections are checked and if common nodes can be found, they will be accepted as results and the algorithm finishes. In each further iteration the nodes accessible from the already found nodes are checked. One such iteration means two steps in depth increment since we are growing our graph from both ends. The local target is to find nodes which have parents to both of the source nodes. When such node has been found, the resulting graph must be simplified. All nodes not on the common path are eliminated. In this case it still might happen that we have more nodes in the path than the number we can comfortably handle on the user interface.

Our department operates a LOD server (lod.sztaki.hu) with the contents of Hungarian archives based on the National Digital Data Archive of Hungary. The dataset (11 million triples) contains information about books, movies, articles published in Hungary with links to other datasets such as DBpedia or VIAF. Furthermore, publication data of SZTAKI are also converted to triples. These two datasets serve as the initial testbed for LODmilla, but it was also tested with DBpedia, DBLP and other servers.

Currently LODmilla is available internally in our institute to browse the lodified publication data of colleagues.
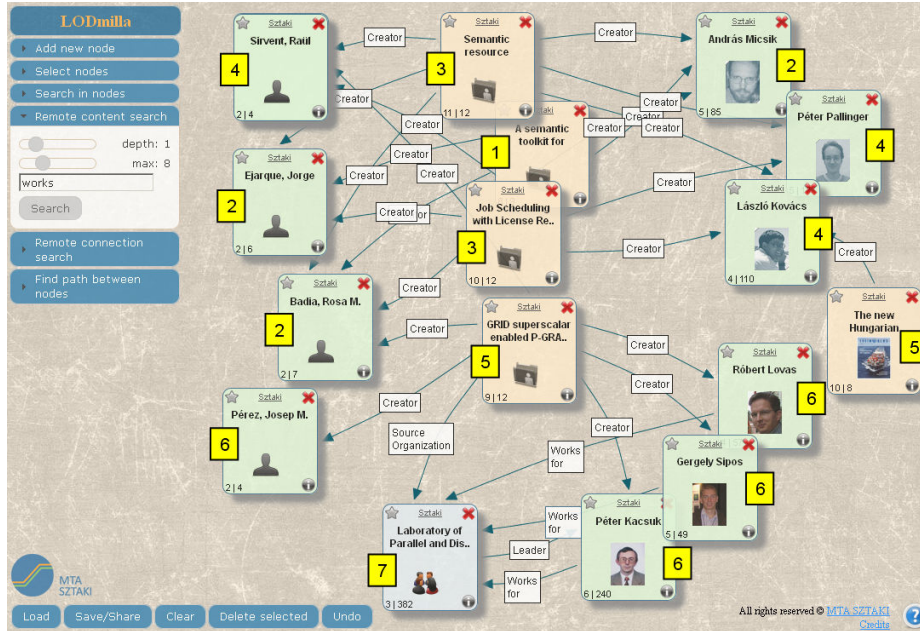
3.3 Example use case

Fig. 5. A sample session of LODmilla demonstrating a use case

In the following we briefly illustrate a use-case for the exploration of LOD space using LODmilla. The numbers in boxes represent the order of appearance for the nodes in Fig. 5. Let's assume that we found an interesting paper, so we enter its keywords into the "Add new node" box, and open the listed semantic description for the paper (1). We can now expand all the Creator type links of this node to see all authors (2). Next, we may check if the authors have created more papers together, and we find two more papers (3). Then we display all authors for the 3 papers, and we get 3 other authors (4). We see that Grid is a common topic in the papers found so far, so we issue a content search on connected nodes for the new authors containing the word Grid, and 2 more papers are shown (5). One of the papers is not linked to anybody from SZTAKI, so we instruct to display all co-authors (6), and get 3 colleagues from a different department (7). As a result, we explored the neighborhood of a selected paper discovering 3 authors groups (one from Spain and two departments from SZTAKI), and found some more papers related to the topic of Grids and the Semantic Web. The graph can now be saved and shared with other colleagues.

4 Conclusion

In this paper we argue that generic tools for exploring and navigating the LOD cloud are necessary not only for computers but for humans as well. The example of Web browsers show that functionality for using a similar technology converges to a com-

mon visualization and a common set of functions. With the LODmilla browser we experiment and test these common functions for generic LOD browsing, based on some key features:

- The link structure of LOD is visualized as a graph,
- Browsing works on several datasets in the same time, connections are also shown between separate datasets,
- Either dereferenceable URIs or a SPARQL endpoint is required,
- Zooming and panning in the graph are supported,
- It is possible to save and share graph views among users.

Additionally, new ways of searching and exploring the LOD graph are needed. LODmilla supports several candidates for these: content search starting from a resource, finding paths between resources, expanding the graph via given connection types, etc.

With this approach LODmilla is capable to handle, and more importantly connect most LOD knowledge bases easily and transparently, and provide a shared knowledge exploration and visualization experience for its users. The features of the browser enable the users to access, visualize and explore all contextual and relational information of LOD resources including research datasets or publications as a particular application area.

Our future plans include creating a plug-in mechanism for advanced operations like searching or automatic organization of nodes on the canvas. We also plan to open up the browser for public use and community-based development.

References

1. T. Berners-Lee: Linked Data - Design Issues. 2006, http://www.w3.org/DesignIssues/LinkedData.html
2. Bizer, C.: The Emerging Web of Linked Data. IEEE Intelligent Systems, Vol.24, no.5, pp.87-92, Sept.-Oct. 2009, doi: 10.1109/MIS.2009.102
3. R. Cyganiak, A. Jentzsch: The Linking Open Data cloud diagram. http://lod-cloud.net/
4. The Best Open Data Releases of 2012, http://www.theatlanticcities.com/technology/2012/12/best-open-data-releases-2012/4200/
5. Graphity, http://graphity.org
6. D. Herrmannova and Petr Knoth. Visual Search for Supporting Content Exploration in Large Document Collections. D-Lib Magazine Volume 18, Number 7/8, July/August 2012, doi:10.1045/july2012-herrmannova
7. LodLive, http://en.lodlive.it/
8. ::oobian::, http://oobian.com/
9. VisualExplorer, http://academic.research.microsoft.com/VisualExplorer
10. VisualDataWeb tools. http://www.visualdataweb.org/tools.php
11. jsPlumb, http://jsplumbtoolkit.com/
12. jQuery, http://jquery.com/
13. De Vocht L, Coppens S, Verborgh R, Vander Sande M, Mannens E, Van de Walle R: Discovering meaningful connections between resources in the web of data. 6th Workshop on Linked Data on the Web, Proceedings. 2013.