# Handling the Complexity of RDF Data: Combining List and Graph Visualization

**Philipp Heim and Jürgen Ziegler**
**(University of Duisburg-Essen, Germany**
**philipp.heim, juergen.ziegler@uni-due.de)**

**Abstract:** An increasing amount of valuable information is stored in RDF. In order to let humans access this information, providing an appropriate visualization of RDF data is an important challenge. In this paper, we present a new approach, combining list and a graph visualization to counterbalance the respective disadvantages of both representation paradigms to better handle the complexity of both the size and the structure of RDF data.
**Key Words:** RDF Visualization, Graph Visualization, Interaction, Data Exploration
**Category:** H.3.3, H.5.2, M.7

## 1 Introduction

Web content is typically distributed. With the use of the Resource Description Framework (RDF), the W3C standard to model information for the Semantic Web, web content can be annotated semantically and thus can be exchanged and integrated in an certain and efficient way. This happens by classifying the information contained in web content to certain classes of an ontology and thereby gives the content a well defined meaning.

In order to allow humans to access the meaning that has been given to certain web content and thereby facilitate their understanding of this content, appropriate techniques to visualize RDF data are needed. Since RDF data is organized as a graph, using a graph to present that data to the user seems to be an appropriate approach. The explicit visualization of the RDF graph, the data as it is stored in the computer, guarantees that all the information is displayed, however, it does not guarantee that all the displayed information can be understood by the user.

The problem with directly presenting the RDF graph is that graph visualization does not scale well to large datasets [Frasincar et al 2006]. Their visualization tends to result in a complex graph, which is hardly manageable or understandable by the user [schraefel and Karger 2006]. Additionally, since RDF data is often highly interconnected [Angles and Gutierrez 2005], visualizing all these connections by edges will cause a lot of edge crossings and hence will further hinder the understanding of the information contained in the RDF graph.

List visualizations by contrast are good in dealing with large datasets. Due to their clear and linear way to display data in combination with sophisticated scrolling and paging techniques as they are implemented in many well known applications, users become empowered to handle large datasets properly. However, lists have very limited abilities in showing multi-dimensional information.

On this account we proclaim a combined approach using both, a list to display large datasets in a clear way and a graph to show multi-dimensional information in its structural complexity. The use of both representation paradigms counterbalances their respective disadvantages and thereby facilitates an enhanced understanding of the information contained in RDF data.

## 2 Related Work

Making semantic information represented in RDF accessible for users has been addressed by several projects.

Graph-based tools such as RDF Gravity [Goyal and Westenthaler 2004], IsaViz [IsaViz 2006], Welkin [Welkin 2005] and the Paged Graph Visualization [Deligiannidis et al. 2007], display RDF data as node-link diagrams, explicitly showing its internal representation structure.

Other tools like Longwell [Longwell 2005], Haystack [Quan et al. 2003], m-Space [schraefel et al. 2005] and Tabulator [Berners-Lee et al. 2006] use lists in combination with the faceted browsing paradigm [Yee et al 2003] to visualize and browse arbitrarily complex RDF datasets, allowing the end users to alter their search criteria at each step of the exploration.

A mixed approach, however, allowing the user to switch from one visualization to the other in a smooth transition, has to our best knowledge not been addressed so far. So with the RDF-ListGraph we present a first attempt in combining a list with a graph to improve the accessibility of RDF data.

## 3 RDF-ListGraph

The RDF-ListGraph is a tool to visualize RDF data as both a list and a graph. The user can switch between list and graph visualization in order to look at the same data in differing ways. The change from one visualization to the other happens in a smooth transition to not confuse the user and thereby rather hinder an understanding of the RDF data than to support it. To provide a smooth transition we use Adobe Flex, an open source framework for building rich Internet applications, using Flash's animation power. The RDF data to be viewed in the RDF-ListGraph is requested by the use of SPARQL, a protocol and RDF query language, allowing to access RDF data from every location with Internet access. The accessed data can then be visualized in any browser with Flash plug-in.

The RDF-ListGraph is developed within the context of the research project Softwiki [Softwiki 2008] that aims to support the collaborative requirement engineering process of large and spatially distributed groups of stakeholders. The stakeholders are enabled to semantically enrich requirements and integrate them in an ontology where central classes like requirement, description and keyword are defined. Classified that way, the requirements are stored in RDF, leading to large RDF datasets. These datasets can be visualized by the RDF-ListGraph to analyze the complexity of the semantically enriched requirements and thus discover interesting relationships between them.

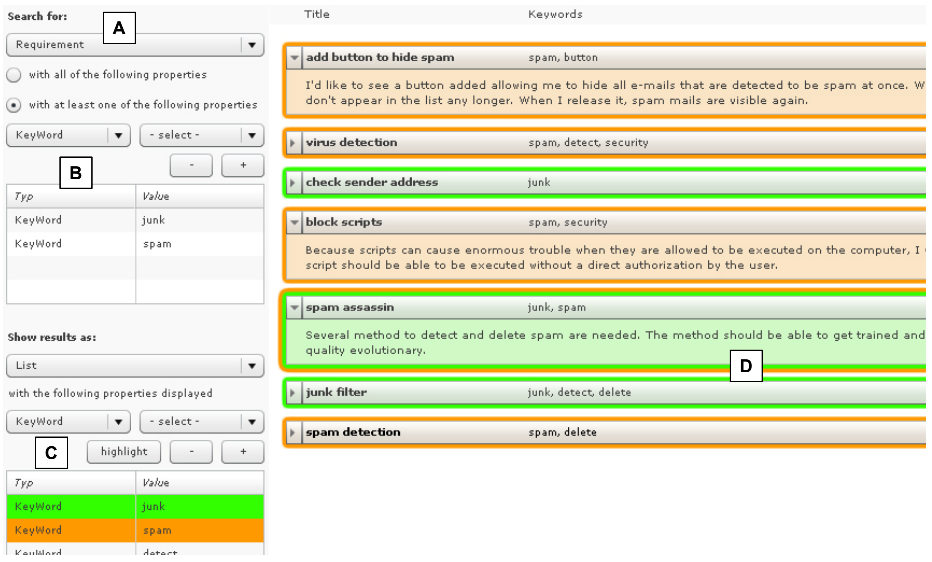
The analysis process with the RDF-ListGraph starts with the definition of the class of instances the user is interested in, for example the class of all requirements. Thereafter, all the existing requirements get visualized as a list, which can be controlled by filters. Having found a set of interesting instances, the user has the opportunity to change the representation paradigm and switch smoothly animated to a graph-based visualization. Since the structural complexity of RDF data can often not be understood in the direct visualization of the RDF graph, a specific graph transformation is applied to enhance users' understanding of this complexity. So the RDF-ListGraph in comparison to existing RDF visualization tools provides the three following key advantages:

1. The combination of list and graph visualization to handle the complexity of RDF data in both size and structure.
2. A smoothly animated transition between both representation paradigms to make the change clear and traceable.
3. A specific graph transformation to directly show similarities between certain instances.

## 4 Using a List to Handle Large Datasets

Since RDF data is often large in size, the RDF-ListGraph has to be capable in dealing with large datasets. Therefore the RDF-ListGraph provides a list visualization together with different kinds of filters to iteratively narrow down a possibly large set of RDF data in order to end up with a small set of instances, which is of special interest to the user.

In order to control what information to display, the user can apply the following three different filters: The *class filter*, the *property filter* and the *display filter*. With the *class filter* the user can narrow down the search space to instances of a certain class, for example requirements (see figure 1, A). The number of instances can then be reduced by the *property filter* to only instances with specific properties, for example to requirements with keywords "spam" or "junk" (see figure 1, B). Additionally, the *display filter* can be used to show or hide certain properties of the found instances and to highlight properties of special interest by different colors, for example the properties "Keyword spam" and "Keyword junk" (see figure 1, C).



**Search for:**

Requirement

( ) with all of the following properties

[✓ with at least one of the following properties]

KeyWord | - select -

| *Typ* | *Value* |
|---|---|
| KeyWord | junk |
| KeyWord | spam |
| | |
| | |

**Show results as:**

List

with the following properties displayed

KeyWord | - select -

| *Typ* | *Value* |
|---|---|
| 🟩 KeyWord | junk |
| 🟧 KeyWord | spam |
| KeyWord | detect |

| Title | Keywords |
|---|---|
| **add button to hide spam** | spam, button |
| I'd like to see a button added allowing me to hide all e-mails that are detected to be spam at once. W don't appear in the list any longer. When I release it, spam mails are visible again. | |
| **virus detection** | spam, detect, security |
| **check sender address** | junk |
| **block scripts** | spam, security |
| Because scripts can cause enormous trouble when they are allowed to be executed on the computer, I script should be able to be executed without a direct authorization by the user. | |
| **spam assassin** | junk, spam |
| Several method to detect and delete spam are needed. The method should be able to get trained and quality evolutionary. | |
| **junk filter** | junk, detect, delete |
| **spam detection** | spam, delete |

**Figure 1:** The list visualization shows information in a clear and linear way.

Each modification of the filters leads to an update of the list, allowing the user to instantly notice the effects caused by the modification. Because a list is a well known representation paradigm and frequent changes of the displayed information do not result in new layouts, but stay in the same linear structure, the user do not get lost while narrow down the list by filters. The possibility to order the list by certain properties can additionally help to force the list in a clear structure to better keep track of changes and updates. To see the instances in more detail, for example to see the description of a requirement (see figure 1, D), the user can open and close them via a dropdown method. It is possible to open several instances simultaneously and hence become able to better compare them in detail. So the list acts as a multi-accordion where instances can be opened and closed user-defined and independently.

Until this point the list is suitable, but lacks capability to present a small set of instances in all its structural complexity. This is mainly the case, because the ability of lists to linearly arrange information has the serious disadvantage that it is not suitable to show non-linear information in all its multiplicity. Only one dimension can be represented properly by a list at one time. So in which way the instances in the set interrelate according to their properties cannot be properly shown by list visualization. In this specific situation, graph visualization seems to be the better visualization paradigm.

## 5 Using a Graph to Handle Structural Complexity

In order to understand the information contained in RDF data, it is important to understand how it is structured. A closer inspection of the data structure can unveil patterns such as dependencies between certain properties. The knowledge of possible dependencies can help to better interpret and understand certain aspects of the information. Since RDF data commonly consists of instances that share multiple properties, their possible dependencies are multi-dimensional. In order to allow the user to discover multi-dimensional dependencies in RDF data, we use a graph visualization to present multi-dimensional data in an appropriate way. The standard RDF graph, however, does not suit the task of discovering dependencies, because equal properties are not directly visualized. To make it easy for the user to spot equal properties and hence facilitate the discovery of dependencies, we transform the standard RDF graph into a graph where shared properties are directly visible.

### 5.1 Graph Transformation

In the transformed graph, only properties shared by at least two instances are visualized. The advantage of this approach is to help users to focus on those properties shared by several instances.

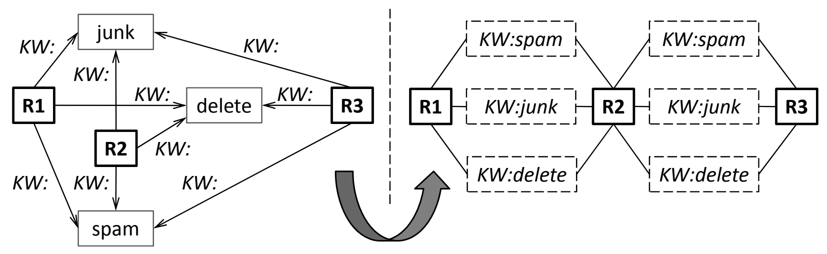


**Figure 2:** The RDF graph is transformed to better discover similarities

A second advantage is that equal properties are directly represented by labeled edges what makes similarities directly visible for the user. In the visualization of the standard RDF graph by contrast, equal properties are not directly visible and hence are hard to spot for the user. As can be seen in figure 2, the requirements R1, R2 and R3 in the transformed graph are connected by labeled edges according to their three equal properties *KW:spam*, *KW:junk* and *KW:delete*, making it easy to recognize similarities between requirements.

A third advantage of the transformed graph is a fewer number of crossing edges. Where in the RDF graph instances with the same property are connected to one and the same node (see figure 2, left), in the transformed graph they are arranged in a row (see figure 2, right). Having several instances connected to one and the same node brings up the question of shortage of space around this node. Since the space around a node is limited, the more instances are connected to the same node, the more they get clustered. With several instances sharing several properties, their clustering results in crossing edges, as it is the case in figure 2, on the left side. The transformed graph reduces the clustering by using a row to arrange instances with the same property instead of using a circle as it is in the RDF graph and thus avoid edge crossings (see figure 2, right).

## 5.2 Graph Visualization

In the graph visualization of the RDF-ListGraph, instances are represented by nodes and connected by labeled edges. To layout the graph in an aesthetically pleasing way, we use a force directed algorithm [Fruchterman and Reingold 1991] to position the nodes of the graph so that all the edges are of more or less equal length and there are as few crossing edges as possible (figure 3).

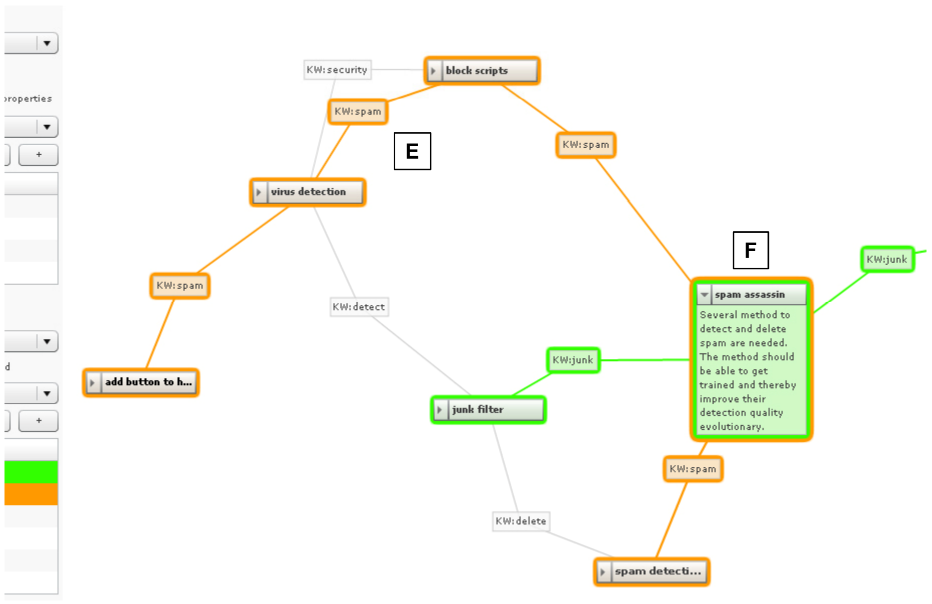


**Figure 3:** The graph visualization shows interrelations directly.

Since even in simple settings the problem of automatic label placement turns out to be NP-hard [Christensen et al 1995], we treat the labels as additional nodes to get the placement solved along with the computation of the force directed layout. Treating a label as a node divides the connections between two instances into two edges with the label as articulated joint in between (see figure 3, E). With a connection consisting of two edges and an articulated joint in between, the shape of an edge gets more flexible and hence avoids overlapping when two instances are connected by more than one edge.

If two instances share a property, they get connected by an edge, which is labeled by that property. If two instances share several properties, they get connected by several edges and therefore get positioned close to each other in the force directed layout of the graph (see figure 3, E). Every property can be highlighted by a specific color, making it easy for the user to see how instances interrelate by certain properties. If an instance interrelates with other instances by more than one property that is highlighted by the user, it gets surrounded by as many colored rings as there are similarities with this instance (see figure 3, F). That way, instances with certain properties are marked by certain colors and thus are easy to find in the graph.

## 6 Conclusions and Future Work

In this paper we introduced the RDF-ListGraph, a tool that offers the user both, a list to handle large datasets and a graph to show the structure of a selected set of instances in all its complexity. To not confuse the user when switching the representation paradigm, the change from a list to a graph and vice versa is smoothly animated in Flash. That way the RDF-ListGraph counterbalances the respective disadvantages of both representation paradigms and let the user handle the complexity of RDF data in both, size and structure.

We also introduced a specific graph transformation that makes similarities directly visible and thereby further facilitate the understanding of the interrelations within the set of instances. Finally, the *class*, the *property* and the *display filter*, allow the user to narrow down large datasets in order to focus on information of special interest.

In our future work we will evaluate our approach by user studies and work on a further integration of the two paradigms to reduce the cost of changing between list and graph visualization. Soften the strict separation between both visualization states seems therefore to be a promising approach. So, offering several stable states in-between list and graph visualization could improve the combination of both representation paradigms and hence further facilitate the understanding of the information contained in RDF data.

## References

[Angles and Gutierrez 2005] Angles, R., Gutierrez, C.: “Querying RDF data from a graph database perspective”; Proceedings of the 2nd European Semantic Web Conference (ESWC), Greece (2005), 346-360.

[Berners-Lee et al. 2006] Berners-Lee, T. Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., Lerer, A., Sheets, D.: “Tabulator: Exploring and Analyzing linked data on the Semantic Web”; Proceedings of the 3rd International Semantic Web User Interaction Workshop (2006).

[Bizer er al. 2006] Bizer, C., Pietriga, E., Karger, D., Lee, R.: “Fresnel: A Browser-Independent Presentation Vocabulary for RDF”; Proceedings of the 5th International Semantic Web Conference, LNCS 4273, Springer, Berlin.

[Christensen et al 1995] Christensen, J., Marks, J., Shieber, S.: “An empirical study of algorithms for point-feature label placement”; ACM Transactions on Graphics (1995), 203-232.

[Deligiannidis et al. 2007] Deligiannidis, L., Kochut, K., Sheth, A.: “RDF data exploration and visualization”; Proceedings of the ACM first workshop on CyberInfrastructure, New York (2007), 39-46.

[Frasincar et al 2006] Frasincar, F., Telea, A., Houben, G.-J.: “Adapting graph visualization techniques for the visualization of RDF data”; Visualizing the Semantic Web (2006), 154-171.

[Fruchterman and Reingold 1991] Fruchterman, T., Reingold, E.: “Graph drawing by force-directed placement”; Softw. Pract. Exper., John Wiley & Sons, New York, 1129-1164.

[Goyal and Westenthaler 2004] Goyal, S. and Westenthaler, R.: RDF Gravity; Salzburg Research (2004). http://semweb.salzburgresearch.at/apps/rdf-gravity/.

[IsaViz 2006] IsaViz: A Visual Authoring Tool for RDF (2001-2006). http://www.w3.org/2001/11/IsaViz/.

[Longwell 2005] SIMILE: Longwell RDF Browser (2003-2005). http://simile.mit.edu/longwell/.

[Quan et al. 2003] Quan, D., Huynh, D., Karger, D.: “Haystack: A Platform for Authoring End User Semantic Web Applications”; Proceedings of the 2nd International Semantic Web Conference (2003), 738-753.

[schraefel and Karger 2006] schraefel, m. c. and Karger, D: “The Pathetic Fallacy of RDF”; International Workshop on the Semantic Web and User Interaction (2006).

[schraefel et al. 2005] schraefel, m.c., Smith, D., Owens, A., Russell, A., Harris, C., Wilson, M.: “The evolving mspace platform: leveraging the semantic web on the trail of the memex”; Proceedings of the sixteenth ACM conference on Hypertext and hypermedia, New York (2005), 174-183.

[Softwiki 2008] Softwiki (2006-2009). http://www.softwiki.de.

[Welkin 2005] SIMILE: Welkin (2004-2005). http://simile.mit.edu/welkin/.

[Yee et al 2003] Yee, K.-P., Swearingen, K., Li, K., Hearst, M.: “Faceted metadata for image search and browsing”; Proceedings of the ACM Conference on Computer-Human Interaction (2003).