Node-centric RDF Graph Visualization

Craig Sayers
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto

HPL-2004-60
April 7, 2004*

RDF, visualization, Resource Description Framework, graph, browser, node-centric

This paper describes a node-centric technique for visualizing Resource Description Framework (RDF) graphs. Nodes of interest are discovered by searching over literals. Subgraphs for display are constructed by using the area around selected nodes. Wider views are created by sorting and displaying nodes based on the number of incoming and outgoing arcs.

The visualization provides a consistent left-to-right orientation for node connectivity and the user browses by simply selecting nodes or arcs of interest.

An example interaction is presented along with a discussion of the design and implementation tradeoffs. Providing an acceptable interactive response time required introducing a preprocessing step and trading space for speed by creating large cross-referenced indexes.

1 Introduction

In the Resource Description Framework (RDF) information is encoded as a set of statements about resources. Those statements may be abstractly viewed as a graph [5].

While visualizing tree-like structures (such as filesystems or XML documents) is relatively easy, visualizing a graph is complicated by the possible presence of cycles and the fact that there is no unique root node. The difficulties of graph visualization have been well studied. See [4] for review.

In the case of RDF, one natural technique is to translate the RDF statements into a graph notation and then view that using standard graph visualization tools (such as GraphViz developed at AT&T [1]). There have been a number of implementations of this approach - see for example Pietriga's IsaViz [9] or Brinkley's RDFViz [2]. There are also several graph visualization tools designed for use with ontologies - see for example the range of plugins for the Protégé ontology editor [10].

These graph-based techniques work very well for small graphs, but as the graphs become larger it is increasingly difficult to find a layout which is readable. This is not unique to RDF. It is merely a consequence of the general difficulty in rendering graphs well.

It is also possible to view RDF as plain text using one of the RDF serialization syntaxes [6, 3]. While these can work well for small files, it can be difficult to see relationships between nodes scattered throughout the serialization. For example it is relatively easy to see the arcs from a node, but very hard to see all the arcs to a node.

The text-based view can be improved using a text-based browser such as BrownSauce [13]. This focuses on a node and shows related information in a human-readable form. Other text-based alternatives include viewing the RDF resources in a tree-like filesystem view as in the RDF Model Browser [12] or using ontological information to view RDF using a faceted browser (for an introduction to faceted browsing see [16]).

In this paper we describe a graphical tool for visualizing arbitrary RDF graphs. A search tool is provided to find a portion of the graph likely to be of interest, and the system then focuses on providing a clear graphical visualization of that portion using a node-centric view.

The paper begins with an example visualization, discusses the design and implementation and then provides some recommendations regarding the handling of additional information about a resource.

2 Example Interaction

To show an example interaction we'll look over the shoulder of a user browsing the rdf-schema (this is a relatively small RDF file, we choose it to ease the task for others who may wish to compare our visualization against alternatives, and we note that the nature of our visualization means it is relatively insensitive to graph size). The user begins with a search. In this case, by selecting the file from a drop-down list and entering `container*` into the search box. The results are shown in Figure 1.

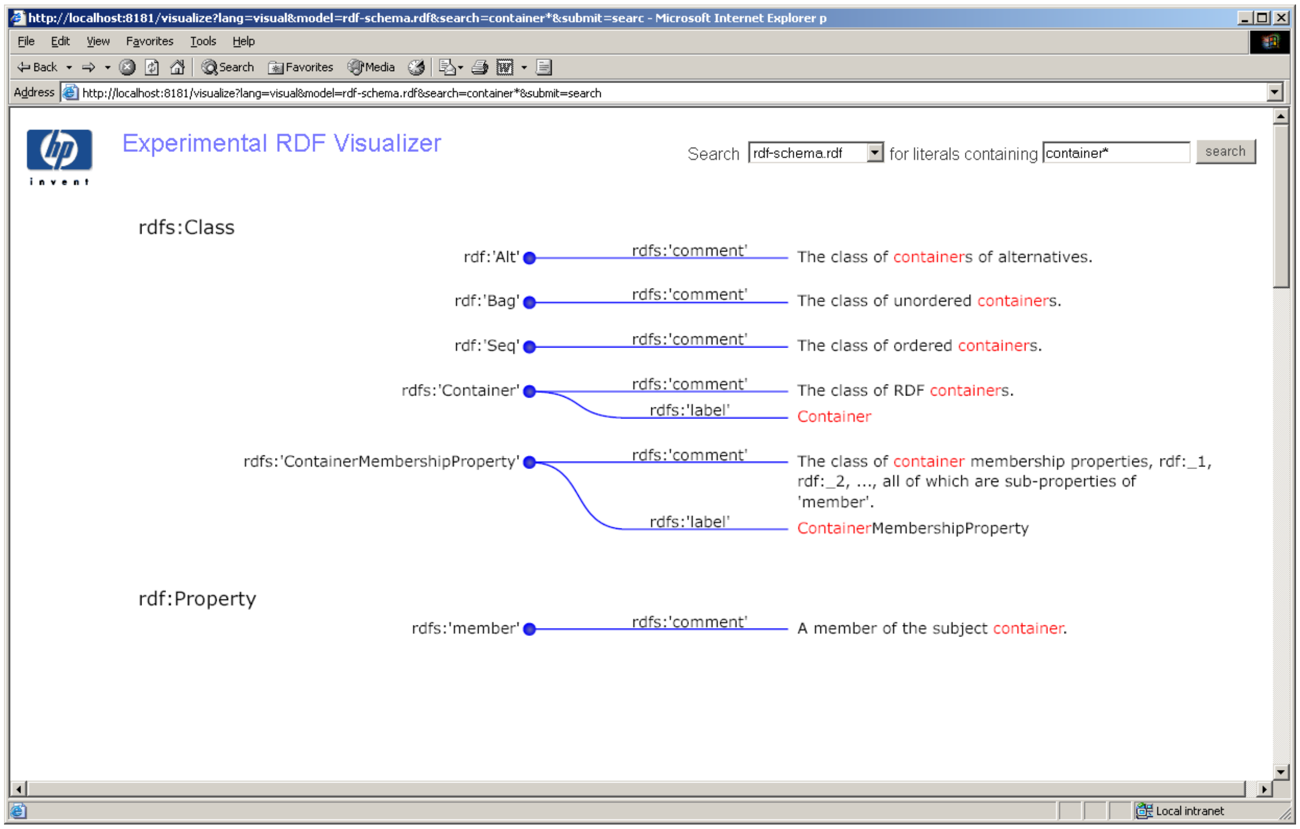
The resulting display, returned in a fraction of a second, shows the matching literals and also the nodes one arc back from those literals. It has also sorted those nodes by type, listing those of type `rdfs:Class` first, then those of type `rdfs:Property`. In this case, because one end of each arc is a literal, the direction of the arcs is unambiguous, but more generally the direction in our visualizations is always left-to-right.

Each node in this diagram is a live link, and the user may click on it for more information. For example, when the user clicks on the node `rdfs:Container` the display refreshes to show the graph around that node (see Figure 2).

We display one arc backwards and two arcs forwards from the selected node. Again, all arcs go left-to-right, so we can see that there are three things (Alternate, Bag, and Sequence) which are subclasses of a Container; that a Container is itself a subclass of Resource, and that a Container is also of type Class.

If the user now clicked on the node `rdfs:Class` they would see the display shown in Figure 3.

Once again, we show one arc back and two arcs forward. By examining nodes on the left we see that a Container is just one of many things which are of type Class; while examining nodes on the right we see information about



http://localhost:8181/visualize?lang=visual&model=rdf-schema.rdf&search=container*&submit=searc - Microsoft Internet Explorer p

http://localhost:8181/visualize?lang=visual&model=rdf-schema.rdf&search=container*&submit=search

Experimental RDF Visualizer

Search rdf-schema.rdf for literals containing container*

rdfs:Class

rdf:'Alt' — rdfs:'comment' — The class of containers of alternatives.

rdf:'Bag' — rdfs:'comment' — The class of unordered containers.

rdf:'Seq' — rdfs:'comment' — The class of ordered containers.

rdfs:'Container' — rdfs:'comment' — The class of RDF containers.
rdfs:'label' — Container

rdfs:'ContainerMembershipProperty' — rdfs:'comment' — The class of container membership properties, rdf:_1, rdf:_2, ..., all of which are sub-properties of 'member'.
rdfs:'label' — ContainerMembershipProperty

rdf:Property

rdfs:'member' — rdfs:'comment' — A member of the subject container.

Figure 1: Display after searching the rdf-schema for the literal text “container*”. Matching text is highlighted in each literal.

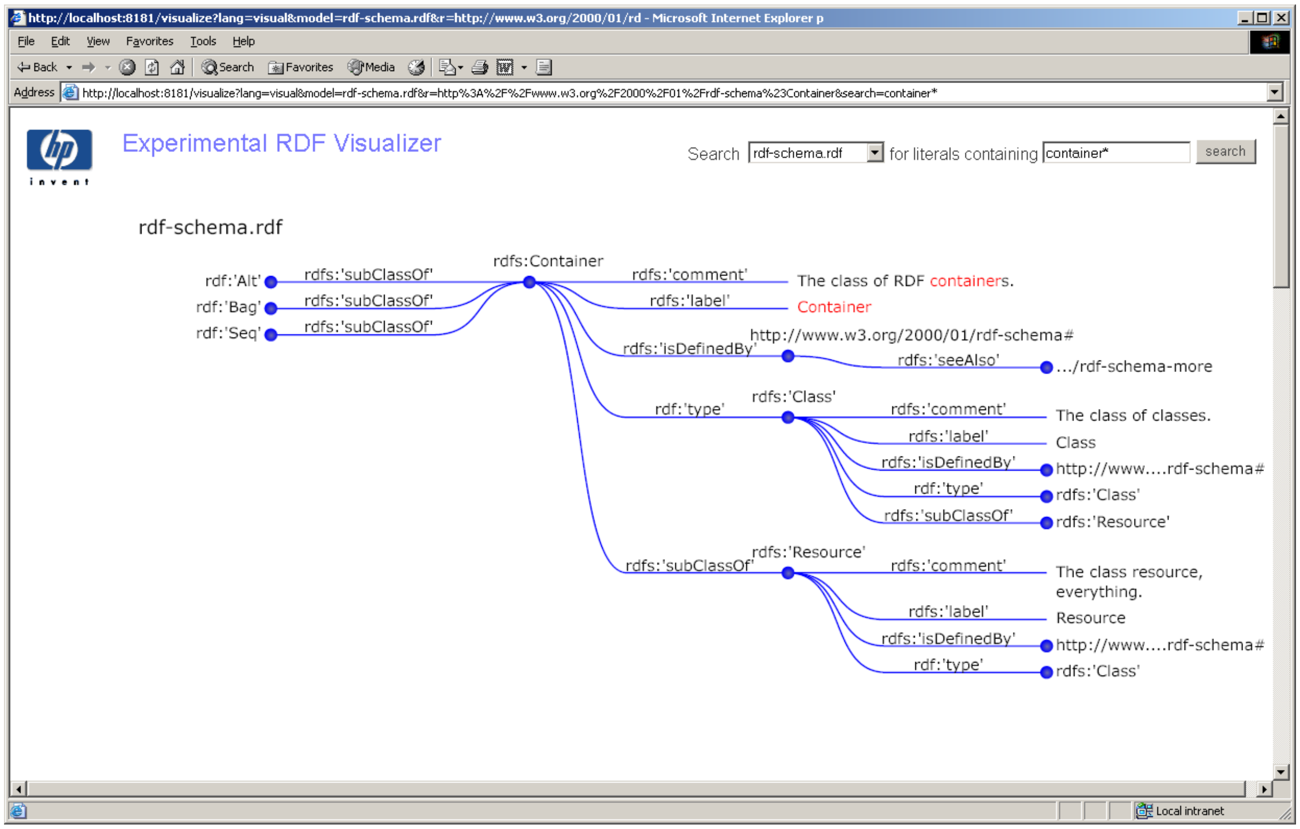

http://localhost:8181/visualize?lang=visual&model=rdf-schema.rdf&r=http://www.w3.org/2000/01/rd - Microsoft Internet Explorer p
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8181/visualize?lang=visual&model=rdf-schema.rdf&r=http%3A%2F%2Fwww.w3.org%2F2000%2F01%2Frdf-schema%23Container&search=container*
invent
Experimental RDF Visualizer
Search rdf-schema.rdf for literals containing container* search
rdf-schema.rdf
rdf:'Alt'
rdf:'Bag'
rdf:'Seq'
rdfs:'subClassOf'
rdfs:'subClassOf'
rdfs:'subClassOf'
rdfs:Container
rdfs:'comment'
The class of RDF containers.
rdfs:'label'
Container
rdfs:'isDefinedBy'
http://www.w3.org/2000/01/rdf-schema#
rdfs:'seeAlso'
.../rdf-schema-more
rdf:'type'
rdfs:'Class'
rdfs:'comment'
The class of classes.
rdfs:'label'
Class
rdfs:'isDefinedBy'
http://www....rdf-schema#
rdf:'type'
rdfs:'Class'
rdfs:'subClassOf'
rdfs:'Resource'
rdfs:'subClassOf'
rdfs:'Resource'
rdfs:'comment'
The class resource, everything.
rdfs:'label'
Resource
rdfs:'isDefinedBy'
http://www....rdf-schema#
rdf:'type'
rdfs:'Class'
Local intranet

Figure 2: Display after selecting the node `rdfs:Container`. Here the system is showing one arc back and two arcs forward from the selected node.

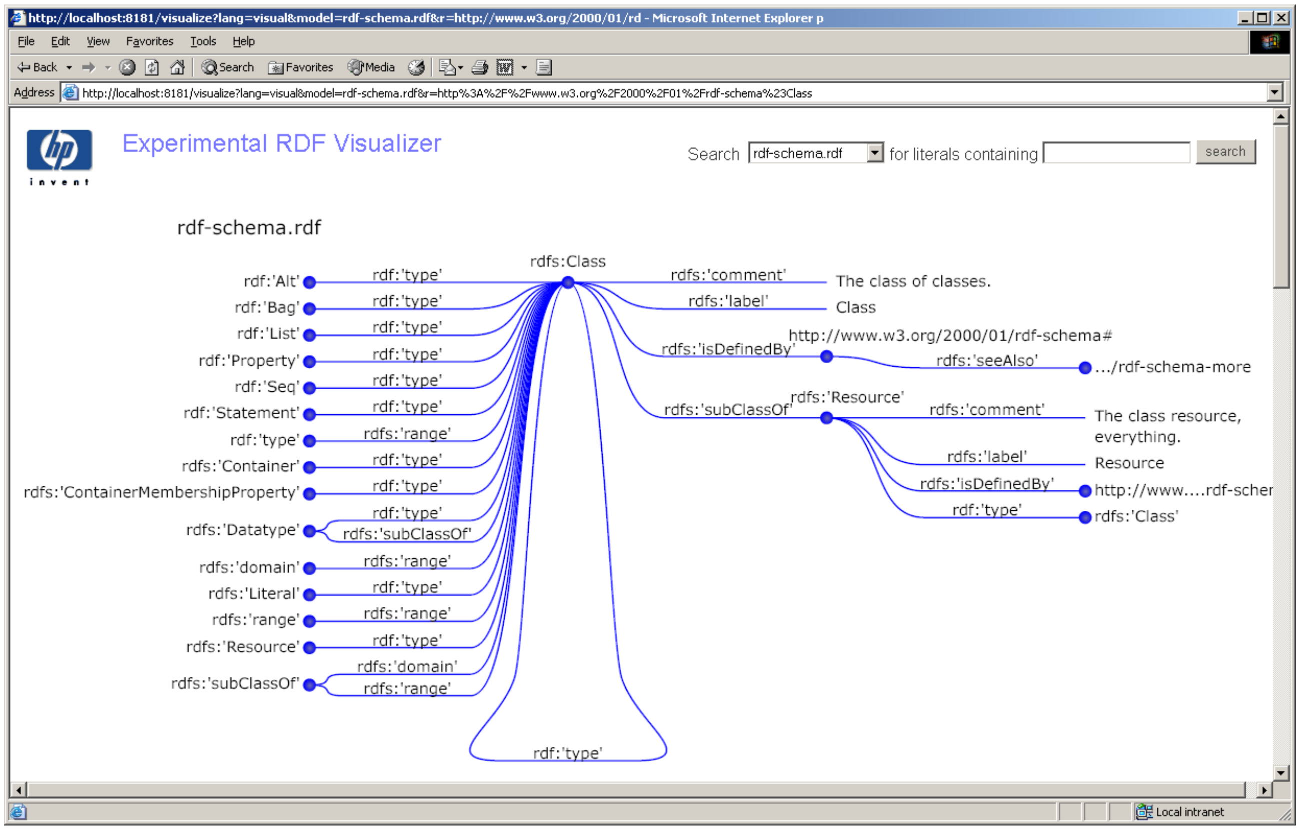


Figure 3: Display after selecting the node `rdfs:Class`. Again one arc back and two arcs forward are displayed. Notice that a Class is of type Class resulting in a circular arc.

the definition of a Class. Notice that the display is focused on a particular node, showing arcs to and from that node, including circular arcs.

To keep the display readable, we do not attempt to collapse nodes which are reachable by both forward and backward arcs, nor do we attempt to show each arc exactly once. Rather we show the selected node and then all nodes reachable by following an arc backward from that node and all nodes reachable by following arcs forward up to two steps from the selected node. Referring to the figure, note that the node `rdfs:Resource` appears both on the left and right of the diagram (since it is reachable both by following an arc backwards and by following an arc forwards). Notice also that the statement:

```
    rdfs:Resource  rdf:type  rdfs:Class .
```

also appears twice - once on the left-hand-side as an arc arriving at the rdfs:Class node and once on the right-hand-side leaving the rdfs:Resource node. We view this duplication as desirable since it simplifies the view while keeping everything consistent. We do collapse duplicate nodes that would otherwise be neighbors in the diagram - an example is `rdfs:Datatype` since in those cases there is no need to compromise consistency.

Each displayed arc is also a live link. Examining each arc carefully, you'll see that the system has done some work to generate those names. It shows the namespace and then a human-readable label rather than the full URI. In general, the format for a displayed node or predicate URI is selected from:

1. *namespace*: '*label*'
2. *namespace*: *localname*
3. *uri*

The first option is preferred in cases where a label is available (based on the presence of an `rdfs:label` property)[1]. The only exception is when we're generating a visualization centered on a particular node. In that case the `rdfs:label` property is already visible as a property of the selected node and we prefer not to hide the localname from the user. If no namespace is available then we use the full URI.

[1]An idosyncrasy of the current implementation is that we only look for labels within the file being displayed. In the future it would obviously be preferable to use node labels from any related graphs.

For more information on any arc, the user simply clicks on it. Continuing this example interaction, if the user clicked on the arc `rdfs:subClassOf` they would see the display shown in Figure 4.

Here the display shows instances of the subClassOf predicate and then its definition. Notice that we've made a very natural move here from viewing statements, to viewing an ontology which describes the properties of those statements. In this case the instances of the predicate and its definition are all in the same file, but there is also additional information about that node in the OWL ontology and that is also listed in the visualization.

In the example so far we started with a search. Sometimes, when viewing unfamiliar files, it can be difficult to know where to start. To assist in such situations we also provide a view of all nodes in the model. The user accesses this by simply clicking the search button without entering any search text. An example is shown in Figure 5.

Rather than trying to display all the nodes as a graph, we instead sort the nodes based on the number of incoming or outgoing arcs: nodes with no incoming arcs are shown on the left of the screen; those with no outgoing arcs are shown on the right; all others fall in the middle. This gives users a quick summary of the graph and the nodes in the center are generally good candidates for a node-centric visualization. Again, in this view all the nodes are live links and any may be selected for more detailed browsing.

3 Design

Having decided on a node-centric view, the design choices revolved around how best to display the information around a node. We first experimented with graph-based approaches, but the display was problematic: finding a suitable layout meant compromising consistency (both for the layout of arcs/nodes within one visualization and when navigating from node to node). We quickly settled on a tree-based approach. Treating the selected node as the root and forming trees of nodes visible by following arcs either forward or backward. This causes duplication in both nodes and arcs, but that was considered preferable to the alternative.

A characteristic of RDF is that while arc labels can usually be shortened (using namespaces and `rdf:label` tags in the ontology) node labels are

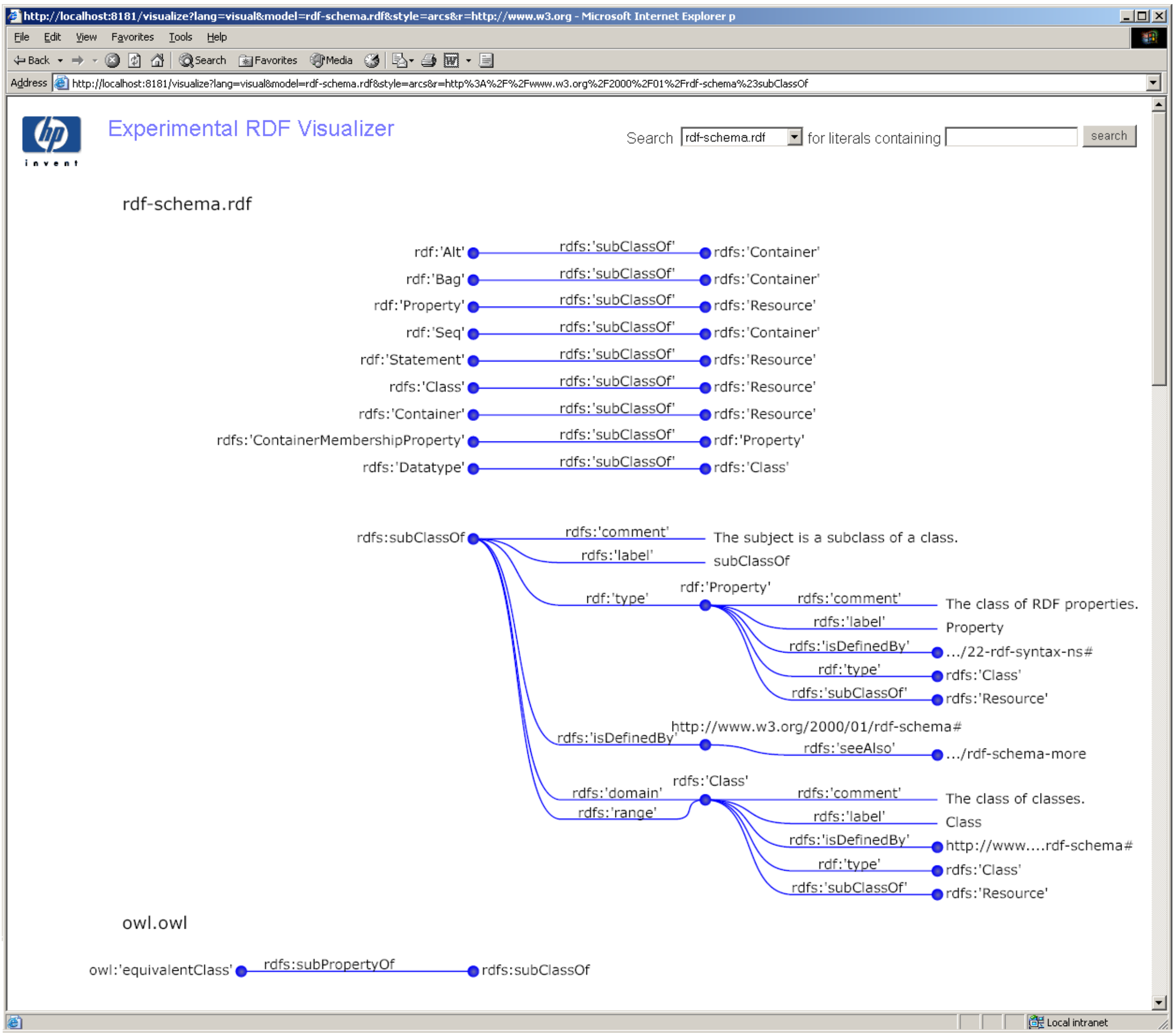
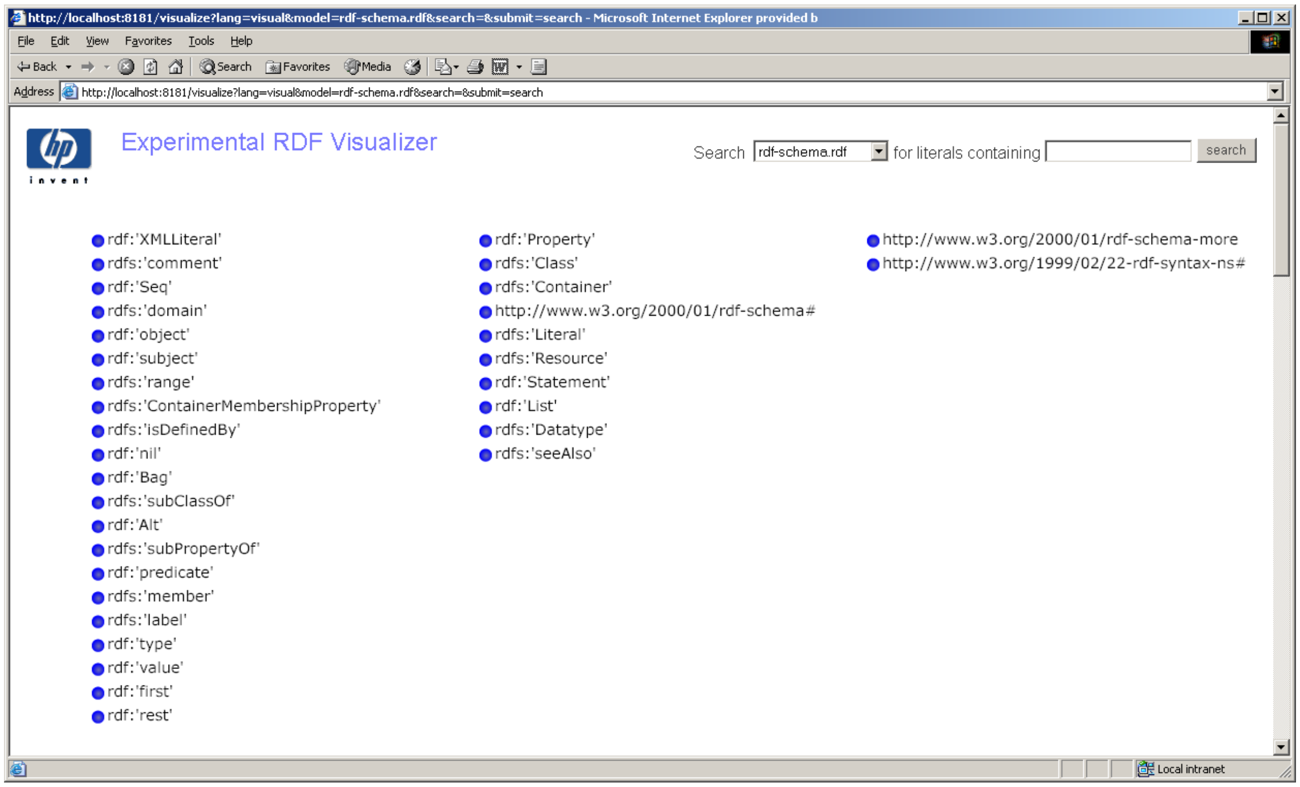

http://localhost:8181/visualize?lang=visual&model=rdf-schema.rdf&style=arcs&r=http://www.w3.org - Microsoft Internet Explorer p
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8181/visualize?lang=visual&model=rdf-schema.rdf&style=arcs&r=http%3A%2F%2Fwww.w3.org%2F2000%2F01%2Frdf-schema%23subClassOf
Experimental RDF Visualizer
Search rdf-schema.rdf for literals containing search
rdf-schema.rdf
rdf:'Alt' rdfs:'subClassOf' rdfs:'Container'
rdf:'Bag' rdfs:'subClassOf' rdfs:'Container'
rdf:'Property' rdfs:'subClassOf' rdfs:'Resource'
rdf:'Seq' rdfs:'subClassOf' rdfs:'Container'
rdf:'Statement' rdfs:'subClassOf' rdfs:'Resource'
rdfs:'Class' rdfs:'subClassOf' rdfs:'Resource'
rdfs:'Container' rdfs:'subClassOf' rdfs:'Resource'
rdfs:'ContainerMembershipProperty' rdfs:'subClassOf' rdf:'Property'
rdfs:'Datatype' rdfs:'subClassOf' rdfs:'Class'
rdfs:subClassOf
rdfs:'comment' The subject is a subclass of a class.
rdfs:'label' subClassOf
rdf:'type' rdf:'Property'
rdfs:'comment' The class of RDF properties.
rdfs:'label' Property
rdfs:'isDefinedBy' .../22-rdf-syntax-ns#
rdf:'type' rdfs:'Class'
rdfs:'subClassOf' rdfs:'Resource'
rdfs:'isDefinedBy' http://www.w3.org/2000/01/rdf-schema#
rdfs:'seeAlso' .../rdf-schema-more
rdfs:'domain' rdfs:'range' rdfs:'Class'
rdfs:'comment' The class of classes.
rdfs:'label' Class
rdfs:'isDefinedBy' http://www....rdf-schema#
rdf:'type' rdfs:'Class'
rdfs:'subClassOf' rdfs:'Resource'
owl.owl
owl:'equivalentClass' rdfs:subPropertyOf rdfs:subClassOf
Local intranet

Figure 4: Display after selecting the arc `rdfs:subClassOf`. This shows instances of that arc in the chosen file and then the definition of that predicate.


http://localhost:8181/visualize?lang=visual&model=rdf-schema.rdf&search=&submit=search - Microsoft Internet Explorer provided b
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8181/visualize?lang=visual&model=rdf-schema.rdf&search=&submit=search
invent
Experimental RDF Visualizer
Search rdf-schema.rdf for literals containing search
Local intranet

- rdf:'XMLLiteral'
- rdfs:'comment'
- rdf:'Seq'
- rdfs:'domain'
- rdf:'object'
- rdf:'subject'
- rdfs:'range'
- rdfs:'ContainerMembershipProperty'
- rdfs:'isDefinedBy'
- rdf:'nil'
- rdf:'Bag'
- rdfs:'subClassOf'
- rdf:'Alt'
- rdfs:'subPropertyOf'
- rdf:'predicate'
- rdfs:'member'
- rdfs:'label'
- rdf:'type'
- rdf:'value'
- rdf:'first'
- rdf:'rest'

- rdf:'Property'
- rdfs:'Class'
- rdfs:'Container'
- http://www.w3.org/2000/01/rdf-schema#
- rdfs:'Literal'
- rdfs:'Resource'
- rdf:'Statement'
- rdf:'List'
- rdfs:'Datatype'
- rdfs:'seeAlso'

- http://www.w3.org/2000/01/rdf-schema-more
- http://www.w3.org/1999/02/22-rdf-syntax-ns#

Figure 5: Display showing the whole model. Nodes on the far left have no incoming arcs; while those on the far right have no outgoing arcs. As in the previous examples, all connections go from left-to-right and all nodes are live links.

frequently less amenable to abbreviation. As a result, we choose a horizontal layout and kept the selected node label at the top (so it can extend far to the right without overlapping other information).

We also experimented with different levels of information and found that one arc back and two arcs forward provided a reasonable compromise. Showing a larger subgraph would have required using a smaller font size; which showing any less provided insufficient context around the selected node.

One drawback to the node-centric visualization is that it only shows a small fraction of the graph. This works well if you need detail or if you have a specific term to search for. However, if you have little knowledge of a file's contents, then it is helpful to have more context with a wider view. To this end we experimented with mechanisms for displaying a larger set of nodes. Again considering consistency in the left-to-right arc direction to be paramount, we chose to position the nodes based on their incoming and outgoing arcs. The root nodes (with no incoming arcs) were placed on the far left, while terminal nodes (with no outgoing arcs) were placed on the far right of the screen. All other nodes fall in the middle and, having both incoming and outgoing arcs, and are thus likely a good choice for node-centric visualization. In this way we give users a general sense of the graph size, some hints as to its structure, and suitable places to start a more detailed visualization.

4 Implementation

The visualizer is implemented as an HTTP request handler. It responds to GET requests using a style influenced by the RDFNet proposed standard [8, 11]; generating results in a combination of HTML and Scalable Vector Graphics(SVG).

4.1 Request protocol

All requests are of the form

`GET http://`*hostname* `/visualize&lang=` ... `HTTP 1.1`

Where `lang` is a required parameter:

- `lang=` [`svg` | `visual`]
  use the `svg` option to receive a Scalable Vector Graphics (SVG) file or the `visual` option to receive that same svg file, but embedded in an HTML form with user interface controls.

and there are additional optional parameters:

- `model=`*modelName*
  specifies the name of the primary model from which results should be displayed. In the future, more than one model may be permitted, hence this is a parameter and not part of the pathname.

- `r=`*resourceURI*
  specifies a particular resource to display. This is the node for which a node-centric display (showing one arc forward and two arcs back) will be generated. This is an optional parameter. If missing, then the system will return all nodes which match the specified search text.

- `search=`*searchText*
  specifies text to search for in all literals. In the present implementation this is a single word, optionally ending in `*` to indicate a search for all literals containing a word that starts with the search text. A search text of `*` matches all literals. If no search text is specified and no particular resource is selected then a view of the entire model is returned.

- `style=` [`list` | `arcs`]
  specifies the style for returned search results. The default is a `list` of nodes, however for the case where the user selects an arc, we handle that specially, returning both a list of instances of that arc along with definitions of the arc predicate.

The visualizer is implemented in Java and uses HP's Jena RDF library [7] for reading and parsing input files. Source code is available under an open source license at: http://www.hpl.hp.com/personal/Craig_Sayers/rdf/visual.

4.2 Preprocessing

In early implementations the rendering engine operated in a stateless manner; starting from scratch for each display. This was convenient for implementation, but suffered from poor performance. Rendering a node-centric view from within a 20 Mbyte dataset could take more than 30 seconds - a quite unacceptable time.

Thus, more recent implementations incorporate an off-line analysis phase. The system is given a directory of RDF files at startup and examines them to generate a set of cross-referenced indexes. For each node, it stores sorted lists of all the forward and backward arcs from/to that node along with all nodes reachable by following those arcs. To aid in searches, it also generates indexes of all words used in literals. For each word, it stores the literal and a sorted list of nodes which are reachable by following a link backward from each literal.

On a standard 2003-era personal computer it takes around 80 seconds to perform the analysis on a 20 Mbyte RDF file containing 57,000 nodes that was written using the xml/rdf syntax (about 1/3 of that time is simply reading and parsing the file). Thereafter, responding to search and visualization requests takes less than 1 second.

5 Discussion

5.1 Additional Information about resources

Recently there has been considerable discussion in the RDF community regarding mechanisms for finding more information about a resource (see for example [15]). This has implications for our visualization system.

One proposed approach is to use a variant of the HTTP GET protocol; where GET would obtain a resource itself, while MGET would obtain a description of the resource [14].

Another proposed approach is to vary the name of the resource, so for example one might use HTTP GET resourceURI to get the resource, or HTTP GET resourceURI.rdf to get metadata about the resource [15].

Unfortunately, those and related techniques provide no way to know if additional information for a resource exists unless you look for it[2].

For a visualization, that is problematic. We could create a link from every resource to potential additional information, however, the probability that clicking on the link will work is very much less than for a link in any other web page. We expect that after a user clicks on several bad links they will quickly learn to ignore them. Testing each link beforehand during the preprocessing phase would allow us to only show links which are known to work. This is appealing until the time required is considered. We already visualize files containing more than 57,000 nodes - trying to get each one is just not realistic.

A much-preferable approach (at least from a visualization perspective) is to explicitly add RDF statements to indicate the availability of additional information. For example using predicates `rdfs:seeAlso` or `rdfs:isDefinedBy`.

In each case we see such statements we have the option to generate an appropriate visualization with a link which we know is likely (or at least expected) to work.

5.2 Future work

While the off-line analysis speeds processing of large files it also requires a fixed and static information set. More dynamic and intelligent approaches will be necessary.

Other potential areas of improvement abound in the system. In particular, there are opportunities for enhancing the wider view to show more connectivity information and to improve search and display taking into account language or datatype, supporting multiple constraints, and making use of namespace and ontological information.

We note that the visualization is the result of experimentation driven by intuition and aesthetic sensibility. We have not conducted any user studies.

[2]There is also no guarantee that the result of attempting to look will return a definitive answer (a server may be down or temporarily-overloaded for example)

6 Conclusions

A node-centric approach to RDF graph visualization has been described. Rather than trying to show a whole graph, we instead use a search over literal text to choose potential starting nodes and then allow the user to browse using an interactive display which shows the area one arc back and two arcs forward from a selected node. In cases where a wider view is desired we sort and display nodes based on the number of incoming and outgoing arcs.

The visualization features a consistent left-to-right orientation for node connectivity and all displayed nodes and arcs are live links which may be selected to obtain additional information.

Providing an acceptable interactive response time required introducing a preprocessing step and trading space for speed by creating large cross-referenced indexes.

We recommend using explicit RDF statements to indicate the location of additional information about a resource.

7 Acknowledgements

Thanks to all those who have contributed to HP Labs Jena RDF library, and to Mark Butler for providing a test dataset. We are especially grateful to Steve Cayzer for suggesting a number of improvements to both the visualizer and this document.

References

[1] AT&T. Graphviz. http://www.research.att.com/sw/tools/graphviz/, 2004.

[2] D. Brinkley. Rudolf: RDFViz. http://www.ilrt.bris.ac.uk/discovery/rdf-dev/rudolf/rdfviz/, 2004.

[3] J. Grant and D. Beckett. Resource description framework (RDF) test cases, W3C working draft. http://w3.org/TR/2002/WD-rdf-testcases-20020429, April 2002.

[4] Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, /2000.

[5] G. Klyne and J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. http://www.w3.org/TR/rdf-concepts, November 2002.

[6] O. Lassila and R. Swick. Resource description framework (RDF) model and syntax specification, W3C recommendation. http://w3.org/TR/1999/RED-rdf-syntax-19990222, February 1999.

[7] B. McBride et al. The Jena semantic web toolkit. http://www.hpl.hp.com/semweb/, 2004.

[8] G. Moore and A. Seaborne. The RDF net API, W3C member submission. http://www.w3.org/Submission/2003/SUBM-rdf-netapi-20031002/, October 2003.

[9] E. Pietriga. IsaViz. http://www.w3.org/2001/11/IsaViz/, 2003.

[10] The Protégé ontology editor and knowledge acquisition system. http://protege.stanford.edu/plugins.html, 2004.

[11] A. Seaborne. Joseki: The Jena RDF server. http://www.joseki.org/, 2003.

[12] D. Skvortsov. RDF model browser. http://visualrdf.sourceforge.net/.

[13] D. Steer. Brownsauce: an introduction. Technical Report HPL-2003-10, Hewlett-Packard Laboratories, Bristol, England, January 2003.

[14] P. Stickler. The uri query agent model: A semantic web enabler. http://sw.nokia.com/uriqa/URIQA.html, 2004.

[15] D.-W. van Gulik. Summary various 'about:' solutions. http://lists.w3.org/Archives/Public/www-rdf-interest/2004Mar/0011.html, March 2004.

[16] K.-P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *ACM CHI 2003*, April 2003.