Chapter 6

Data Visualization in Hera

A common and natural representation for RDF data is a directed labeled graph. Although there are tools to edit and/or browse RDF graph representations, we found their architecture rigid and not easily amenable to producing effective visual representations, especially for large RDF graphs. We discuss here how GViz, a general purpose graph visualisation tool, allows the easy construction and fine-tuning of various visual exploratory scenarios for RDF data. GViz's extended ability of customizing the visualization's icons showed to be very useful in the context of RDF graph structures visualisation. We demonstrate our approach by applying the developed visualization techniques for the RDF data models used in the Hera methodology. Based on the proposed visualization techniques one can answer complex questions about this data and have an effective insight into its structure.

6.1 Introduction

RDF is intended to describe the Web metadata so that the Web content is not only machine readable but also machine understandable. In this way one can better support the interoperability of Web applications. RDF Schema (RDFS) is used to describe different RDF vocabularies (schemas), i.e., the classes and properties of a particular application domain. An instantiation of these classes and properties form an RDF instance. It is important to note that both an RDF schema and an RDF instance have RDF graph representations.

Realizing the advantages that RDF offers, in the last couple of years, many tools were built in order to support the browsing and editing of RDF data. Among these tools we mention Protégé [Noy et al., 2001], OntoEdit [Sure et al., 2003], and RDF Instance Creator (RIC) [Grove, 2002]. Most of the text-based environments are unable to cope with large amounts of data in the sense of presenting them in a way that is easy to understand and navigate [Card et al., 1999]. The RDF data we have to deal with describes a large number of Web resources, and can thus easily reach tens of thousands of instances and attributes. We advocate the use of visual tools for browsing RDF data, as visual presentation and nav-

igation enables users to effectively understand the complex structure and interrelationships of such data. Existing visualization tools for RDF data are: IsaViz [Pietriga, 2002], OntoRAMA [Eklund et al., 2002], and the Protégé visualization plugins like OntoViz [Sintek, 2004] and Jambalaya [Storey et al., 2002].

The most popular textual RDF browser/editor is Protégé [Noy et al., 2001]. The generic modeling primitives of Protégé enable the export of the built model in different data formats among which is also RDF/XML. Protégé distinguishes between schema and instance information, allowing for an incremental view of the instances based on the selected schema elements. One of the disadvantages of Protégé is that it displays the information in a hierarchical way, i.e., using a tree layout [Sugiyama et al., 1981], which makes it difficult to grasp the inherent graph structure of RDF data.

In this chapter, we advocate the use of a highly customizable, interactive visualization system for the understanding of different RDF data structures. We implemented an RDF data format plugin for GViz [Telea et al., 2002], a general purpose visual environment for browsing and editing graph data. The largest advantage that GViz provides in comparison with other RDF visualization tools is the fact that it is easily and fully customizable. GViz was architected with the specific goal in mind of allowing users to define new operations for data processing, visualization, and interaction to support application specific scenarios. GViz also integrates a number of standard operations for manipulation and visualization of relational data, such as data viewers, graph layout tools, and data format support. This combination of features has enabled us to produce, in a short time, customized visualization scenarios for answering several questions about RDF data. We demonstrate our approach to RDF data visualization by using a real dataset example of considerable size.

In the next section, we describe the real-world dataset we use, and show the results obtained when visualizing it with several existing RDF tools. Our visualization tool, GViz, is presented in Section 6.3. Section 6.4 presents several visualization scenarios we built with GViz for the used RDF dataset, and details various lessons learned when building and using such visualizations. Finally, Section 6.5 concludes the chapter proposing future directions for visualizing RDF information.

6.2 Related Work

Throughout this chapter, we will use an example based on real data made available by the Rijksmuseum in Amsterdam, the largest art and history museum in the Netherlands. In the example there is a museum schema used to classify different artists and their artifacts. The museum instance describes more than 1000 artists and artifacts.

Figure 6.1 depicts the museum schema in Protégé. As can be noticed from this figure such a text-based representation cannot nicely depict the structure of a large amount of data. More exactly, a text-based display is very effective for data mining, i.e., posing targeted queries on a dataset once one knows what structure one is looking for. However, text-based displays are not effective for data understanding, i.e., making sense of a given (large) dataset of which the global structure is unknown to the user.

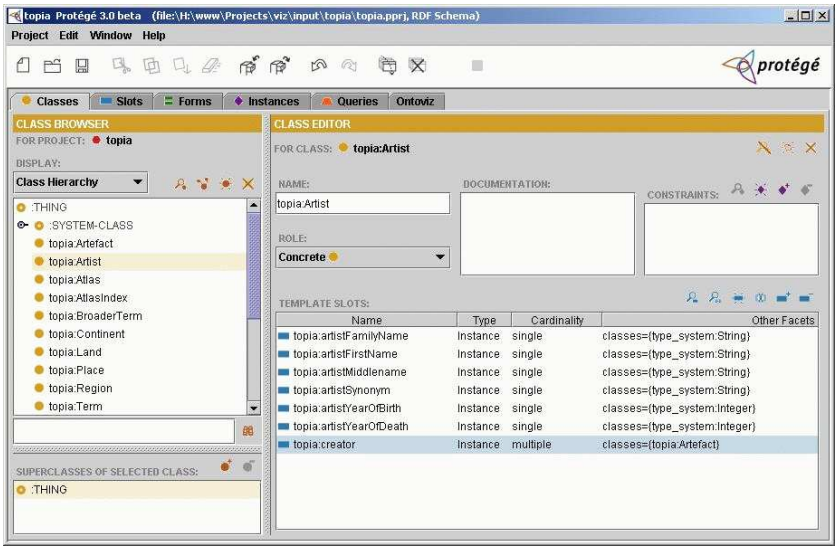

Figure 6.1: Museum schema in Protégé (text-based).

In order to alleviate the above limitation, Protégé offers a number of built-in visualization plugins. Figure 6.2 shows the graph representation generated by the OntoViz plugin for two classes from the museum schema. The weak point of OntoViz is the fact that it is not able to produce good layouts for graphs that have more than 10 nodes.

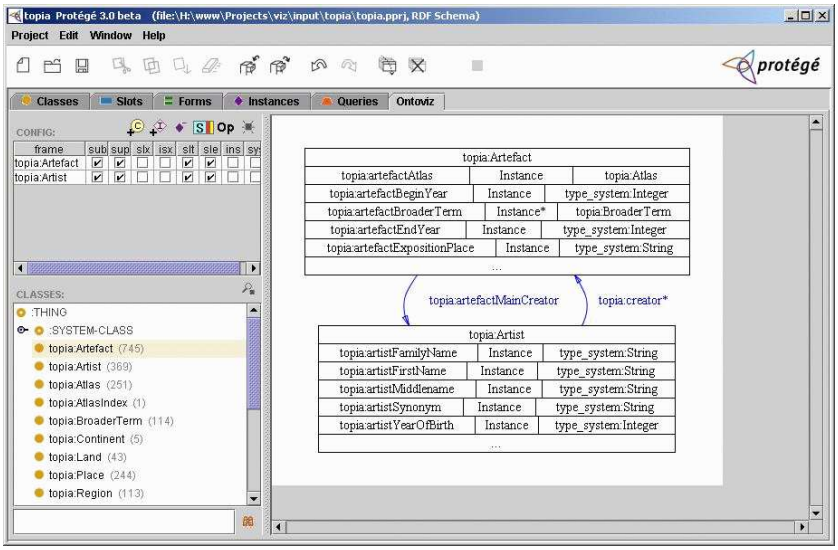

Figure 6.2: Museum schema in Protégé (with OntoViz plugin).

IsaViz [Pietriga, 2002] is a visual tool for browsing/editing RDF models. IsaViz uses AT&T's GraphViz package [Gansner et al., 2002; North, 2002] for the graph layout. Figure 6.3 shows the same museum schema using IsaViz. The layout produced by the tool is much better than the one generated with OntoViz. However, the directed acyclic graph

layout used [Sugiyama et al., 1981] becomes ineffective when the dataset at hand has roughly more than hundred nodes, as can be seen from Figure 6.3. IsaViz has a 2.5D GUI with zooming capabilities and provides numerous operations like text-based search, copy-and-paste, editing of the geometry of nodes and arcs, and graph navigation.

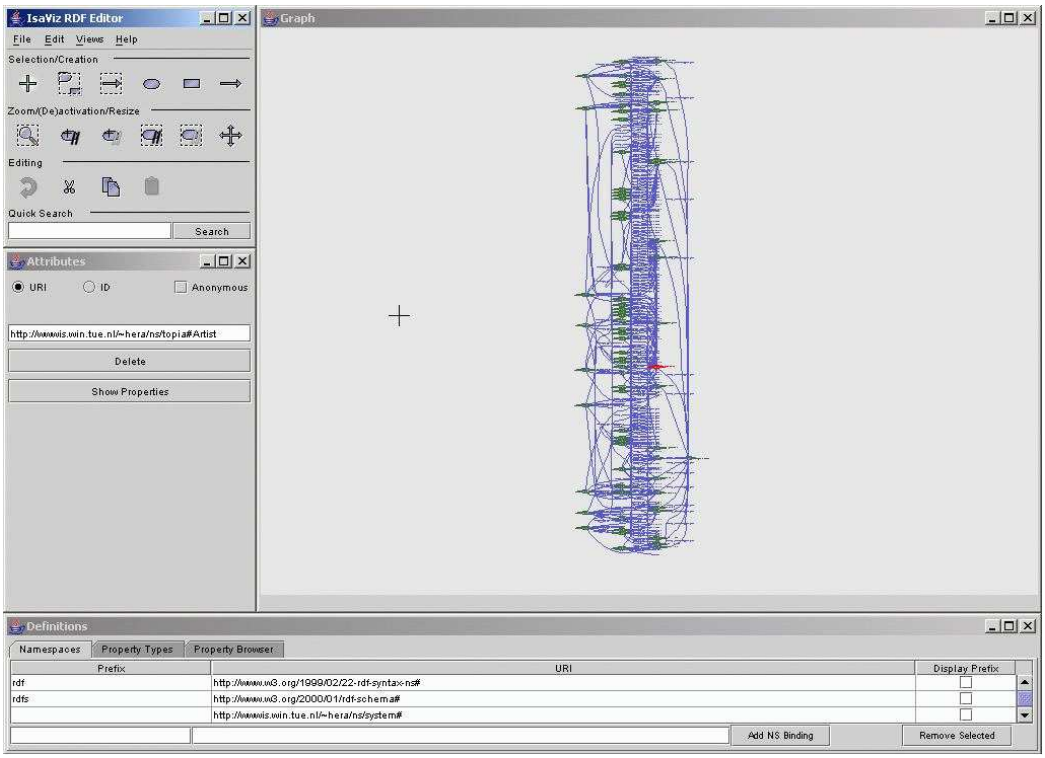

Figure 6.3: Museum schema in IsaViz.

For all these reasons, we believe that IsaViz is a state of the art tool for browsing/editing RDF models. However, its rigid architecture makes it difficult to define application-dependent operations others than the standard ones currently provided by the tool. Experience in several communities interested in visualizing relational data in general, such as software engineering and web engineering, and our own experience with RDF data in particular, has shown that tool customization is extremely important. Indeed, there is no "silver bullet" or best way to visualize large graph-like datasets. The questions to be answered, the data structure and size, and the user preferences all determine the "visualization scenario", i.e., the kind of (interactive) operations the users may want to perform to get insight in the data and answers to their questions. It is not that each separate application domain demands a specific visualization scenario. Users of the same domain and/or even the same dataset within the same domain may require different scenarios. Building such scenarios often is responsible for a large part of the complete time spent in understanding a given dataset [Telea, 2004]. This clearly requires the visualization tool in use to be highly (and easily) customizable.

6.3 GViz

In our attempt to understand RDF data through visual representations, an existing tool was used. We implemented an RDF data format plugin for GViz [Telea et al., 2002], a general purpose visual environment for browsing and editing graph data. The largest advantage that GViz provides in comparison with other RDF visualization tools is the fact that it is easily and quickly customizable. One can seamlessly define new operations to support application specific scenarios, making thus the tool more amenable for the user needs. In the past, GViz was successfully used in the reverse engineering domain, in order to define application specific visualization scenarios. Figure 6.4 presents the architecture of GViz based on four components: selection, mapping, editing, and visualization. In the next section we describe the data model used in GViz. Next, we outline the operation model describing the tasks that can be defined on the graph data. We finish the description of the GViz architecture with the visualization component which we illustrate using the museum schema dataset. The GViz core implementation is done in C++ while the user interface and scripting layer were implemented in Tcl [Raines, 1998] to take advantage of the run-time scripting and weak typing flexibility that this language provides. All the GViz customization code developed for the RDF visualization scenarios presented in this chapter was done in Tcl.

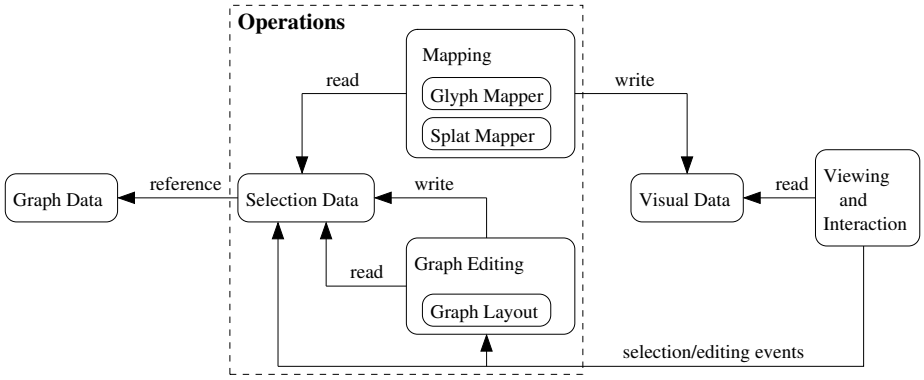

Figure 6.4: GViz architecture.

6.3.1 Data Model

The data model of GViz consists of three elements:

- *graph data*: this is the RDF graph model, i.e., a labeled directed multi-graph in which no edges between the same two nodes are allowed to share the same label. Nodes stand for RDF resources/literals and edges denote properties. Each node has a type attribute which states if the node is a *NResource* (named resource), an *AResource* (anonymous resource), or a *Literal*. The label associated to a node/edge is given by the value attribute. The labels for NResource nodes and edges are URIs. The

label for Literals is their associated string. The value of an AResource is an internal identifier with no RDF semantics. Note that the type and value attributes are GViz specific attributes that should not be confused with their RDF counterparts. Since GViz's standard data model is an arbitrary attributed graph, with any number of (name, value) type of attributes per node and edge, the RDF data model is directly accommodated by the tool.

- *selection data*: selections are subsets of nodes and edges in the graph data. Selections are used in GViz to specify the inputs and outputs of its operations; their use is detailed in Section 6.3.2.
- *visual data*: this is the information that GViz ultimately displays and allows the user to interact with. Since GViz allows customizing the mapping operation, i.e., the way graph data is used to produce visual data, the latter may assume different look-and-feel appearances. Section 6.4 illustrates this in the context of our application.

6.3.2 Operation Model

As shown in Figure 6.4, the operation model of GViz has three operation types: *selection*, *graph editing*, and *mapping*. Selection operations allow users to specify subsets of interest from the whole input graph. In the RDF visualization scenarios that we built with GViz, we defined different complex selections based on the attributes of the input model. These selections can perform tasks like: "extract the schema from an input set of RDF(S) data (which mixes schema and instance elements)". Custom selections are almost always needed when visualizing relational data, since a) the user doesn't usually want to look at too many data elements at the same time, and b) different subsets of the input data may have different semantics, thus have to be visualized in different ways. A basic example of the latter assertion is the schema extraction selection mentioned above.

Graph editing operations enable the modification, creation, and deletion of nodes/edges and/or their attributes. For our RDF visualization scenarios, we did not create or delete nodes or edges. However, we did create new data attributes, as follows. One of the key features of GViz is that it separates the graph layout, i.e., computing 2D or 3D geometrical positions that specify where to draw nodes and edges, from the graph mapping, i.e., specifying how to draw nodes and edges. The graph layout is defined as a graph editing operation which computes position attributes. Among the different layouts that GViz supports we mention the spring embedder, the directed (tree), the 3D stacked layout, and the nested layout [Telea et al., 2002]. Although based on the same GraphViz package as IsaViz, the layouts of GViz are relatively more effective, as the user can customize their behavior in detail via several parameters.

Mapping operations, or briefly mappers, associate nodes/edges (containing also their layout information) to visual data. The latter is implemented using the Open Inventor 3D toolkit, which delivers high quality, efficient rendering and interaction with large 2D and 3D geometric datasets [Wernecke, 1993]. GViz implements two mappers: the glyph mapper and the splat mapper.

The glyph mapper associates to every node/edge in the input selection a graphical icon (the glyph) and positions the glyphs based on the corresponding node/edge layout attributes. Essentially, the glyph mapper produces the "classical" kind of graph drawings, e.g., similar to those output by IsaViz. However, in contrast to many graph visualization tools, the glyph mapper in GViz allows full customization of the way the nodes and edges are drawn. The user can specify, for example, shapes, sizes, and colors for every separate node and edge, if desired, by writing a small Tcl script of 10 to 20 lines of code on the average. We used this feature extensively to produce our RDF visualizations described in Section 6.4. The splat mapper produces a continuous two-dimensional splat field for the input selection. For every 2D point, the field value is proportional to the density of nodes per unit area at that point. Essentially, the splat mapper shows high values where the graph layout used has placed many nodes, and low values where there are few nodes. Given that a reasonably good layout will cluster highly interconnected nodes together, the splat mapper offers a quick and easy way to visually find the clusters in the input graph (Figure 6.9, Section 6.4). For more details on this layout, see [van Liere and de Leeuw, 2003].

A final way to customize the visualizations in GViz is to associate custom interaction to the mappers. These are provided in the form of Tcl callback scripts that are called by the tool whenever the user interactively selects some node or edge glyph with the mouse, in the respective mapper windows. These scripts can initiate any desired operation using the selected elements as arguments, for example showing some attributes of the selected arguments. Examples of this mechanism are discussed in Section 6.4.

As explained above, GViz allows users to easily define new operations. For the incremental view of RDF(S) data, we defined operations as: extract schema, select classes and their corresponding instances, select instances and their attributes. As for the glyph mappers, these operations have been implemented as Tcl scripts of 10 to 25 lines of code. The usage of the custom selection, layout, and mapping operations for visualizing RDF(S) data is detailed in the remainder of this chapter.

6.3.3 Visualization

Figure 6.5 presents the museum data schema in GViz. We use here a radial tree layout, also available in the GraphViz package, instead of the directed tree layout illustrated in Figure 6.3 for IsaViz. As a consequence, the structure of the schema is easier to understand now.

In the above picture the edges with the label *rdf*:*type* are depicted in blue. There are two red nodes to which these blue edges connect, one with the label *rdfs*:*Class* and the other with the label *rdf*:*Property*, shown near the nodes as balloon pop-up texts. We chose to depict the property nodes (laid out in a large circular arc around the upper-left red node) in orange and the class nodes (laid out in a smaller circle arc around the lower-right red node) in green. As it can be noticed from the picture there are a lot of orange nodes which is in accordance with the property-centric approach for defining RDFS schemas. In order to express richer domain models we extended the RDFS primitives with the cardinality

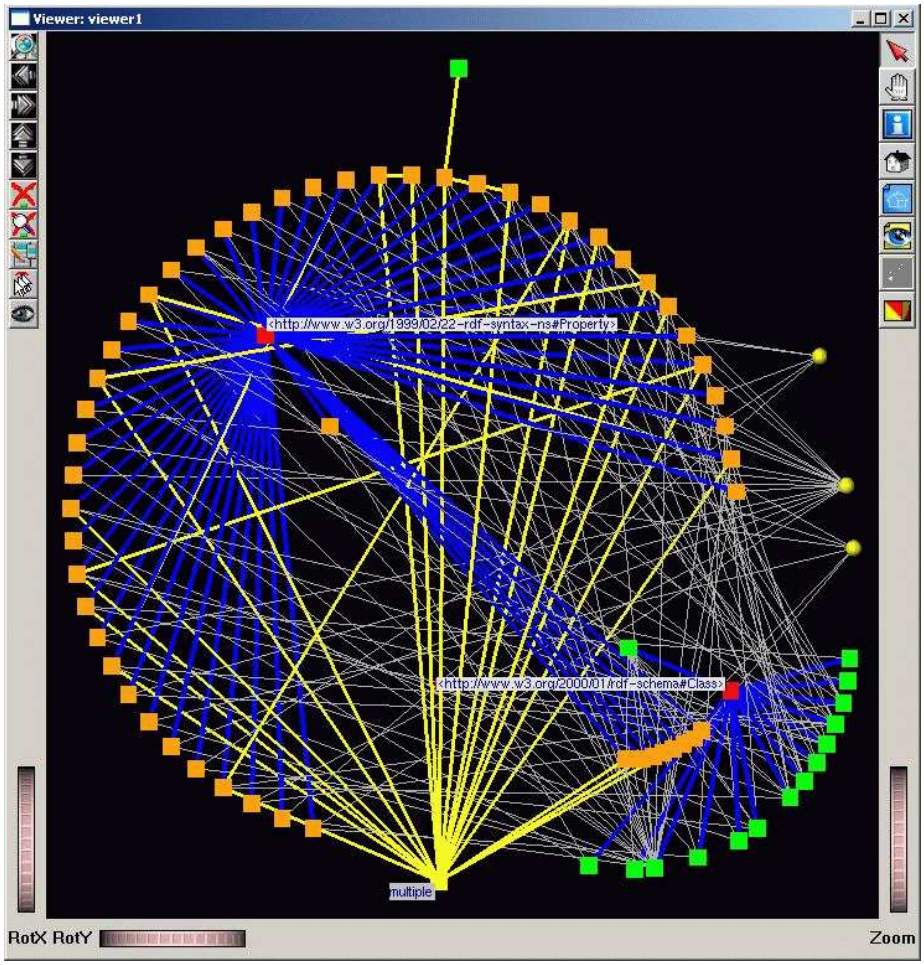

Figure 6.5: Museum schema in GViz (2D).

of properties, the inverse of properties, and a media type system. These extensions are showed in yellow edges and yellow spheres (positioned at the right end of the image). The yellow edges that connect to orange nodes represent the inverse of a property. The yellow edges that connect an orange node with the yellow rectangle labeled *multiple* (positioned at the middle of the figure bottom) state that this property has cardinality one-to-many. The default cardinality is one-to-one. Note that there are not many-to-many properties as we had previously decomposed these properties in two one-to-many properties. The three yellow spheres represent the media types: *String*, *Integer*, and *Image*. The light gray thin edges denote the domain and the range of properties. Note that only range edges can have a media node at one of its ends. As these edges are a) not so important for the user and b) quite numerous and quite hard to lay out without many overlaps, we chose to represent them in a visually inconspicuous way, i.e., make them thin and using a background-like, light gray color.

The tailoring of the graph visualization presented above is only one example. One can

define some other visualizations depending on ones needs. Figure 6.6 presents a 3D view of the same museum schema example. Here, we used a spring embedder layout, also available from the GraphViz package, to position all schema nodes in a 2D plane. Next, we designed a custom operation that selects the two $rdfs{:}Class$ and $rdf{:}Property$ nodes and offsets them away from the 2D layout plane, in opposite directions. This creates a 3D layout, which allows the user to better distinguish the different kinds of edges. For example, the edges labeled $rdf{:}type$ (colored in blue) are now clearly separated, as they reach out of the 2D plane to the offset nodes.

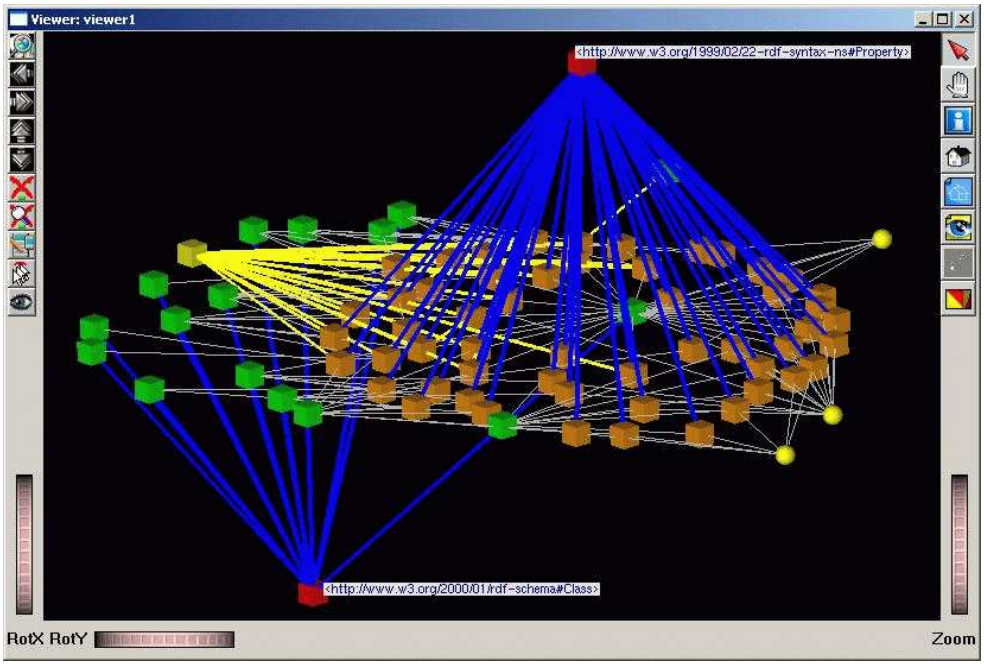

Figure 6.6: Museum schema in GViz (3D).

6.4 Applications

In order to better understand the context in which we developed our visualization applications we now briefly describe the Hera project [Vdovjak et al., 2003]. Hera is a methodology for designing and developing Web Information Systems (WISs) on the Semantic Web. All the system specifications are represented in RDFS. For the scope of this chapter it is important to have a look at two of these specifications: the conceptual model (domain model) and the application model (navigation model).

The conceptual model describes the types of the instances that need to be presented. An example of the conceptual model we already saw in Figure 6.5. A conceptual model is composed of concepts and concept properties. There are two kinds of concept properties: relationships (properties between concepts) and attributes (properties that refer to media types).

The application model defines the navigation over the data, i.e., a view on the conceptual model that contains appropriate navigation links. The application model is an overlay model over the conceptual model, a feature exploited in the definition of the transformations of the conceptual model instances into application model instances. An application model is composed of slices and slice properties. A slice contains concept attributes (not necessarily from the same concept) as well as other slices. There are two kinds of slice properties: compositional properties (aggregations) and navigational properties (links). The owner relationship is used to associate a slice to a concept. Each slice has a title attribute related to it.

A conceptual model instance and an application model instance are represented in RDF (which should be valid according to the corresponding RDFS specifications, i.e., the conceptual model and the application model, respectively). In the WIS application it is only the application model instance that will be visible to the user.

We consider now four types of RDF(S)-related visualization scenarios that are relevant in the support of the WIS application designer:

- conceptual model visualization
- conceptual model instance visualization
- application model visualization
- application model instance visualization

In Section 6.3.3 we already showed how one can visualize conceptual models. A second similar scenario for the conceptual model visualization is described next.

6.4.1 Conceptual Model Visualization

The conceptual model visualization enables one to better understand the structure of the application's domain. It answers questions like: what are the concepts?, what are the properties?, what are the relationships between concepts and properties? what are the most referenced concepts?, what are the most referenced media types?, etc.

Figure 6.7 shows the extracted schema from an RDF file that contains both the schema and its associated instance. The extraction is done by a custom selection operation, as described in Section 6.3.2. The picture is very similar to the one from Figure 6.5. However, there are two differences between this picture and the one from Figure 6.5. First, we now use a different layout, i.e., a spring embedder instead of a radial tree. Secondly, we now depict also the direction of the edges. The edges are fading out towards the start node. A direction effect is created: the edges get brighter as they approach the end node. We found this representation of the edge direction much more effective than the arrow representation when visualizing large graphs, as the drawing of arrows produces too much visual clutter in this case. Moreover, the edge fading glyph is faster to render than an arrow glyph, as it involves a single (shaded) line primitive.

From Figure 6.7 we can deduce that the most used media type is *String* (the text-based descriptions are the most popular for this domain specification) and the most referenced concept is the *Artifact* (it has the most relationships). Each artifact is classified by some museum terms (e.g., *Self Portraits*). There is a hierarchy of museum terms, terms are grouped in broader terms (e.g., *Portraits*), and broader terms are grouped in top terms (e.g., *Paintings*).

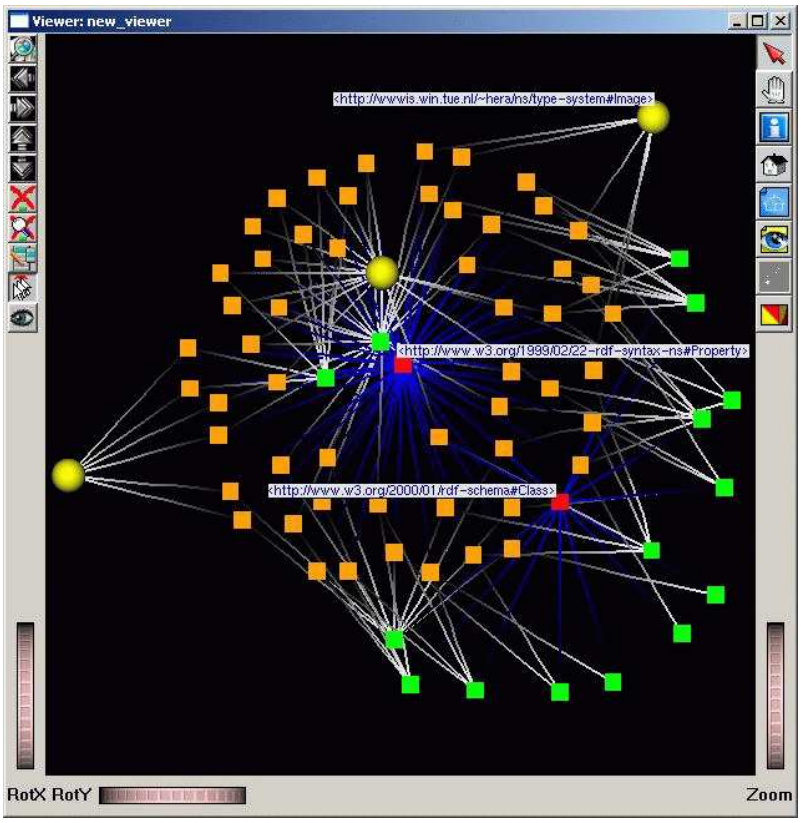

Figure 6.7: Museum extracted conceptual model.

6.4.2 Conceptual Model Instance Visualization

The conceptual model instance visualization answers questions like: what are the instances of a certain concept?, what are the relations between two selected concept instances?, what are the most referenced instances?, what are the attributes of a selected instance?, etc.

In most of the encountered situations, there are (much) more concept instances than concepts. For example, our museum dataset contains tens of thousands of instances. It is easy to imagine other applications where this number goes up to hundreds of thousands, or even more. Drawing all these instances simultaneously is neither efficient nor effective. Indeed, no graph layout we were able to test could produce an understandable image of an arbitrary, relatively tightly connected graph with tens of thousands of nodes in a

reasonable amount of time (e.g., tens of seconds). In order to keep the instance visualization manageable, we decided for an incremental view scenario on the RDF(S) data. First, the user selects the subpart of the schema for which he wants the corresponding instances to be visualized. Next, we use a custom interaction script (Section 6.3.2) of about ten lines of code to separately visualize the instances of the selected items. For example, when the user selects the Artist and Artifact concepts from Figure 6.7, the GViz tool automatically shows the instances of these concepts and their relations in another window, using a spring embedder layout (Figure 6.8). In Figure 6.8 we used a custom glyph mapper to depict the artifacts with blue rectangles and the artists with green rectangles. The relations between these instances are represented by fading white edges. One can note that there are more artifacts than artists, as expected.

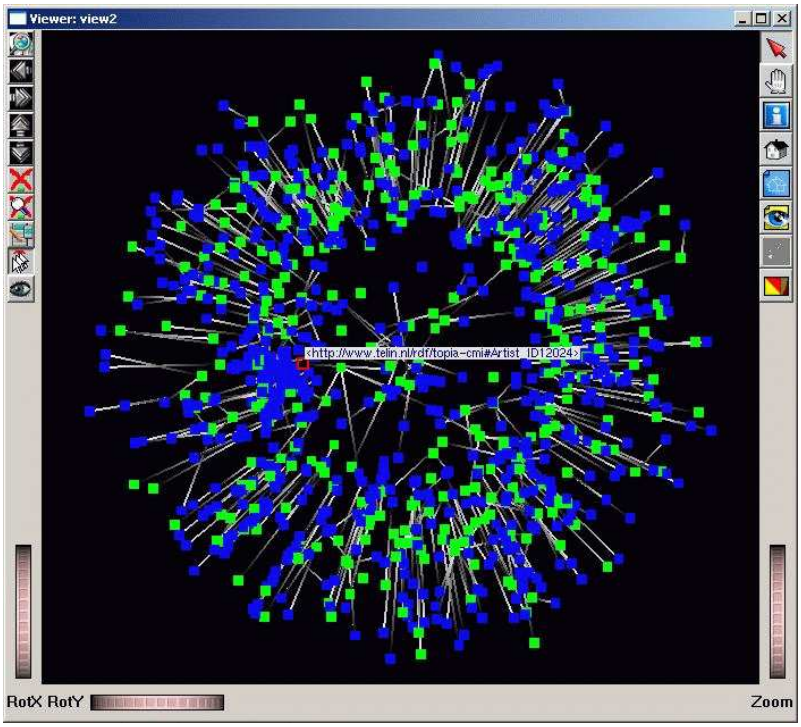

Figure 6.8: Artists/artifacts properties in the conceptual model instance.

Figure 6.9 shows the same selected data (artists and artifacts) but using a splat mapper instead of the classical glyph mapper. The scalar density function (splat field) is constructed as outlined in Section 6.3.2. We visualize the splat field using a blue-to-red colormap that associates cold hues (blue) to low values and warm hues (yellow, red) to medium and high values. Figure 6.9 (left) shows the splat field as seen from above. Figure 6.9 (right) shows the same splat field, this time visualized using an elevation plot that shows high density areas also by offsetting these points in the Z (vertical) direction. A red/yellow color in Figure 6.9 (left and right) or a high elevation point in Figure 6.9 (right) indicate that there are a lot of relations for a particular instance or group of instances. In this way one can notice from Figure 6.9 which are the artists with the most artifacts. The

artists with the most artifacts are the unknown artists (potter, goldsmith, bronzesmith, etc.) that show up as the singular peak in the left of Figure 6.9 (left). On the average, these artists have several tens (up to 60) artifacts. They are followed by Rembrandt and the unknown painters, who show up as the other two higher peaks to the right of Figure 6.9 (left). This can be explained by the fact that in the 17th century, for which the Rijksmuseum has a special focus, there were a lot of artifacts done by unknown artists.

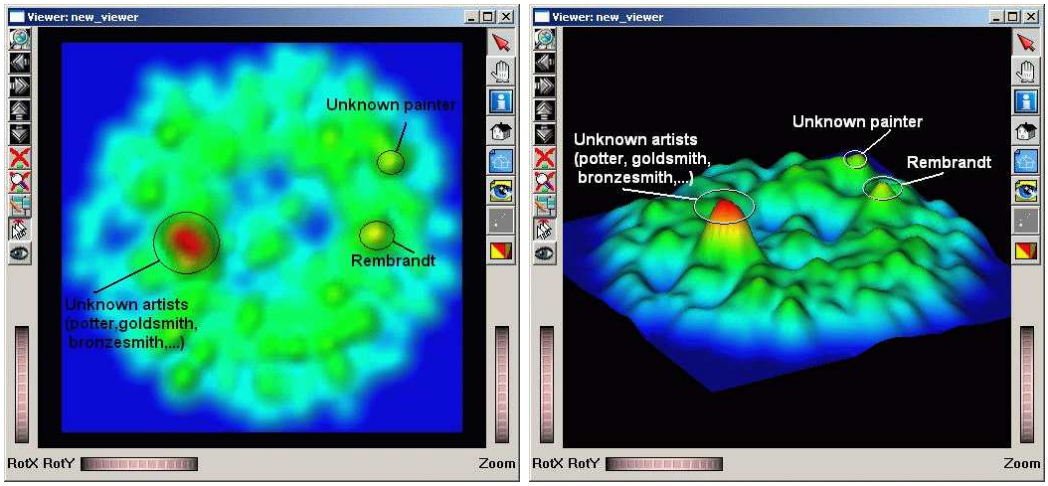

Figure 6.9: Conceptual model instance splatting (left: 2D; right: elevation plot).

We have further customized our visualization scenario, as follows. When the user selects one instance of Figure 6.8, we use a custom interaction script on the mapper of Figure 6.8 to pop up another window to display the instance attributes. The selected instance is shown as having the balloon pop-up label in Figure 6.8. Figure 6.10 shows the attributes of the selected instance, in this case Rembrandt.

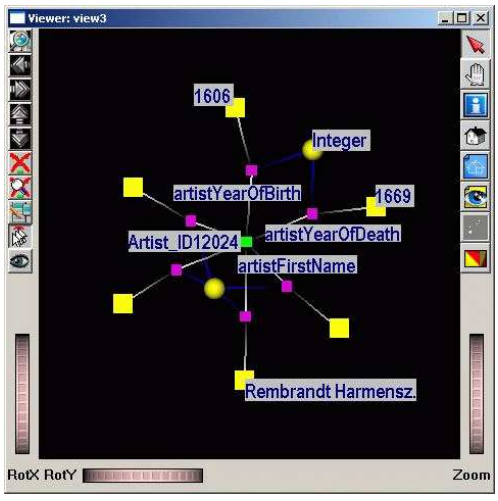

Figure 6.10: Attributes of a selected concept instance.

6.4.3 Application Model Visualisation

The application model visualization enables one to better understand the navigation structure of a hypermedia presentation. It answers questions like: what are the application model slices?, what are their links?, what are the slice owners?, what are the slice titles?, what slices are navigation hubs?, what are possible navigation paths from a certain slice?, etc.

Figure 6.11 depicts the application model for the museum example. We chose to present here the top-level slices (slices that correspond to web pages) and the links between them in order to decrease the complexity of the picture. A new glyph shape was designed in order to represent the pizza slice shape for slices (as defined in the application model's graphical representation language). The blue thick edges represent links between slices. Each slice has associated with it two attributes. We use a custom layout to place these nodes right above the top of the slice node. The slice nodes themselves are laid out using the spring embedder already discussed before. The two attributes of each slice are visualized by using two custom square glyphs, as follows: the yellow glyph (left) stands for the name of the slice and the green glyph (right) denotes the concept owner of the slice (remember that the concept owner is a concept from the conceptual model). In the center of the picture is the *Slice.artefact.main* slice which has the most links associated with it, i.e., it is a navigation hub. The figure also shows the designer's choice to present the museum information based of the terms hierarchy: top terms, broader terms, and terms.

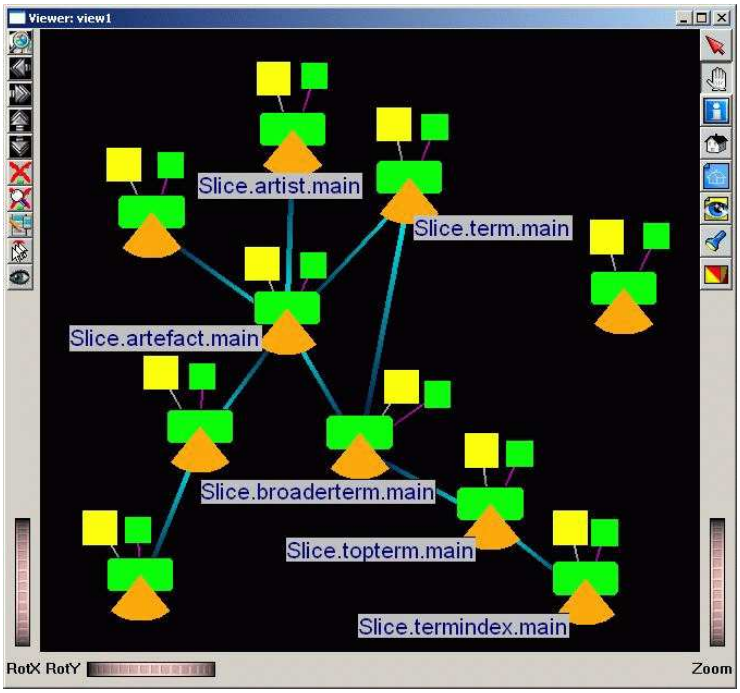

Figure 6.11: Museum application model.

6.4.4 Application Model Instance Visualisation

The application model instance visualization answers questions like: what are the instances of a certain slice?, what are the slice instances reachable from a certain slice instance?, what are the most referenced slice instances?, what are the attributes of a selected slice instance?, etc.

As there are more slice instances than slices, in order to keep the visualization manageable we used the same visualization scenario as for conceptual model instances, i.e., to use incremental views. The user can select from the mapper in Figure 6.11 the slices for which he wants the corresponding instances to be visualized. For example, after selecting the *Slice.topterm.main*, *Slice.broaderterm.main*, and *Slice.term.main* slices from Figure 6.11, we use the same mechanism of a custom interaction script (Section 6.3.2) to pop up another window that shows the instances of these slices and their associated links. Figures 6.12 and 6.13 show the corresponding slice instances, as described below.

For the visualizations in Figures 6.12 and 6.13, we use yellow sphere glyphs for nodes labeled *Slice.topterm.main*, green sphere glyphs for nodes labeled *Slice.broaderterm.main*, and blue rectangle glyphs for nodes labeled *Slice.term.main*. The chosen colors and shapes are motivated by the need to produce an expressive, easy to understand picture when presenting a large number of instances coming from three slices linked in a hierarchical way, as follows. We did give up the pizza slice glyph for these visualizations as we found out that this glyph produces too much visual clutter for large graphs. Next, we chose colors of increasing brightness (blue, green, and yellow) to display items of increasing importance (terms, broader terms, and top terms, respectively). The size of the glyphs used for these items also reflects their importance (the top term glyphs are the largest, whereas the term glyphs are the smallest). A final significant cue is the shape: the more important top and broader terms are drawn as 3D shaded spheres, whereas the less important terms are drawn as 2D flat squares. For the edges connecting these glyphs in the visualization, we used a varying color and size scheme that varies both line color and line thickness along the edge between the end nodes' colors and sizes respectively. Summing up, the combination of above choices produces a visualization where the overall structure of top terms and broader terms "pops" into the foreground, whereas the less important terms and their links "fade" into the background. As a comparison, we were unable to get the same clear view of the structure by just varying the layout parameters and using the same glyph for all nodes.

After selecting the slice instance corresponding to the *Paintings* top term, we obtain in Figure 6.12 the broader term slice instances accessible after one step, showed in red. By this, we mean the terms that a user of the web site (whose design our dataset captures) can access after one navigation step. This translates to nodes which are directly connected (via an edge) to the selected slice instance in our RDF dataset.

In Figure 6.13 we visualize the term slice instances accessible from the same Paintings top term instance slice after two steps, also drawn in red. These correspond to web pages that the user of the web site can access after two navigation steps. An example for the second step is the navigation from the broader term *Portraits*.

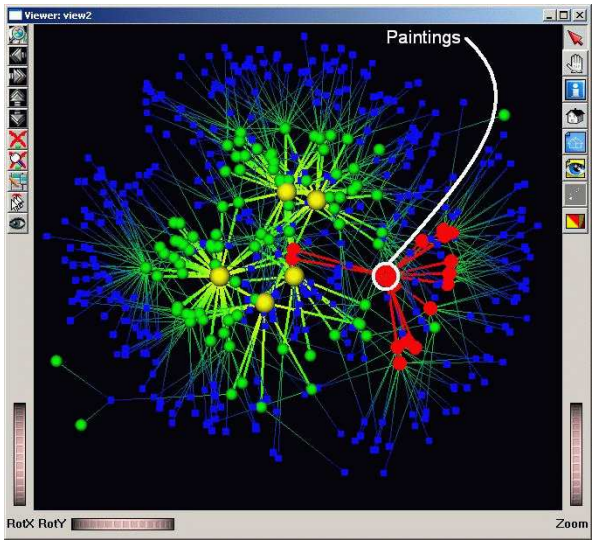

Figure 6.12: Broader term slice instances accessible from the Paintings slice instance.

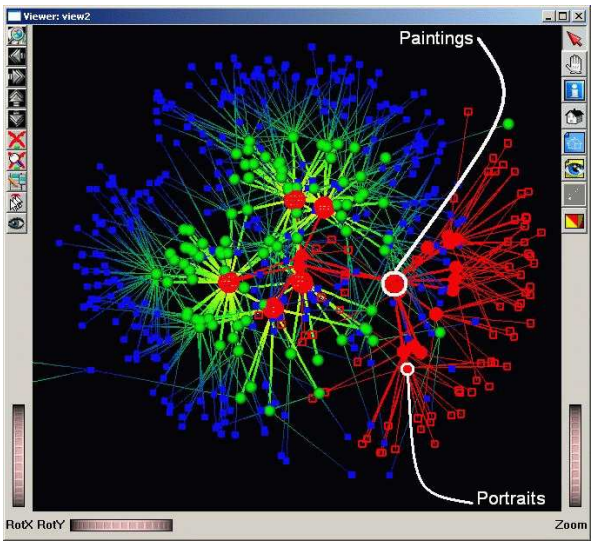

Figure 6.13: Term slice instances accessible from the Paintings slice instance.

6.5 Conclusions

In this chapter we have shown how a general purpose graph visualization tool, GViz, can be used for the visualization of large RDF graphs produced from real-world data. All experiments were performed in the context of the Hera project, a project that investigates the designing and developing of Web Information Systems on the Semantic Web. The visualization of large amounts of RDF input data and RDF design specifications enabled us to answer complex questions about this data and to give an effective insight into its

structure.

Several ingredients were crucial for obtaining these results. First, the amount of customizability of the GViz tool (layouts, selections, node and edge drawing, choice between glyph and splat mappers, and custom user interaction) was absolutely necessary to produce the desired visualization scenarios. We found all these elements to be necessary to create the desired results. We have actually experimented with customizing just the layout but not the glyphs and/or the interaction. In all cases, the results were not flexible enough to give the users the desired look-and-feel that would make the scenario effective for answering the relevant questions. Secondly, the script-based customization mechanism of GViz allowed a user experienced with Tcl scripting to produce the scenarios described here (which were imagined by a different user, inexperienced with Tcl) in a matter of minutes. Thirdly, we found that using several visual cues (shape, color, size, and shading) together to enhance a single attribute, as for example described in Section 6.4.4, is much more effective than using a single cue. Finally, we mention that none of the investigated RDF visualization tools (Section 6.2) showed the high degree of customization of GViz needed for our scenarios.

In the future, we would like to explore the GViz 3D visualization capabilities for RDF data, possibly getting an even better insight into the data structure. Another research direction would be to use GViz in conjunction with a popular RDF query language (like RQL for example). Our purpose is here twofold: to use the RDF query language as a selection operation implementation for GViz when visualizing RDF data and to support the RDF query language with the visualization of the input and resulted set of RDF data. Finally, as it is planned in the Hera project to use OWL instead of RDF for the future input data/design specifications we would like also to conduct visualization experiments on the more semantically rich OWL data.

Bibliography

Card, S. K., Mackinlay, J. D., and Shneiderman, B. (1999). *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann.

Eklund, P. W., Roberts, N., and Green, S. (2002). Ontorama: Browsing rdf ontologies using a hyperbolic-style browser. In *1st International Symposium on Cyber Worlds (CW 2002)*, pages 405–411. IEEE Computer Society.

Gansner, E., Koutsofios, E., and North, S. (2002). Drawing graphs with dot. `http://www.graphviz.org/Documentation/dotguide.pdf`.

Grove, M. (2002). Rdf instance creator. `http://www.mindswap.org/~mhgrove/RIC/RIC.shtml`.

North, S. C. (2002). Drawing graphs with neato. `http://www.graphviz.org/Documentation/neatoguide.pdf`.

Noy, N. F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R. W., and Musen, M. A. (2001). Creating semantic web contents with protege-2000. *IEEE Intelligent Systems*, 16(2):60–71.

Pietriga, E. (2002). Isaviz: a visual environment for browsing and authoring rdf models. The Eleventh International World Wide Web Conference (WWW 2002), Developer's day.

Raines, P. (1998). *Tcl/Tk Pocket Reference*. O'Reilly & Associates.

Sintek, M. (2004). Ontoviz tab: Visualizing protégé ontologies. `http://protege.stanford.edu/plugins/ontoviz/ontoviz.html`.

Storey, M.-A. D., Noy, N. F., Musen, M. A., Best, C., Fergerson, R. W., and Ernst, N. (2002). Ontoedit: Multifaceted inferencing for ontology engineering. In *International Conference on Intelligent User Interfaces (IUI 2002)*, pages 239–239. ACM.

Sugiyama, K., Tagawa, S., and Toda, M. (1981). Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems*, 11(2):109–125.

Sure, Y., Angele, J., and Staab, S. (2003). Ontoedit: Multifaceted inferencing for ontology engineering. *Journal on Data Semantics*, 1:128–152.

Telea, A. (2004). *Managing Corporate Information Systems Evolution and Maintenance*, chapter 9: An Open Architecture for Visual Reverse Engineering, pages 211–227. Idea Group Inc.

Telea, A., Maccari, A., and Riva, C. (2002). An open toolkit for prototyping reverse engineering visualization. In *IEEE EG VisSym '02*, pages 241–250. Eurographics.

van Liere, R. and de Leeuw, W. (2003). Graphsplatting: Visualizing graphs as continuous fields. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):206–212.

Vdovjak, R., Frasincar, F., Houben, G. J., and Barna, P. (2003). Engineering semantic web information systems in hera. *Journal of Web Engineering*, 2(1-2):3–26.

Wernecke, J. (1993). *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor*. Addison-Wesley.