Visualization of Semantic Metadata and Ontologies

Paul Mutton[1] and Jennifer Golbeck[2]
[1]*University of Kent, Canterbury, UK*
[2]*University of Maryland, College Park, USA*
pjm2@kent.ac.uk, golbeck@cs.umd.edu

Abstract

Implicit information embedded in semantic web graphs, such as topography, clusters, and disconnected subgraphs is difficult to extract from text files. Visualizations of the graphs can reveal some of these features, but existing systems for visualizing metadata focus on aspects other than understanding the greater structure. In this paper, we present a tool for generating visualizations of ontologies and metadata by using a modified spring embedder to achieve an automatic layout. Through a case study using a mid-sized ontology, we show that interesting information about the data relationships can be extracted through our visualization of the physical graph structure.

1. Introduction

The Semantic Web is based on the idea of creating "machine understandable" data that can be used and exchanged. Using web languages, such as RDF, DAML+OIL, and OWL, it is possible to create semantically rich data models. These models are made up of triples (subject-predicate-object), where subjects and objects are entities, and predicates indicate relationships between those entities. Users can define their own properties, as well as their own classes. Classes in semantic web languages are categories or types, similar to how classes work in programming languages. Instances of these classes can then be created and described with values for related properties.

Implicit in these models is more information than can usually be found in their text representation. Each triple forms a graph with two nodes connected by an edge. Each instance can have several properties, and that graph expands to have many nodes connected to the central instance. Finally, when two instances are connected via a property, their respective sub-graphs become connected. The graphs produced from RDF triples contain more information than just which entities are related to which. Implicit information, such as the underlying structure of a data model or which instances are most closely connected is all contained in a graph. This information, though, is difficult, if not impossible, to extract from a text-based reading of the data.

Since the Semantic Web is so new and the languages themselves are still quickly evolving, much of the research in this area has focused on editors, applications, tools, and languages. Tools for viewing the data have been primarily text based. The few graphical visualization tools have focused on other aspects of the data and its use.

In this paper, we present a tool for generating graphs of ontologies and instance data on the semantic web. Using the properties that relate instances as edges, we can apply graph layout algorithms that attempt to place related nodes near to each other while keeping all other nodes evenly distributed. The resulting graph drawings give the user insight into the structure and relationships in the data model that are hard to see in text.

2. Background and Related Research

One of the most widely seen tools for graph visualizations of RDF metadata is IsAViz [2], built on AT&T's Graphviz graph visualization software. In addition to producing the graphs for the W3C's RDF validator, it is a stand alone application for browsing and authoring RDF documents. Though the graphs it generates are suited to the task, because of their layout, they are difficult to use in seeing the overall structure of a set of instances. In addition to showing instance data, IsAViz shows connections from instances back to their originating classes.

The Protégé ontology editor, produced at Stanford University, is one of the more popular open source semantic web tools available today. It is easily extensible, and has two visualization components. OntoViz [10] is an ontology visualizer that, like IsAViz, uses Graphviz as its base. It shows classes grouped with their properties, and information about those properties, and instances grouped with lists of their properties. These groups are connected by edges indicating relationships between the objects. Jambalaya, another Protégé based visualization tool, displays information similarly, with instances and classes being grouped with their respective properties. Jambalaya adds in a zooming feature, allowing users to look at the ontology at several levels of detail. Though Jambalaya comes closer to our goal of providing a view of the overall structure of

ontologies, both of these tools are designed to allow the user to browse ontologies. Because they use a visual structure that lists all of the information related to a specific object as nodes, the overall picture is full of large boxes, overlapped with edges, obscuring much of the associative structure.

The Spectacle system [1] is designed to present instance data to users in a way that aids their navigation of search results and ontologies. Each instance is placed into a cluster based on its class membership. Instances that are members of multiple classes are placed in overlapping clusters. These visualizations provide a clear and intuitive depiction of the relationships between instances and classes, as well as illustrating the connections between classes as they overlap.

In many cases, though, grouping by classes is not ideal for understanding the underlying structure of the data. For example, in looking at data about people, projects, and papers produced by an organization, it is not as useful to see people grouped together, projects grouped together, and papers grouped together. What is more illustrative is to see clusters of people based on their interactions: small clusters that group people and papers around their respective projects say much more about how the organization is structured. Similarly, in an ontology, concepts related to each other through the domains and ranges of properties, through subclasses, and through other semantic connections can be clustered according to these properties. These properties in instances and classes, which make up the edges in a graph on the semantic web, are what we use to determine the layout of our graphs.

3. Graph Drawing

Graphs are often used to visualize relationships and patterns between entities. Graph drawing methods are important in such visualizations for providing automatic layout of entities and their relationships. A good layout can ease user exploration and make it easier to detect patterns in the data. We define a graph G = (N,E), where N is the set of node entities and E is the set of directed edge relationships, each between a pair of nodes.

Spring embedding [3,4] is one such graph drawing method that is suitable for application to our data. Its effect is to distribute nodes in a two-dimensional plane with some separation, while attempting to keep connected nodes reasonably close together. The spring embedder graph drawing process considers the graph model as a force system that must be simulated. Each node in the graph is modeled as a charged particle, thereby causing a repulsive force between every pair of nodes. Each edge is modeled as a spring that exerts an attractive force between the pair of nodes it connects. The graph is laid out by repeated iterations of a procedure that calculates the repulsive and attractive forces acting on all nodes in the graph. At the end of each iteration, all nodes are moved according to the resultant forces acting on them.

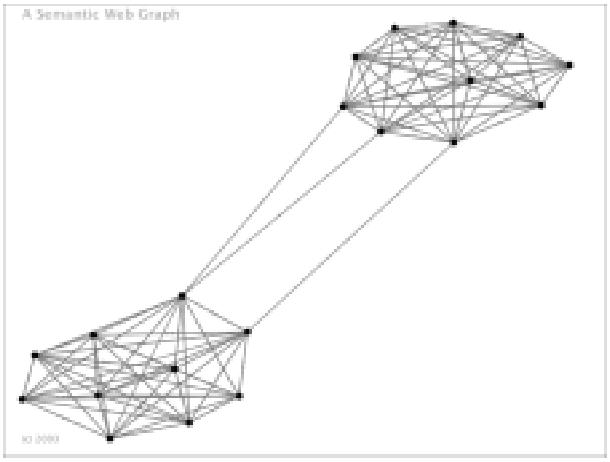

Figure 1: Spring embedded ontology

The force models that we use for the spring embedder are based on those of Fruchterman and Reingold [4]. This version of the spring embedder is effective and widely used. It is also relatively easy to implement and requires a minimal set of parameter values that can be adjusted to achieve good automatic layouts. In this model, the repulsive force acting between a pair of nodes is $-k^2/d$ and the attractive force due to an edge is d^2/k, where d is the distance between the two nodes and k is a constant. We start our graph drawing process by allocating each node to a random location on a two dimensional plane and then we begin the iterative calculation of these forces and move nodes accordingly. This results in a layout where connected nodes are close together, yet no pair of nodes are too close to each other due to the repulsive forces acting between them.

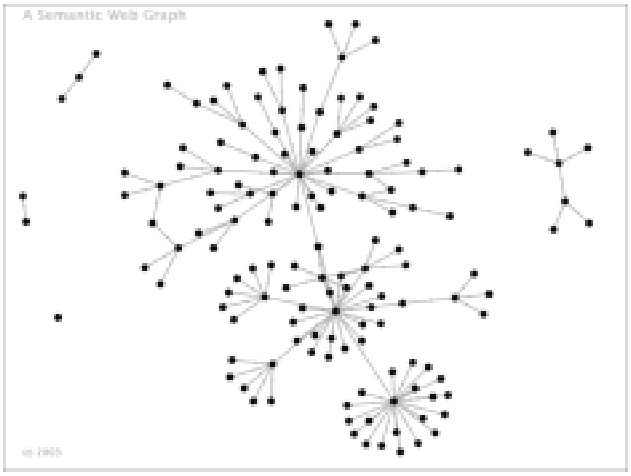

Figure 2: A disconnected graph.

When visualizing ontologies and instances, not all graphs we encounter will be connected. With a simple spring embedder model, this can cause the layout to rapidly expand, as there is nothing to counter the repulsive forces acting between each of the largest connected subgraphs. We solve this problem by limiting the distance over which repulsive forces may act. A pair of nodes with separation greater than *m* does not exert a repulsive force. This alteration to the force model ensures that we do not end up with an unnecessarily sparse graph drawing.

A simple implementation of the spring embedder calculates the repulsive force between every pair of nodes and so has a time complexity of $O(N^2)$ per iteration. In practical terms, this limits the maximum size of our drawings to several hundred nodes if we want them to be drawn in less than one second on affordable hardware. Various optimizations exist to make this process quicker, such as preprocessing the initial random layout with linear time complexity [5], speeding up the calculation of forces between pairs of nodes [6], or reducing the number of nodes that are paired [6,7]. Multi-level approaches [8,9] provide a heuristic method that clusters a graph and lays out the coarsened graph, reintroducing the other nodes in uncoarsening steps until a final drawing is produced. These can be used to reduce the time complexity of each spring embedder iteration to O(N*log*N) without any significant reduction in its effectiveness, making the method suitable for application to graphs with tens of thousands of nodes in real-time.

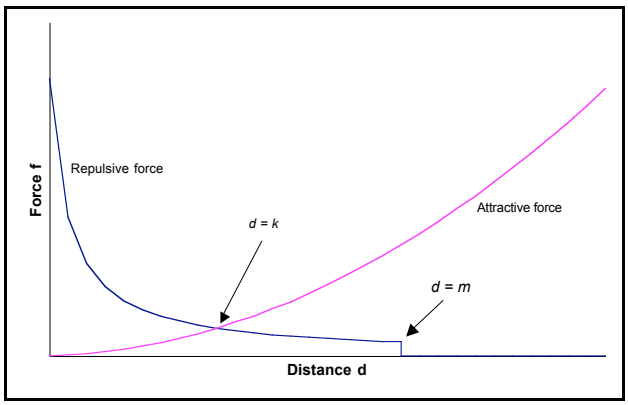

Figure 3: m-limited force model.

In this implementation, the algorithms work well on graphs with up to several hundred nodes. Clustering nodes and producing a hierarchical navigation structure with an improved implementation of the spring embedder would be useful when dealing with larger graphs. A common observation worth noting is that the spring embedding process as a whole requires a greater number of iterations to reach an equilibrium with larger graphs, so the overall time complexity is greater than that stated for a single iteration.

4. Case Study: Zoological Information Management System

The Zoological Information Management System (ZIMS) is a project of the International Animal Data Information Systems Committee (IADISC). As they state, "Standardized data collection and animal records are very important to the zoo and aquarium profession. They are critical for basic animal management and for achieving our conservation, research, and education goals. Unfortunately, zoological facilities are currently struggling with outdated software and inconsistent records that hinder our ability to efficiently and scientifically manage the animals in our care. A most notable inadequacy of the current system is that it does not track the history of group animals, such as fish and invertebrates, or environmental conditions to the extent necessary for many aquatic collections. In an effort to solve this problem, an international, coordinated effort, called the Zoological Information Management System (ZIMS) Project, is underway to improve how animal data is managed." [13]

Over a series of workshops, the ZIMS project has produced an evolving conceptual data model (CDM). The CDM is designed to clearly define rules and concepts associated with a particular area from a data management perspective. This data model contains over two hundred concepts and relations. Concepts, also called entities, are explicitly described in Table 1.

Entity Type	Description
Animal	Represents all mammals, reptiles, birds, fish, etc that are managed by a zoo or aquarium and includes actual or estimated birth date which may be postdated to capture information prior to an animal's birth. All relationships apply to individual animals or on a percentage basis for group animals.
AnimalBusinessTransaction TransferPhysical	Captures all animals that are part of a physical transfer business transaction
Animal-Collection	Captures the historical ownership of animals at a zoo or aquarium and may include a localized accession number. Note: Partial ownership of animals may be added later but is not currently considered part of core requirements.
AnimalEnclosure	Captures which animals were in which enclosure at a given point in time and supports multi-species displays and includes datetime.

Table 1: A sample description taken from the ZIMS CDM.

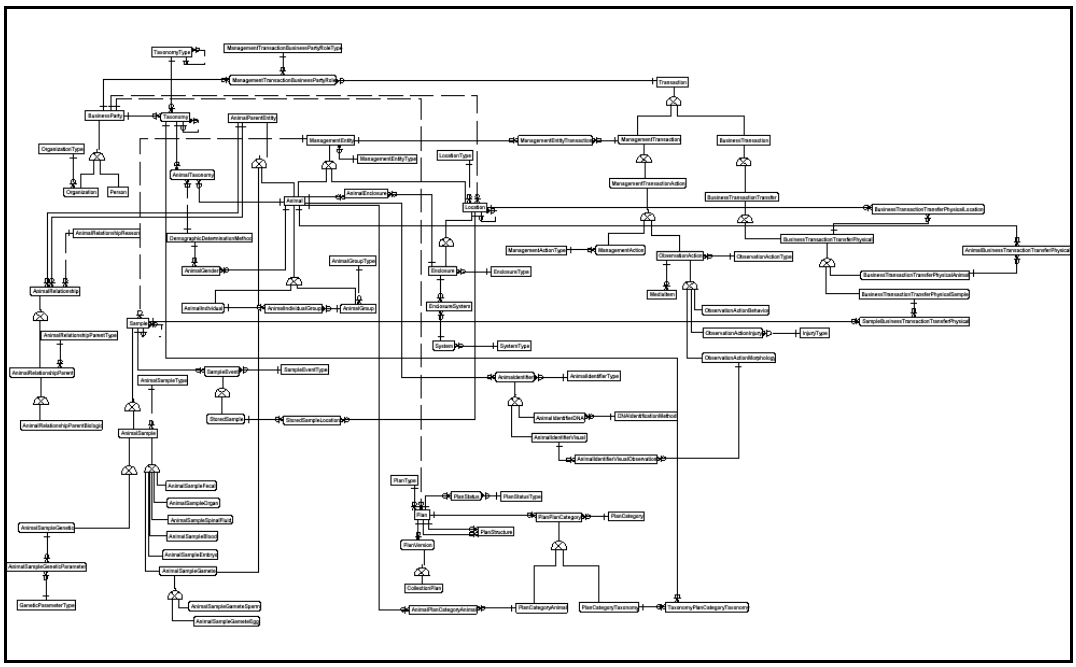

Figure 4: The ZIMS Conceptual Data Model showing entities and relations

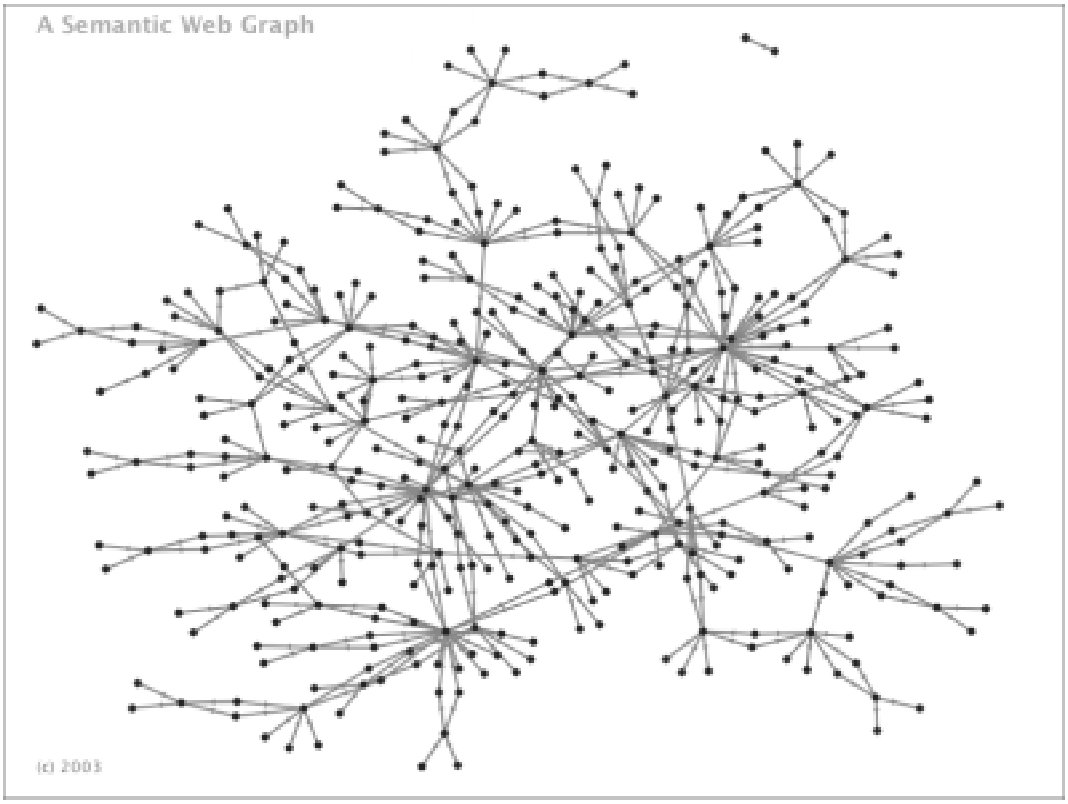

Figure 5: A spring embedded graph of the OWL version of the ZIMS CDM. This graph displays the entities and relations as well as other data associated with each entity.

Relationships between concepts are indicated by edges in a visual diagram (Figure 4). Symbols on the edges indicate the type of relationship. Just looking at the CDM diagram in Figure 4 does not convey much information about the relationships between entities. A user could certainly follow the links along to find relationships between entities, but the information is not easily visually accessible. Furthermore, there is no way to easily identify which groups of entities form closely related clusters of information and which are just adjacent by chance.

The CDM was converted into an OWL representation, describing the entities as classes, and the relationships between them as properties. This conversion to a standardized form, understandable by intelligent web agents, is a great step toward improving the way zoological information at a wide range of distributed centers is coherently managed. The conversion also puts the data into a form where it can be visualized by the graph drawing system described here.

Figure 5 shows the same CDM from Figure 4 as a spring embedded graph. For visual clarity, the labels have been removed from edges and nodes, though in application form they can be accessed as tool tips. The spring embedded graph reveals several details about the general graph structure that are not clear from the CDM diagram.

First, there is a central ring structure, made up of connected clusters. These clusters are groups of closely interrelated concepts. One example is a collection of entities in the CDM describing business transactions. Semantically, it makes sense that these terms would be connected to one another, but the clustered relationship is clear from neither the text of the ontology nor the original CDM diagram.

Long chain like structures also appear extending off around the edges of the graph. These show sequentially linked concepts, and reveals the presence of potentially important indirect relationships between the concepts at the start and end of the chain. For example, one of the longest of these chains begins with "Animal" and ends with "AnimalParentRelationshipBiologic". Again, it is intuitive that these two concepts should have some semantic relation, and through one intermediate concept and a simple string of subclasses, the two are chained together in the graph. However, the five steps separating the two concepts make it nearly impossible to recognize this relationship through text. Though tracing the path through the original CDM diagram is not difficult, there are no visual clues that would indicate it without close inspection.

By adjusting parameters of the layout algorithm, we are able to generate the graph in figure 6 with data points grouped more tightly into clusters. Though some of the nodes obscure one another, the clustering makes the structural features of the ontology all the more obvious.

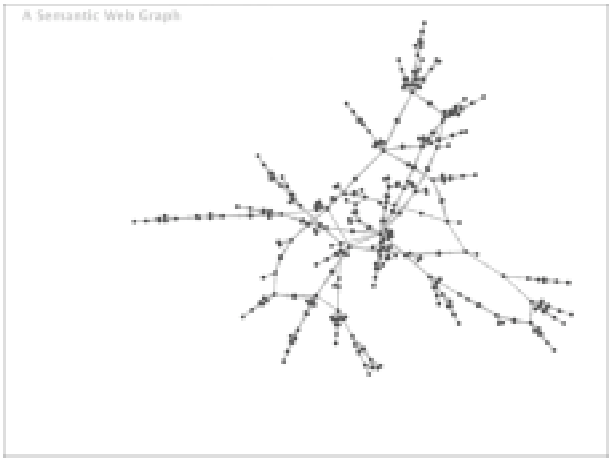

Figure 6: Spring embedded graph of the ZIMS CDM with tight clustering

5. Future Work

The next step in expanding this graph drawing system is to make it an effective tool for ontology browsing. In its current state, users can mouse over a node or edge, and see its URI. We identify two features which, when added to the graph drawing system, will increase its power to illustrate the structure of the ontology and to provide information about the elements.

The first is to add in elements of a more traditional ontology browser. Brownsauce [11] is one browser which shows all of the properties and values associated with a given instance, coupled with links to related information. Giving the user an option to click a node and see this information adds a lot of power to the exploratory process. The second improvement is to add a dynamic query interface. Dynamic queries allow the user to rapidly adjust query parameters and see those changes reflected in the visualization in real-time [12]. When viewing instance data, for example, users could select specific classes and see only instances of those classes reflected in the graph. By adding and removing instances in this fashion and quickly seeing the results in the graph, patterns about which instances are closely grouped, or how instances of specific classes are connected become clearer. This in itself is powerful and leads the way to integrating a more robust query interface to the visualization if it appears to be useful.

6. Conclusions

In this paper, we have presented a graph drawing system for visualizing ontologies and collections of instance data on the semantic web. Related entities are drawn close to each other with a directed edge to symbolize the relationship, and the system is also capable of producing sensible automatic layouts of

disconnected graphs. Through a case study involving a real, deployed ontology, we show how patterns about the underlying structure are more easily understood through the graph drawing than through text or other types of visual displays.

7. References

[1] C. Fluit, M. Sabou, F. van Harmelen, "Ontology-based Information Visualization," *Proceedings of Information Visualization '02,* 2002.

[2] E. Pietriga, "IsaViz, a Visual Environment for Browsing and Authoring RDF Models," *Eleventh International World Wide Web Conference Developers Day*, 2002.

[3] P. Eades. "A Heuristic for Graph Drawing." *Congressus Numerantium* 42. pp. 149-60. 1984.

[4] T. M. J. Fruchterman, E. M. Reingold. "Graph Drawing by Force-directed Placement." *Software – Practice and Experience* Vol 21(11). pp. 1129-1164. 1991.

[5] P.J. Mutton, P.J. Rodgers. "Spring Embedder Preprocessing for WWW Visualization." *Proceedings of International Symposium on Web Graphics and Visualization, IV02-WGV*. 2002.

[6] D. Tunkelang. "JIGGLE: Java Interactive Graph Layout Algorithm." *GD '98, LNCS* 1547. pp. 413-422. 1998.

[7] Quigley, P. Eades. "FADE: Graph Drawing, Clustering, and Visual Abstraction." *GD 2000, LNCS* 1984. pp. 197-210. 2001.

[8] D. Harel, Y. Koren. "A Fast Multi-scale Method for Drawing Large Graphs." *GD 2000, LNCS* 1984. pp. 183-196. 2001.

[9] C. Walshaw. "A Multilevel Algorithm for Force-Directed Graph Drawing." *GD 2000, LNCS* 1984. pp. 171-182. 2001.

[10] The Ontoviz Tab: Visualizing Protégé Ontologies - http://protege.stanford.edu/plugins/ontoviz/ontoviz.html

[11] BrownSauce RDF Browser: http://brownsauce.sourceforge.net/

[12] C. Ahlberg, C. Williamson, and B. Shneiderman, "Dynamic Queries for Information Exploration: An Implementation and Evaluation," *Proc. Conf. Human Factors in Computer Systems 1992,* pp. 619-626.

[13] S. DuBois, "The ZIMS Project: Building Better Animal Information Systems for Zoos and Aquariums", *presented at the Management and Technology Conference, 2003,* Orlando, FL, 2003. http://www.zims.org/reports/building_better_info_systems.htm