OSNAP! Introducing the Open Semantic Network Analysis Platform

Peter J. Radics\textsuperscript{a}, Nicholas F. Polys\textsuperscript{b}, Shawn P. Neuman\textsuperscript{c}, and William H. Lund\textsuperscript{d}

Virginia Tech, Blacksburg, VA, USA

ABSTRACT

Graph visualization continues to be a major challenge in the field of information visualization, meanwhile gaining importance due to the power of graph-based formulations across a wide variety of domains from knowledge representation to network flow, bioinformatics, and software optimization. We present the Open Semantic Network Analysis Platform (OSNAP), an open-source visualization framework designed for the flexible composition of 2D and 3D graph layouts. Analysts can filter and map a graph’s attributes and structural properties to a variety of geometric forms including shape, color, and 3D position. Using the Provider Model software engineering pattern, developers can extend the framework with additional mappings and layout algorithms. We demonstrate the framework’s flexibility by applying it to two separate domain ontologies and finally outline a research agenda to improve the value of semantic network visualization for human insight and analysis.

Keywords: Ontology Visualization, Graph Visualization, Software Frameworks.

1. INTRODUCTION: VISUALIZING GRAPHS

Unfortunately, the “Hairball Effect” is a commonly encountered phenomenon in graph visualization (see Figure 2). Even in “Small World” graphs, the connectivity among a few hundred nodes can overwhelm human and computer investigators.\textsuperscript{1,2} Yet, since the introduction of GraphViz\textsuperscript{3} or Prefuse,\textsuperscript{4} there has been little progress on developing the frameworks and libraries that enable researchers and practitioners with the expressive power needed to explore the broad design space of network visualization. While traditional tools have used the canonical visualization pipeline\textsuperscript{5} to map focused on visualizing the abstract properties and attributes of nodes and edges, there is a lack of frameworks that support visual mappings for the metadata and structural properties of the graph. In this paper, we introduce the Open Semantic Network Analysis Platform (OSNAP), a framework that endeavors to address these issues.

We focus on the interactive visualization of ontologies. Ontologies, often referred to as semantic networks, can be represented using mixed pseudo graphs (i.e., graphs that can contain loops, multiple edges between nodes, and both directed

\textsuperscript{a}peter.radics@vt.edu (corresponding author); \textsuperscript{b}npolys@vt.edu; \textsuperscript{c}spn2460@vt.edu; \textsuperscript{d}whlund15@vt.edu
and undirected edges). As such, ontology visualization faces the same problems as graph visualization, most notably the so-called “Hairball Effect”. Algorithms and toolkits for graph visualization have the disadvantage that they are not specifically targeted at ontologies. Yet ontologies have an inherent structure and schema that can be leveraged in layout algorithms: they contain hierarchical relationships (e.g., the subclass relationship) as well as data type properties, which represent paths of length one. This structure disappears in general graph layouts. The primary goal for OSNAP is to develop and evaluate new visualization techniques to analyze the abstract properties, metadata properties, and structural properties of ontologies. However, as we will demonstrate, our method and open framework also support the visualization and analysis of general graph and network types.

The metadata contained in ontologies (and many graphs) can be used to enhance its visual representation. However, custom mappings from metadata to different visual properties (such as color, shape, size, etc.) are not widely supported in graph visualization toolkits today. Moreover, current graph visualization toolkits only provide support for a limited number of input and output formats without the possibility for easy extension. This can limit the usefulness of these toolkits due to the huge variety of data formats prevalent in both research and industry. OSNAP currently supports the loading of GraphML for the case of general graphs and OWL for the case of ontologies, and allows for easy extension with further I/O plugins.

Finally, we recognize that graph visualization and layout (and specifically ontology visualization) is not a solved problem – different layouts can lead to drastically different interpretations. Current graph visualization algorithms do not provide robust mappings for the inherent structure of ontologies. Therefore, we must also develop and test new layout algorithms specifically for ontologies, leveraging their inherent structures and with the purpose of making them usable to a human user. The scoping of OSNAP (see Figure 3) means it can be used to explore the broad design space of filters, mappings, and renderings of graphs and ontologies.

Figure 2. The ‘Hairball Effect’: Ontology visualization using a 2D Force-Directed layout showing one (left), two (center), or three (right) object properties.

Figure 3. The scope of the OSNAP framework within the canonical visualization pipeline.

In the next section, we describe our motivating applications, background, and prior work in graph mining and visualization. In Section 3 we discuss the implementation of the OSNAP framework, followed by the details of an evaluative pilot study in Section 4. Section 5 features a variety of results from our framework and example application. In Section 6 we conclude by outlining the new research opportunities enabled by OSNAP.
2. BACKGROUND

Graphs and ontologies are at the heart of many recent innovations in numerous fields, ranging from telecommunication networks to social or biochemical networks due to their inherent suitability to describe complex systems. While graphs allow the representation of connections (edges or hyperedges) between two or more concepts (nodes), ontologies provide additional expressivity and structure to the connectivity information and metadata contained in pure graph representations. Notably, they allow the definition of a taxonomical hierarchy of concepts, along with the definition and restriction of semantic relationships between concepts. These additions allow the implementation of reasoners, providing automated computation of subclasses and consistency\(^6\). However, these additions can also be interpreted as additional metadata of the nodes, edges, and hyperedges of the graph, allowing the equivalent representation of an ontology as a mixed hyper graph.

Thus, graphs and ontologies are used to describe a large variety of domains. For example, the members of the U.S. National Science Foundation Industry-University Cooperative Research Center (I/UCRC) for e-Design\(^7\) are working on several ontology-based representations and showing the benefits for the knowledge capture of engineering design and product family knowledge\(^8,9\). Their e-Design Framework 2.0 ontology provides the vocabulary for modeling decisions and factors of product lifecycle management (PLM) and lifecycle assessment (LCA). The Biological Pathways Exchange (BioPAX) Level 3 ontology\(^10\) covers metabolic pathways, molecular interactions, signaling pathways (including molecular states and generics), as well as gene regulation and genetic interactions. Wordnet\(^11\) provides a model of the English language, adding additional semantic relationships between words to the synonym relationship found in thesauri.

The utility of graphs and ontologies has lead to the advent of large catalogues (e.g., the Open Biological and Biomedical Ontologies\(^12\)) as well as efforts to create the “Semantic Web”\(^13\) – the emerging network of concepts and meanings binding data on the World Wide Web. This has lead to the development of many data formats for graphs and ontologies like GraphML\(^14\), the Web Ontology Language (OWL)\(^15\), and OBO\(^12\). However, these machine-readable and machine-reasonable format of these graphs is not designed for human analysis and therefore difficult for mortals to comprehend and use. Furthermore, the vast size of some ontologies and graphs, featuring millions of nodes and edges, surpass the limits of human cognition.

To address these limitations, there are two complementary approaches: Computational Graph Query, Analysis, and Mining and Graph Visualization.

2.1 Computational Graph Query, Analysis, and Mining

As mentioned in the previous section, one of the challenges of interacting with graphs and ontologies is their potentially large size. This does not only affect human investigators in terms of information overload, but also computers in terms of memory use and number of calculations. Relational data stores can help address this problem. One such example is Neo4j\(^16\), a promising graph database. Such data stores can be accessed through a REST interface or Java API using Cypher, designed to be a humane SQL-like query language for graph stores. There are also recent, novel methods for the extraction of sub-graphs, clusters, and narratives within and between networks.

Another approach to reduce the complexity of interacting with graph-based data lies in automated analysis. Witherril et al.\(^17\) introduce an algorithm for identifying engineering relationships (AIERO). The algorithm finds connections between concepts based on semantic relatedness. This results in a narrowed search space, reducing the complexity for both computers and human investigators.

Finally, Hossain et al.\(^18,19\) describe a Java-based tool for graph mining as applied to the cellular signaling pathways in the Signal Transduction Knowledge Environment (STKE). They presented new algorithms and a graphical tool that can help biologists discover relationships between pathways by looking at structural overlaps within the database. They addressed this problem of finding pathway relationships by two data mining approaches: clustering and storytelling. In the first approach, one brings similar pathways to the same cluster and in the second, one determines intermediate overlapping pathways. Such queries can lead biologists to new hypotheses and experiments regarding relationships between the pathways. The authors formulate the problem of discovering pathway relationships as a subgraph discovery problem and propose a new technique called Subgraph-Extension Generation (SEG) that outperforms the traditional Frequent Subgraph Discovery (FSG) approach by magnitudes. Their tool also provides an interface to compare these two approaches with a variety of similarity measures and clustering techniques including terms of computational performance measures such as runtime and memory consumption.

However, these approaches do not readily allow human investigators to interactively explore graph-based data. The field of graph visualization attempts to address this issue.
2.2 Graph Visualization

The interactive, visual exploration of meanings and relationships can provide new insight into ‘familiar’ systems. However, graphs and ontologies also present unique challenges to the visualization community, often represented through the canonical visualization pipeline. Arguably the most important challenge is the creation of meaningful representations of information through visualization techniques and layouts, transforming the data through filtering and mapping into a geometric representation (see Figure 3).

2.2.1 Techniques and Layouts

There is a sizable history of research into graph visualization techniques covering many graph types and domains and tasks. In the book Information Visualization and Virtual Environments, Chen showed how 3D graph visualization could be used to interactively display citation networks and topics in digital library contents. Using techniques such as spanning trees and ‘Pathfinder networks’, Chen generates several unique views and insights using VRML as the presentation layer. When we consider designing the mappings for node and edge markers, we can refer to the suggested pre-attentive rankings of visual properties like Shape, Color, Scale, etc.

Similarly, the layout of the nodes and edges of a graph is an algorithmic and design challenge. Hierarchies are a common form of graph ontology, typified by the is-a or part-of relation. Robertson et al. and Munzner addressed the layout of hierarchical graphs with their Cone Tree and Hyperbolic 3D techniques respectively. Occlusion has been addressed with force-directed techniques such as Fruchterman-Reingold and Kamada and Kawai. The principal lessons from these works are: a) that different techniques can highlight different properties or topologies and b) that one technique does not work for all types of graphs. Indeed, for exploratory graph visualization, users may need to try several techniques.

However, in addition to the creation of a meaningful representation, the right choice of display platform for rendering can make a huge difference in terms of interaction and exploration of graph visualizations (see Figure 3).

2.2.2 Display Platforms

Networks and graphs are a common type of abstract data; in order to understand the varied relationships between entities in a network, it is crucial to acquire some spatial knowledge about the layout and connectivity of its components. There is evidence that large high-resolution displays and immersive virtual environments (IVEs) are beneficial for interacting with multi-dimensional information visualizations.

Andrews et al. found that large high-resolution displays (LHRDs) allowed users to organize their data spatially, leading to interactions with abstract data similar to that with physical objects. Furthermore, the spatial organization served as a memory aid, obviating the creation of additional summary information. However, the authors found that it took users up to several weeks to get used to LHRDs.

Henry and Polys presented an empirical study designed to determine what effect level of immersion and navigation technique can have on a user’s acquisition of spatial knowledge of network data, specifically cell signaling pathways. In this CAVE study (CAVE Automatic Virtual Environment), the level of immersion was controlled by changing the Field-Of-Regard, while the navigation was also varied between one egocentric and one exocentric technique. The results show that both immersion and navigation technique can affect the acquisition of spatial knowledge regarding abstract networks in an immersive virtual environment.

Bacim et al. also evaluated the relationship among graph complexity, immersive display fidelity, task scope, and users’ personal spatial ability. Over a variety of task types, they showed significantly better overall task performance with higher display fidelity (four CAVE screens versus one). These results show the importance of considering multiple factors when calculating the overall difficulty and complexity of a spatial task, and suggest that visual clutter makes a greater impact on speed than correctness. Further, the study of different task types suggest enhanced virtual reality displays offer more benefits for spatial search and fine-grained component distinction, but may provide less gain for sense of scale or size comparison.

Finally, visualization techniques and layouts require a framework or tool to allow developers or domain experts to interact with the rendered visualization on the various display platforms.
2.2.3 Tools and Frameworks

Fundamentally, one can distinguish between domain-specific and generic tools and frameworks. Domain-specific tools and frameworks (like, e.g., the Visual Thesaurus\textsuperscript{34} based on the Th!nkmap visualization toolkit\textsuperscript{35}) have the advantage of highly specific and tailored visualizations. However, this comes at a high cost for the development process: Developers have to frequently interact with domain experts to determine the exact functionality needed. Furthermore, these tools assume that the domain in question is reasonably well explored, since determining requirements for visualizations would otherwise not be possible. Therefore, we will focus the discussion in this section on generic graph visualization tools and frameworks.

Generic graph visualization tools and frameworks rely on abstraction of domain-specific data into metadata. As such, domain experts should be able to use that metadata to change characteristics of the visualization. An ideal framework should, furthermore, provide support for subgraphs, ontologies (or ontology loading), allow hypergraphs (edges with more than two nodes). Furthermore, it should provide dynamic three-dimensional layouts that a user can directly interact with.

In terms of tool support, a good framework would provide techniques and layouts so that users do not have to manually set visual properties of the graph-based data. To that end user interfaces for editing the graph-based data (graph editor) as well as the generated layouts (layout editor) are desirable. Table 1 shows a comparison of popular open-source graph visualization frameworks.

<table>
<thead>
<tr>
<th>Support for</th>
<th>GraphViz\textsuperscript{36}</th>
<th>prefuse\textsuperscript{4}</th>
<th>Ubigraph\textsuperscript{37}</th>
<th>Tulip\textsuperscript{38}</th>
<th>Cytoscape\textsuperscript{39}</th>
<th>Gephi\textsuperscript{40}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mappings</td>
<td>no</td>
<td>partial</td>
<td>no</td>
<td>partial</td>
<td>partial</td>
<td>partial</td>
</tr>
<tr>
<td>Subgraphs</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Ontologies</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Hypergraphs</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Dynamic Layouts</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>3D Layouts</td>
<td>partial</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Direct Manipulation</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tooling</th>
<th>Layout Algorithms</th>
<th>yes (2D)</th>
<th>yes (2D)</th>
<th>no</th>
<th>yes (2D)</th>
<th>yes (2D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph Editor</td>
<td>3rd party</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Layout Editor</td>
<td>3rd party</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 1. Framework Comparison

The data in the table reveals that none of the surveyed frameworks fulfill all of the criteria mentioned. Most notably, the support for mapping metadata to visual properties (like color, scale, etc.) is only partially supported by most frameworks. Similarly, only the Cytoscape framework supports ontologies “out of the box”, and none of them supports representation of hypergraphs. Furthermore, only GraphViz and Gephi support 3D layouts, yet do not provide any layout algorithms specifically for that purpose. Since all of the frameworks are open source, most of these functionalities could be added retroactively. The simple plugin architectures of GraphViz and Gephi have already yielded sizable community contributions. However, since the direct support of ontologies, hypergraphs, and 3D layouts would represent fundamental changes in most cases, we decided to develop a framework that addresses all issues noted above. The result of these efforts is the Open Semantic Network Analysis Platform (OSNAP).

3. IMPLEMENTATION

The goal of the Open Semantic Network Analysis Platform (OSNAP) is to provide domain experts with the means to gain insight into the semantics of their data while addressing the shortcomings of existing frameworks. In order to stay domain agnostic, to provide access to the capabilities of existing tools, and to enable domain experts in the exploration of their data, the main focus of our implementation of the OSNAP framework lay on adaptability, flexibility, and extensibility. These concerns are reflected in the framework’s architecture (shown in Figure 4), as well as its graphical interface.
3.1 Architecture

To allow portability between different operating systems and device types, we decided to implement the framework in Java. In order to prevent lock-in into a particular representation – and thus limiting the flexibility of the framework – we furthermore chose to provide our own internal representations of graphs, metadata, layouts, and visualizations. These representations constitute the core of the OSNAP framework (cf. Figure 4).

Internal Graph Format The basis of our representation are basic graph concepts that allow the definition of connectivity between concepts. Graph universes serve as root containers for any number of graphs, providing a means for avoiding explicit containment conflicts between graphs and their subgraphs by containing an authoritative version of each node, edge, and hyperedge. Edges and hyperedges are modeled as sets of endpoints, connecting them to a single node at a time, that can be either directed or undirected. Furthermore, the implementation provides access to different structural properties of the different concepts, like degree for nodes, cardinality for (hyper)edges, as well as size, rank, order, uniformity, and regularity of both graphs and graph universes.

Metadata Format Layered on top of the basic graph format are the representation of metadata and metadata schemas. Currently, numeric data (of type Integer, Long, Float, Double), Boolean data, and textual data (as String) are supported. Entries in the metadata schema allow to define which metadata are associated with a universe, graph, node, edge, or hyperedge. Furthermore, schema entries provide the means of defining whether metadata (identified by a certain key) are unique (i.e., not multi-valued) and/or required for each instance of the class they are associated with. The decoupling of metadata concepts from the underlying graph concepts allows to switch out either implementation with an alternative should the need arise. Furthermore, not all layout components (such as a radial tree coordinate layout) rely on metadata, thus rendering the metadata superfluous in those instances.

Layout Format At the center of our internal representations lies the representation of a layout. Whereas the amount of metadata that can be attached to a node, edge, or hyperedge is theoretically infinite, the same cannot be said about the number of visual design choices one can make to represent those concepts. In case of a visualization representing a graph as a network of nodes and edges, these choices can be reduced to a relatively small set of visual properties. Realistically, these properties include a subset of node shape, node scale, edge scale, color (of both edges and nodes), and node position (in x, y, and z directions). Additionally, one might provide information about nodes or edges in labels attached to the object in question. Thus, a graph layout can be generalized into a set of layout components covering the different visual properties enumerated above. To allow for additional control of the output visualization, the layout components are organized in a priority queue with newer entries potentially override values set by previous layout components. Moreover, basic filtering of nodes and edges allows to restrict the application of visual properties to a subset of the available nodes and edges. Thus, the visual properties a subset of nodes or edges can easily be changed without unnecessarily complicating the layout process.

Figure 4. Architecture of the OSNAP framework.
Visualization Format  The result of the operations of the different layout components contained in a layout is a visualization. Analogous to the structuring of the graph concepts detailed above, a visualization serves as root container of a set of visual graphs that themselves contain visual nodes and visual edges. Since the visual representation of hyperedges is an open, non-trivial problem, we decided against supporting the visualization of hyperedges in the initial version of the OSNAP framework. All concepts of the visualization format, furthermore, are linked to their corresponding concepts in the graph format. This provides layout components with access to the metadata and structural properties of the graph concepts, as well as to the visual properties. In addition to the visual properties enumerated above, visual nodes and edges contain a visibility property that allows to limit the amount of information displayed. The visibility of a visual node is determined by whether or not its coordinates, scale, color, and shape are set. The visibility of a visual edge is determined by whether or not its two endpoints are visible.

The architecture described represents a trade-off: On the one hand, providing our own representations for graph concepts, metadata, layouts, and visualizations yields complete independence from other toolkits and libraries. It also provides the flexibility of creating powerful mappings from metadata to visual properties (discussed in detail in Section 3.2) and implementing powerful layout components (discussed in Section 3.3). On the other hand, however, it requires additional effort in the creation of Import and Export Plugins (see Section 3.4) as well as Extensions (see Section 3.5).

3.2 Mappings

As discussed above, when it comes to visually representing graphs there is only a finite number of visual properties that can be varied. At the same time, there is a potentially infinite amount of metadata available in certain graphs or ontologies. This makes it impossible for anyone but domain experts to distinguish between metadata that is important for the generation of insight into a data set and metadata that is irrelevant to that task. Similarly, for certain analysis problems different structural properties of nodes or edges (such as degree or direction) may be important. Thus, it becomes essential to provide the ability to map arbitrary metadata and properties to the different visual properties available. And while it is impossible to predict the actual values of metadata or properties of different graphs, it is possible to divide the types of those metadata and property values into the following categories: continuous data, ordinal data, categorical data, interval data, set data. Analogously, the different visual properties (i.e., shape, scale, color, position, label) fall into the same categories.

This allows us to completely segment the types of mappings into four groups: one-to-one, many-to-many, many-to-one, and one-to-many. In order to maintain a domain expert’s ability to reason about the graph using visual properties, the mapping between metadata or structural property and visual property have to be consistent across the entire graph (or even the entire graph universe). Furthermore, the mappings should be able to reflect the type of both the metadata or structural property as well as the type of the visual property. However, since one-to-many relationships do not make much immediate sense (such as providing two colors based on a metadata value for a certain node) it is possible to eliminate one-to-many mappings from the list. Furthermore, since there is no straight-forward mapping between two sets, one can simply decompose many-to-many mappings of sets into sets of one-to-one mappings. Therefore, the framework provides implementations of the following value mappings:

- Single value → Single value (includes identity)
- Set → Single value
- Interval → Single value
- Interval → Interval

However, a mapping between two values in itself is ambiguous: A value mapping between a Long value and a Float value could represent a mapping from a metadata value to the scale of an edge, or alternatively a mapping between the degree of a node and its X-coordinate. Therefore, a mapping needs to provide insight into which visual property and which metadata or structural property are involved in the mapping in addition to the actual value mapping. In other words, mappings are defined through the relationship of a subset of the values of a specific domain key (i.e., a metadata type or structural property) to a subset of the values of a specific co-domain key (i.e., a visual property). This results in the internal structure of a mapping shown in Figure 5.

These mappings can be used as part of layout components providing a single visual property at a time. However, the OSNAP framework allows for the creation of layout components providing arbitrary combinations of visual properties.
3.3 Layout Components

As outlined in Section 3.1, our internal representation of a layout is based on a priority queue of layout components. This design choice follows the Provider Model design pattern, allowing arbitrary combinations of visual properties without having to previously define how they operate. As such, every layout component has to provide a set of visual properties (its capabilities), means to separately enable or disable the application of each visual property, as well as provide the means to restrict the nodes and edges that are modified by the layout component. This provides developers complete freedom to implement their own layout components, while allowing domain experts to specify the concrete visual layout of a graph. Furthermore, the layout component model provides a connection point to other visualization toolkits through the use of a layout component registry, in which components from other libraries can be registered at runtime.

The current version of the OSNAP framework provides a number of pre-defined layouts falling into three categories: simple layouts, mapped layouts, and complex layouts. The layout in Figure 6 exercises each of these. The simple layout components allow to directly specify the value of a visual property for a set of nodes or edges. As such, we implemented
a simple layout component for each visual property (i.e., shape, scale, color, position, label). The mapped layout components utilize the mapping structure outlined in Section 3.2 to provide visual property values based on mappings of either metadata or structural properties. Analogously to the simple components, we provide one layout component per visual property. Finally, the complex layout components provide multiple visual properties simultaneously. We built our own initial coordinate layout components for: a 2D grid, a 3D sphere, and a ‘Tiered Orbit’ layout.

3.3.1 Tiered Orbit Coordinate Layout Component

The ‘Tiered Orbit’ coordinate layout component, provided within the OSNAP framework, is specifically designed for the visualization of the structure of ontologies. To that end, it uses a solar system/atom metaphor leveraging the hierarchical relationships commonly found in ontologies. Concepts are placed into three orbitals around a root node chosen at runtime (of which only the middle orbital is supported in the current version). The layout algorithm arranges concepts that are connected to the root node by a chosen hierarchical object property into a tree structure. Child nodes are placed in consecutively smaller orbits around their parents on consecutively lower planes in the Y direction, creating the ‘middle orbital’. Datatype properties (attributes) of a concept are placed in tight orbits around a concept node on the same X-Z plane, creating the ‘inner orbital’. Finally, concepts that are not connected to a displayed node through a hierarchical relationship are displayed in wide orbits around a concept node on the same plane in Y direction, creating the ‘outer orbital’. An example of an ontology visualization based on the ‘Tiered Orbit’ layout can be seen in Figure 7.

Our Tiered Orbit layout algorithm uses the following steps:

1. Filter all edges from the graph (universe) that do not represent the desired relationship.
2. Calculate the minimum spanning tree of the graph (universe).
3. Calculate the radius of each node:
   (a) Calculate the circumference of the node as the sum of the diameters of the children of the node.
   (b) Calculate the radius from the circumference. If node has no children, set radius to 1;
4. Position chosen root node at origin.
5. Calculate x and z position of child node on orbit around parent based on its radius and the radii of its siblings.
6. Calculate y position of nodes based on distance from the root node in the minimum spanning tree.
3.4 Import and Export Plugins

The previous sections have shown the flexibility of the OSNAP framework in creating expressive visualizations of graphs based on decisions of a domain expert at runtime. However, while it is possible to define arbitrary graphs (or ontologies) and their visualizations completely within the internal representations, the utility of the OSNAP framework hinges on providing access to existing graph and visualization formats. The support of these input and output formats is achieved through I/O plugins (cf. Figure 4).

In its essence, every I/O plugin can be separated into two distinct groups based on their support of two distinct tasks: Serializers and Converters. Serializers provide the capability of reading the contents of a file or uniform resource identifier (URI) into a native representation in memory and/or writing content in its native format back to an output target. Converters provide the capabilities of translating content from its native format into one of the internal representations of the OSNAP framework, or vice versa. Note that neither Serializers nor converters are required to provide functionality in both input and output direction, but rather can provide either of the two. Any I/O plugin can be dynamically registered with the IOManager at runtime.

In the current version of the OSNAP framework we focused on the support of open (web) formats in both input and output. Thus, we currently provide the following I/O plugins:

- **XMLSerializer**: serializer for content in the Extensible Markup Language (XML) format,\(^{42}\) based on the Java Architecture for XML Binding (JAXB)\(^ {43}\) (input and output).
- **OWLSerializer**: serializer for ontologies defined in the Web Ontology Language (OWL),\(^ {15}\) based on the Apache Jena framework\(^ {44}\) (input only).
- **GraphMLConverter**: converter from the Graph Markup Language (GraphML) format\(^ {14}\) into the internal graph format.
- **OWLConverter**: converter from OWL into the internal graph format.
- **X3DEngine**: converter from the internal visualization format into the Extensible 3D (X3D)\(^ {45}\) format.

3.5 Extensions

OSNAP's design enables developers to extend with additional algorithms, analyses, and logic. This can be done with Structural Plugins, Metadata Plugins and Layout Plugins. One example is how OSNAP uses the Prefuse 2D visualization toolkit\(^ {4}\) toolkit as a Coordinate Provider. The Prefuse Layout Plugin provides access to the implementations of the following layout algorithms: Fruchterman-Reingold force-directed layout, Generic force-directed coordinate layout, Balloon-tree coordinate layout, Radial-tree coordinate layout and Node-link Tree coordinate layout. Furthermore, by mixing layout providers (e.g., through assigning each a different positional dimension), one can create novel graph visualizations, teasing out different features of membership and grouping, etc. (see Figure 8).

3.6 Graphical User Interface

OSNAP provides a JavaFX graphical user interface to enable users to inspect, filter, and map node and edge information to visualization. Several graphs can live in the same active ‘Universe’. Users can search and select graph information, including nodes and edges, their attributes, as well as their metadata (see Figure 9).

For each basic kind of visual property, we provide defaults settings and customization wizards for all visualization components. Mappings for shape, color, scale and x y z position (layout) can be driven by the wizard user forms and parameter entry. Projects (including graph or ontology data, layouts, and visualizations) can be saved for later editing and re-use. The GUI furthermore provides an immediate, interactive preview of what the layout will look like based on the parameters entered.

The resulting visualizations can be exported as X3D files and then run and explored in any number of commercial and open source tools, or loaded into other content creation tools and scenes. Users can use the built-in X3D navigation types (fly, examine, look-at, etc) or additionally can create mappings for predefined Viewpoints in the scene. This allows viewers to quickly jump around the scene looking at nodes and groups of interest. Keyboard shortcuts like ‘a’ to view all are
important for usability. Nodes can be filtered in OSNAP to include label mappings or not; we added a keyboard shortcut so that users can control if their click is for navigation or selection; if using ‘examine’ type navigation, the click defines a new center of rotation; if it is for selection, the label text is toggled. From this X3D document, native HTML-5/WebGL can be deployed using X3DOM; the project website contains several examples.

4. EVALUATIVE PILOT STUDY

Two undergraduates (ages 21 and 46) were employed to inventory OSNAP layouts with several combinations of Coordinate Provider dimensions. These students had no domain knowledge of the source e-Design Framework 2.0 or BioPAX Level 3 ontologies themselves. With constant shape and color, layouts were generated that varied by their coordinate layout providers and coordinate dimension map ordering for both ontologies. 42 permutations were created for visual review regarding the comprehensibility of the results. Throughout this process, the students also commented on the workflow process of the OSNAP mapping GUI and applied improvements.

Evaluation of the coordinate layout was based on a 1-7 scale where the viewers were asked to rate each visualization by how well it portrayed: **Hierarchy** (how easily the user could identify the node relationships as pertains to rank, order and dependency), **Degree** (the size and number of edges in a graph; order refers to the number of nodes in a graph), **Connectivity** (number of edges present in a graph in relation to the number of nodes), and **Aesthetics**.

Hierarchy was best displayed by using Tree layouts, which was expected. However, adding other layouts on the y or z axis frequently reduced the bounding box size which made the graph more visible and accessible. Of the tree layouts, Node Link tree and Tiered-Orbital produced the best results for hierarchy. Node Link Tree did not show well the interconnectivity of nodes on the same level in a two-dimensional plane. But this problem was eliminated by creating a Fruchterman-Reingold layout on the X and Z axes, and a Node Link layout on the Y axis. The added depth decreased...
the overall view of hierarchy slightly, but gives the added benefit of viewing the inter-connectivity of nodes at the same depth. The Tiered-Orbital layout showed hierarchy extremely well. It also showed connectivity among nodes on the same level. One of the more interesting features of the Tiered-Orbital layout was the clustering effect: child nodes that were also connected to each other appeared as clusters in distinct groups.

Connectivity in terms of total nodes and edges was well displayed using the Grid layout, but edge overlap made the layout useful for little else. Tiered-Orbital, Fruchterman-Reingold, Force-Directed, and all tree layouts were also useful for determining overall connectivity. Degree and order of the ontology are best found by the layouts that fit on the user screen. Radial Tree layout provides a good visual representation of the number of edges, but edge overlap makes it difficult to use for much else. Force-Directed and Fruchterman-Reingold layouts create good representations, but the random nature of node placement makes it slightly more difficult to ascertain certain values. Grid layout gives a good reference for order (node count) but edge overlap makes it useful for little else. However, even edge overlap can be a good indication of overall connectivity and degree of a given ontology.

5. RESULTS

Our OSNAP framework provides flexible, ‘plug-able’, mappings for visually representing the properties of an ontological network. For example, depending on the data type of the metadata or structural attribute (categorical, ordinal, or continuous), a developer can map this to visual representations such as size, color, or shape. One important benefit of this approach is the ability to consider the structural or topological properties of nodes and edges (e.g., cardinality) as input to a visual mapping (so, for example, node with more than six connections could be rendered as spheres instead of circles. Using X3D as the output format enabled us to load the OSNAP visualizations on workstations, multi-screen displays, and immersive platforms such as our CAVE (e.g. Figure 10). Recent developments in HTML5, WebGL, and Javascript ‘shim’ libraries such as X3DOM enable us to publish OSNAP visualizations to native web clients (see project website).

To handle the mixed multi-graph of semantic networks (ontologies), our data structure begins from considering a ‘Graph Universe’ where unique nodes and edges live. This ‘Graph Universe’ conforms to the Factory pattern to populate itself with the base node and edge types; thus multiple graphs may live in the same Universe. We found several software issues along the way to version 1.0 that are worthy of note:

Figure 9. Snapshot of OSNAP’s selection and filter GUI window.

---

Downloaded From: http://spiedigitallibrary.org/ on 02/19/2015 Terms of Use: http://spiedl.org/terms
In GraphML, for example, edges and hyper-edges are treated very differently and it is difficult in Java access to GraphML, IDs are lost (NMTOKENS); unique identifiers must be accessed through other metadata fields (i.e., KeyDataExtensions).

For 2D Coordinate Providers (layouts) such as Prefuse, we have to transcode our graph data structure into and out of their native format.

Finally, in our development of the mapping classes, we implemented Generics from the Number class. We encountered the type-erasure of Java and without operator overloading, were forced to redundantly express all the base numeric operations across our classes.

6. CONCLUSION AND FUTURE WORK

The first version of the OSNAP framework was designed to address the problem of ontology visualization as applied to the domain of engineering knowledge bases, specifically ontologies. Our results have shown this to be successful. The first-principles approach used in the framework design and software engineering has resulted in a flexible framework that can be used in a wide variety of domains and graphs. We have published the code on Sourceforge and released it under the liberal Apache2 license in hopes the community will adopt and extended it. Future development efforts will likely include additional I/O plugins such as a DatabaseSerializer based on the Java Persistence API\textsuperscript{47} and better support for Ontology loading and editing. Further Extensions envisioned are a jGraphT\textsuperscript{48} structural plugin and/or the Signal Transduction Knowledge Environment (STKE) metadata plugin.\textsuperscript{18,19}

OSNAP enables a new research agenda at the intersection of graph analysis and human computer interaction. It is now possible, for example, to run any number of human-subjects experiments comparing different graph visualizations, multiple display technologies such as immersive and mobile devices, as well as 3D user interfaces and collaborative analysis. Through open frameworks, data standards, and empirical usability evaluation, we can move into a new era of understanding and solving graph-based problems and knowledge-based applications.

ACKNOWLEDGMENTS

This work was partially supported by NSF Grant number 1127664.

Figure 10. An X3D OSNAP layout in an immersive virtual reality CAVE.
REFERENCES


