OntoTrix: A Hybrid Visualization for Populated Ontologies

Benjamin Bach, Emmanuel Pietriga, Ilaria Liccardi
INRIA – LRI (Univ. Paris-Sud & CNRS)
Orsay, F-91405, France
{benjamin.bach, emmanuel.pietriga, ilaria.liccardi}@inria.fr

Gennady Legostaev
St Petersburg State University
Saint Petersburg, Russia
glegostaev@gmail.com

ABSTRACT

Most Semantic Web data visualization tools structure the representation according to the concept definitions and interrelations that constitute the ontology's vocabulary. Instances are often treated as somewhat peripheral information, when considered at all. These instances, that *populate* ontologies, represent an essential part of any knowledge base, and are often orders of magnitude more numerous than the concept definitions that give them machine-processable meaning. We present a visualization technique designed to enable users to visualize large instance sets and the relations that connect them. This hybrid visualization uses both node-link and adjacency matrix representations of graphs to visualize different parts of the data depending on their semantic and local structural properties, exploiting ontological knowledge to drive the graph layout. The representation is embedded in an environment that features advanced interaction techniques for easy navigation, including support for smooth continuous zooming and coordinated views.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

General Terms

Design, Human Factors

Keywords

Semantic Web, Exploratory Visualization, Graphs, Matrices

1. INTRODUCTION

Significant research and development effort has been dedicated to the design of visual tools for the Semantic Web [5]. By making use of the richer capabilities of graphical representations, as opposed to textual representations such as RDF/XML or N3, and by abstracting from the complex syntactic details of the latter and explicitly representing relations, visual tools aim at providing better cognitive support [2] to users, from knowledge engineers to domain-expert end-users. They provide users with interactive representations of the data based upon state-of-the art information visualization techniques, better supporting tasks such as ontology understanding, discovery, search, comparison and mapping.

Most tools structure the visualization according to the concept definitions and interrelations that constitute the ontology's vocabulary (the TBox in Description Logics). While many of them do support the visualization of instance (ABox) data, instances are often treated as somewhat peripheral information. The visualization is mainly structured according to the TBox, instances that constitute the ABox being treated as leaf nodes in this tree or graph structure. Exceptions to this general observation exist, but either give a limited view of the ABox or use conventional node-link diagram representations that do not scale beyond a few hundred nodes.

Instances *populate* ontologies and represent an essential part of the overall knowledge base. Understanding instance-level data might be easier for users because of their lower level of abstraction compared to the definition of concepts based on OWL constructs, but instances will often be orders of magnitude more numerous than the definitions that give them machine-processable meaning (see, e.g., many of the datasets currently part of the *Linking Open Data* graph). As such, the visualization of instance-level data poses different but real challenges that remain to be addressed.

We present OntoTrix, a visualization technique designed to enable users to visualize, and navigate in, large instance sets and their relations. The technique is based on a hybrid network visualization that uses both node-link and adjacency matrix representations (Figures 1 and 3) to visualize different parts of the data depending on their semantic and structural properties.

2. ONTOTRIX

Ontology graphs contain many nodes and edges, and are often non planar. Two main issues with node-link diagram representations of such graphs are their inefficient use of screen real-estate and edge crossings that make dense regions difficult to read, both eventually causing scalability problems. A well-known alternative to node-link diagrams for graph visualization are adjacency matrices [3]. Nodes are represented as rows and columns, and edges as filled cells at the intersection of connected rows and columns. While node-link diagrams are good at showing the structure of relatively small and sparse graphs, adjacency matrices are very effective at showing large (better use of screen real-estate) and dense (no edge crossing) graphs. However, adjacency matrix representations are much less familiar to users than node-link diagrams, and make tasks that involve following paths in the graph more difficult [3], significantly increasing the user's cognitive load.

Our technique is inspired by a hybrid visualization called NodeTrix originally introduced by Henry *et al.* [3] and applied to social networks. The technique is very efficient at visualizing locally dense but globally sparse networks, representing the overall structure of the network using a node-link diagram and the dense subgraphs that represent communities using matrices. While the graph structure of ontologies might not always share the small-world characteristics of social networks, we believe that such a hybrid representation, combined with appropriate interaction tech-

WWW 2011, March 28–April 1, 2011, Hyderabad, India.
ACM 978-1-4503-0637-9/11/03.

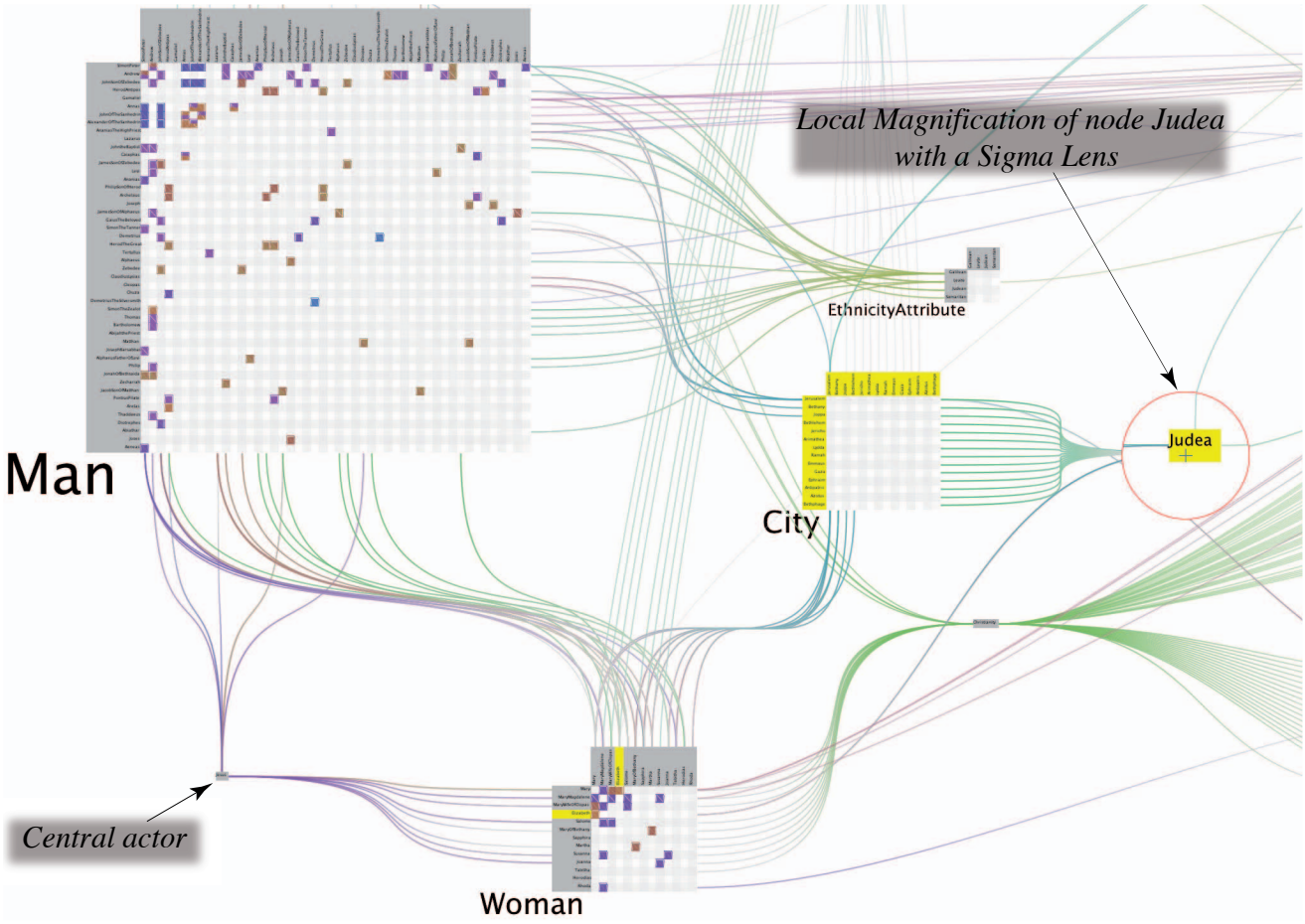

Figure 1: Region of the NTN ontology corresponding to men and women who live in cities in Judea, and people related to them.

niques, can be an efficient means to perform exploratory visualization of large ontology instance sets.

NodeTrix was originally devised for very simple, undirected social network structures that only feature one type of node (actors) and one type of relation between actors. In ontologies, instances belong to different (and possibly multiple) classes, and are connected by different types of object properties. In OntoTrix, we encode the type of object property – a categorical variable – using color: different property types are mapped to different color hues. In the property hierarchy visualization window (Figure 3-D), property types are laid out hierarchically using a radial tree layout (see also Figure 2-a). Users can dynamically change the color hue of a property simply by dragging it, positions on the circle being mapped to color hues (HSB color model). Edges that connect instances that belong to the same matrix are represented by filling the corresponding cell in the matrix with the appropriate color. If several properties connect the same two nodes, the cell is sliced horizontally, each slice colored with one of the property types.

If the graph is directed, matrices are usually read from row to column. We follow the same convention in OntoTrix, and further convey direction by only filling half of the cell to symbolize an arrow head (Figure 2-b). This also gives us some latitude to represent additional information such as symmetrical properties (filling the entire cell) and inverse relationships (filling the other half with the inverse property's color). Rows and columns in matrices are reordered using the Reverse Cuthill-McKee algorithm so that clusters within matrices can better be identified. For edges connecting instances that belong to different matrices (Figure 2-c), we visualize edge direction by varying from a shade of gray that is lowly contrasted with respect to the background color to the highly saturated color mapped to the object property type [4]. Arrow heads are no longer required, thus minimizing occlusion and cluttering.

Matrices in NodeTrix basically correspond to highly-connected groups of actors, i.e., dense subgraphs that represent social communities. In OntoTrix, we propose different methods for grouping instances into matrices, yielding different perspectives on the data.

Density: This first method is similar to the one used in NodeTrix: instances are clustered in matrices depending on density, taking into account all object properties between instance nodes. Clustering by density is driven by the LinLog energy model [8] that will also be used to compute the initial layout.

Global class membership: This method groups instances into matrices according to class membership only. The user can specify what classes in the hierarchy will be used to compute groupings.

Local class membership: This method represents a tradeoff between the previous two. From an initial grouping based on density, all nodes are grouped together on a per-matrix basis according to class membership as described above (performing a reordering of columns and rows). As a next step, each of the original matrices can be optionally split into smaller matrices corresponding to the computed class membership groups. As a result of this process, several matrices corresponding to the same class may be created. Those can be merged back together in a single matrix. The representa-

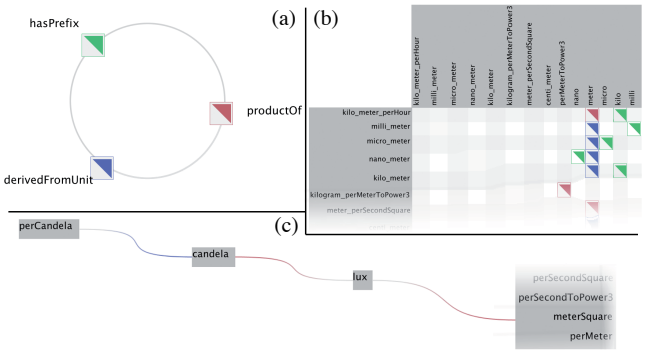

Figure 2: Conveying edge direction: (a) Mapping property type to color hue for units.owl – Semantic Web for Earth and Environmental Terminology; (b) Matrix: milli_meter derives from meter, and has prefix milli ; (c) Node-link: lux is a product of candela and meterSquare, perCandela is derived from unit candela.

tion is then equivalent to the one that would have been obtained by directly grouping with the Global class membership method.

Property type: This last method clusters resource nodes into matrices according their object properties. Properties considered irrelevant can be filtered out: those are still visible, but dimmed so that selected properties stand out.

To compute the initial representation, matrices are treated as the nodes of a higher-level aggregate graph. The edges of this graph are computed according to the object properties connecting ontology instance nodes that belong to separate matrices. As mentioned earlier, LinLog is also used to compute the layout, as it can take into account the size of nodes in the layout process (matrices can have very different sizes). After having grouped instances into matrices according to one of the above-described methods, we construct the aggregated graph structure, whose nodes are the matrices and single nodes that do not belong to a matrix, if any. Node size depends on the number of elements in the associated matrix, and edge weight depends on the number of object properties in the actual ontology graph that link instances in both matrices. OntoTrix then lays out matrices and single nodes according to the positions computed by LinLog for this aggregate graph structure, and draws cubic curves for each object property connecting two instances that belong to different matrices. These are not the edges of the aggregate graph, which were only used for the layout computation, but the actual object properties relating instance nodes.

Beyond improvements in terms of readability, we believe that this hybrid visualization features an interesting property that is related to the above view of matrices as aggregates of instance nodes for the layout process. These aggregates will of course depend on the structure of the ontology and on the grouping method selected. But often, they will by themselves represent interesting entities, not explicitly represented in the ontology, but that bear semantics. For instance, when visualizing a social network such as a co-authorship network in NodeTrix, matrices are used to represent dense subgraphs. These subgraphs will often correspond to research laboratories, or to a group of people led by a prominent researcher representing one particular research direction or theme. Each matrix can be seen as a single, compound graphical element that graphically reifies these implicit entities, thus helping users identify them.

The higher level of granularity of these *clusters* implies an inherently multi-scale representation that can help users identify entities of interest and smoothly navigate between them according to the visual information seeking mantra (overview first, zoom and filter, then details-on-demand [12]). Taking Figure 3 as an example, there

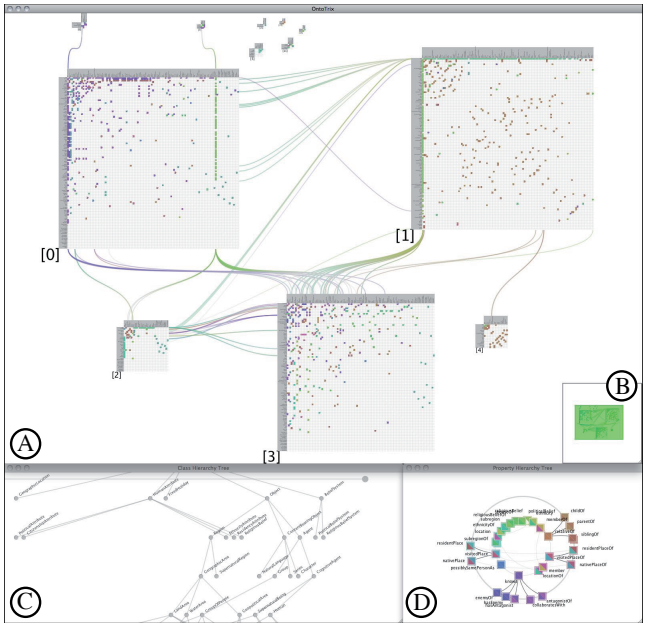

Figure 3: OntoTrix Interface Overview: (A) Main NodeTrix view, (B) Bird's eye view, (C) Class hierarchy view, (D), Property hierarchy view. Visualizing 724 instances (49 classes) and 1 636 object properties (29 definitions) from the NTN ontology.

seems to be three main large communities in the social network visualized. While matrices [0] and [3] look fairly similar (cells in both matrices have similar colors indicating that they contain instances linked with the same properties), matrix [1] looks different even from an overview, which could lead the user to investigate one of the two sets more closely. Similarly, patterns between or within matrices can often be observed and guide navigation, such as the single column in matrix [0] in Figure 3 featuring many green cells (a particular property shared by most instances populating this particular matrix), or the central actor in Figure 1.

With proper labeling, OntoTrix can also provide some basic form of semantic zooming. When grouping by class membership in OntoTrix, matrices obviously represent groups of similar instances. We thus label each matrix with the associated class name (Figure 1). When grouping by other methods, matrices cannot easily be tagged as there is no explicit information about the grouping, that stems from the purely structural clustering of the graph. Matrices might still represent interesting entities, but will have to be labeled manually and are currently assigned a random identifier.

3. TASK AND COGNITIVE SUPPORT

The OntoTrix environment features four main views presented in Figure 3. The main view (A), contains the OntoTrix representation of the instance set. (B) provides an interactive bird's eye view of (A). The class hierarchy is visualized in (C) as a node-link diagram. The property hierarchy in (D) uses a radial tree layout. All views are implemented with the ZVTM user interface toolkit [10] and support smooth continuous zooming. Most visual transitions are smoothly animated to lower the user's cognitive load when relating the two states. This includes transitions when switching between different grouping methods, e.g., from *density* to *local class membership*. This particular transition is currently limited to fading in/out the new/old matrices while trying to preserve their spatial position. More elaborate transitions using elaborate morphing techniques [3] are being investigated.

Ontology	Class definitions	Object Property definitions	Data Property definitions	Instances	Object properties	Total number of statements (TBox+ABox)	Graphical objects	Parsing time	Total time
Units[1]	13	3	5	103	146	438	1715	1.6s	2.7s
Wine[2]	138	16	1	206	246	4 547	3 296	5.5s	7.5s
NTN[3]	49	29	9	724	1 636	4 702	17 946	3.6s	10.9s
SC[4]	29	8	5	3 053	10 105	23 665	67 480	4.8s	73.5s

Tests performed with default OWL transitive reasoner, on a 2.26GHz Quad-Core Mac Pro (OS X 10.6.4). Java 1.6 set with a start and max memory heap sizes of 512MB resp. 2GB.

[1] `http://sweet.jpl.nasa.gov/1.1/units.owl`
[2] `http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine`
[3] `http://www.semanticbible.com/ntn/ntn-view.html`
[4] `http://www.mindswap.org/ontologies/SC.owl`

Table 1: Performance

Numerous interactive features are built in the Ontotrix environment to help users visualize large sets of instances. As OntoTrix is currently limited to visualization and does not support in-place editing, most supporting features are related to navigation for understanding, discovery and search [2]. We present these features using relevant items from the categorization of tasks by Katifori *et al.* based on earlier work by Shneiderman [5] and echoed in the principles enunciated in [1].

Overview: Users are presented with an overview of the entire data in all views when a new ontology is loaded. As discussed earlier, matrices can be seen as coherent aggregations of instances which, together with proper labeling, provide a multi-scale and overview-compatible representation of the ontology [2].

Zoom: All views being implemented with a ZUI toolkit [10], users can smoothly pan and zoom-in on items of interest. Global context is retained thanks to the bird's eye view (Figure 3-B). Objects have a specific location in space, thus ensuring better spatial knowledge preservation (as opposed to, e.g., expanding trees or node-link diagrams laid out using spring-based algorithms).

Filter: Properties can be selected and unselected in the object property hierarchy (Figure 3-D). Properties can thus be removed by type. When grouping matrices by property type, properties not selected are still visible but rendered with a lowly-contrasted color.

Details-on-demand: Data properties associated with nodes pop-up on demand when clicking inside a node, enabling comparisons between nodes, something difficult with non-persistent details-on-demand techniques such as tooltips, that require memorizing the previous value(s). At any given zoom level, a region of the view can be magnified (up to 14x) using a Sigma Lens [11] providing a focus+context view (right-hand side of Figure 1). Additional techniques inspired by work on topology aware navigation in large networks [7] are currently being implemented to further facilitate navigation. For instance, when zoomed-in on a large matrix, node labels might not be visible inside the viewport. Hitting a shortcut key temporarily brings row and column labels inside the viewport (*Bring & Go*). We are also implementing the *Link sliding* technique that will facilitate *"jumping between [related] concepts[/instances]"* [2] located in different matrices.

Relate: All views are coordinated. For instance, hovering a node in the class hierarchy view highlights all corresponding instances in the OntoTrix view. Hovering a node in the property hierarchy view highlights all corresponding rows and columns in matrices, as well as corresponding edges between matrices. Classes declared as being in the domain or range of the property are highlighted in the class hierarchy, and conversely. Changes to the color hue of a property type in the property hierarchy view get dynamically propagated to the OntoTrix view. Hovering an instance in the OntoTrix view highlights all nodes from owl:Thing down to the actual class the instance is a member of, and highlights all properties in the property hierarchy that are used to relate that instance to other instances.

Search: Specific instances can be searched for; the main view smoothly pans and zooms to each matching instance sequentially.

4. IMPLEMENTATION

Rather than a full-featured ontology development environment, we consider OntoTrix as a new visualization technique that complements existing solutions, and could be integrated into applications such as Protégé [6]. OntoTrix is implemented in Java. The user interface is based on the ZVTM user interface toolkit [10]. LinLogLayout [8] is used for layout and clustering (grouping by *density*), Jena 2 for loading ontologies, and the TDB backend for storing them. The built-in OWL transitive reasoner is enabled by default, providing a complete classification of the ontology, i.e., calculating the complete class and property hierarchies. Additional reasoning can be performed before storing and visualization using one of the Jena reasoners or an external reasoner such as Pellet [9]. To evaluate the performance of OntoTrix, we performed tests with a set of ontologies ranging from a few hundred to several thousand instances and relations. Results are reported in Table 1, and show that loading times start to degrade when we try to visualize several tens of thousands of statements. While larger ontologies exist, they cannot be effectively visualized in their entirety with OntoTrix or any other tool we are aware of.

5. REFERENCES

[1] T. d'Entremont and M.-A. Storey. Using a degree of interest model to facilitate ontology navigation. In *Proc. VL/HCC '09*, pages 127 –131. IEEE, 2009.

[2] N.-A. Ernst, M.-A. Storey, and P. Allen. Cognitive support for ontology modeling. *IJHCS*, 62(5):553–577, 2005.

[3] N. Henry, J.-D. Fekete, and M. J. McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE TVCG*, 13(6):1302–1309, 2007.

[4] D. Holten and J. J. van Wijk. A user study on visualizing directed edges in graphs. In *Proc. CHI '09*, pages 2299–2308. ACM, 2009.

[5] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, and E. Giannopoulou. Ontology visualization methods—a survey. *ACM CSUR*, 39(4):10:1–10:42, 2007.

[6] H. Knublauch, R. Fergerson, N. Noy, and M. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *Proc. ISWC 2004*, pages 229–243. Springer, 2004.

[7] T. Moscovich, F. Chevalier, N. Henry, E. Pietriga, and J.-D. Fekete. Topology-aware navigation in large networks. In *Proc. CHI '09*, pages 2319–2328. ACM, 2009.

[8] A. Noack. Energy-based clustering of graphs with nonuniform degrees. In *Proc. Graph Drawing*, pages 309–320. Springer-Verlag, 2005.

[9] B. Parsia and E. Sirin. Pellet: An OWL DL Reasoner. In *Proc. Description Logics Workshop*, 2004.

[10] E. Pietriga. A toolkit for addressing hci issues in visual language environments. In *Proc. VL/HCC '05*, pages 145–152. IEEE, 2005.

[11] E. Pietriga, O. Bau, and C. Appert. Representation-independent in-place magnification with sigma lenses. *IEEE TVCG*, 16(03):455–467, 2009.

[12] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proc. VL '96*, pages 336–343. IEEE, 1996.