Visual Exploration of RDF Data*

Jiří Dokulil[1] and Jana Katreniaková[2]

[1] Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic
jiri.dokulil@mff.cuni.cz
[2] Faculty of Mathematics, Physics and Informatics,
Comenius University, Bratislava, Slovakia
katreniakova@dcs.fmph.uniba.sk

Abstract. We have developed and implemented [1,2] infrastructure and RDF storage for the Semantic Web. When we filled it with data the need for some tool that could explore the data became evident. Unfortunately, none of existing solutions fulfills requirements imposed by the data and users expectations. This paper presents our RDF visualizer that was designed specifically to handle large RDF data by means of incremental navigation. A detailed description of the algorithm is given as well as actual results produced by the visualizer.

1 Introduction

The RDF [3] is one of data formats of the Semantic Web. In RDF the information is encoded as a set of statements about resources. These statements may abstractly be viewed as a graph. The data storage for RDF data is at the core of the Semantic Web infrastructure that was created at the Faculty of Mathematics and Physics of the Charles University in Prague [4]. Since its creation a lot of RDF data was loaded into the storage and a query API is available to access the data. However, not knowing the exact structure of the data even programmers using the infrastructure find it difficult to create a meaningful query. We have therefore decided that some kind of visualization tool is definitely necessary to support further development.

Working with RDF data brings up several issues. Most important of them is the size of the data. The data can be huge (millions of nodes and edges) and contain nodes with extremely high degree (thousands or even hundreds of thousands). This not only limits the possibilities of drawing the graph but also the acceptable complexity (both time and space) of the drawing algorithm. Traditional graph-based techniques work very well for small graphs. Unfortunately, the difficulty of finding readable layout extremely increases with the size of graph. We have therefore focused on finding an approach that is effective both from complexity and user point of view. One possibility to partially overcome the

* This research was supported in part by the National programme of research (Information society project 1ET100300419) and VEGA 1/3106/06.

problem with large data is an incremental navigation [5]. We decided to use the incremental navigation enhanced by our novel *node merging* technique so that we can draw even nodes with large degree. To make the drawing easily readable we proposed a *triangle layout* algorithm [6,7].

The structure of the paper is as follows. Section 2 gives the overview and comparison of known layout algorithms. It also gives a detailed description of our triangle layout. Implementation issues—including the node merging—are described in Section 3. Closing remarks appear in Section 4.

2 Visualization Algorithm

Since the RDF data can be extremely large, some kind of incremental exploration and visualization technique [5] is necessary. The user is given the possibility to explore the neighborhood of the displayed subgraph by extending the displayed part of the graph by one (or more) nodes. This way a *navigation tree* for the data is created. This tree stores the nodes and edges that are currently displayed to the user. We focus on drawing of the navigation tree. The non-tree edges can easily be drawn as lines between corresponding vertices.

2.1 Comparison

There are more approaches to drawing of trees. One of the common techniques is layered drawing where nodes are placed on layers that contain nodes with the same depth. This layers can have different shapes (lines, circles, squares, ...). Examples include:

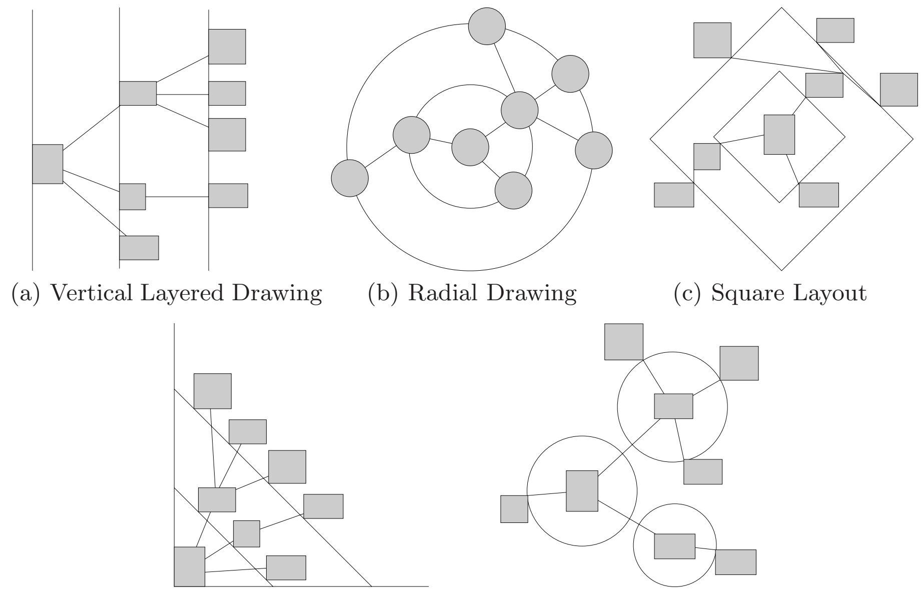
Vertical Layered Drawing (Fig. 1(a)). The layers are vertical lines. It is a very simple approach with good results. The paths in the tree are very easy to follow. The disadvantage is that it is not easy to add not-tree edges to the graph. This approach is used by Experimental RDF Visualizer created in HP labs [8] that avoids non-tree edges by duplicating parts of the graph and transforming it to a tree. Another example is IsaViz [9].

Horizontal Layered Drawing. Is a variant of the vertical layered drawing. It is rarely used because unlike vertical drawing, that offers plenty space for node and edge labels, long node labels make this layout impractical.

Radial Drawing (Fig. 1(b)). The nodes are placed on concentric circles with increasing diameters. The root of the tree is placed in the center. The nodes are usually displayed as circles but as radial drawing is an extremely common technique, there are plenty of variants. Examples of uses of this technique include gnutellavision [10] and GViz [11].

Square Layout (Fig. 1(c)). Square layout is a variant of the radial layout that uses concentric squares instead of circles. It is better suited for drawing rectangular nodes [6].

Triangle Layout (Fig. 1(d)). Triangle layout was introduced as a modification of square layout that uses only the first quadrant of the plane (with coordinate origin in the center of the squares). It is further described in the following parts of this paper.


(a) Vertical Layered Drawing (b) Radial Drawing (c) Square Layout

(d) Triangle Layout (e) Ferris-Wheel Layout

Fig. 1. Layout algorithms

There are more approaches to drawing trees than just layered drawing, including:

Ferris-Wheel Layout (Fig. 1(e)). The Ferris-Wheel layout is inspired by the radial layout but only leaves that are direct neighbors of a node are displayed on a circle around the node. Other nodes are positioned in the drawing space without any sophisticated layout algorithm and positioning them to a 'good' position is left to the user. To handle nodes with high degree, the user is given the option to zoom in on one of the circles (called *wheels*) and gradually explore the nodes by rotating the wheel. This approach is used in PGV [12].

Spring Embedding. Spring Embedding does not specify an exact algorithm for positioning of the nodes. The nodes are connected by *springs* that either pull them together or push them apart. Then the effect of the springs is simulated until a stable position is reached. In the basic version, the connected nodes are connected by springs that pull them together and unconnected nodes with ones that push them apart. By changing power or direction of the springs, layouts with more complex characteristics can be achieved. This approach is used for instance in RDF Gravity [13].

Although many different techniques can be used for the visualization, it is difficult to find a precise way of evaluating them. We have set up several criteria to compare different layout techniques. Some of these criteria are requirements imposed on the layout algorithms by the nature of the RDF data while other criteria were set up to improve user-friendliness of the resulting application. Note that numbers in parentheses after the criteria definitions correspond to numbers of columns in the Table 1.

Data-imposed criteria. Based on the experience with real RDF data we can assume that the data will contain nodes with high degree. Even such nodes should be displayable without making the visualization unreadable to the user (1). For the same reason the area that can be used to draw children of a node should not be too limited (2). Although it may not always be the case, there is a significant chance that number of nodes on each level will be much larger than on the previous one. Thus the size of the layers increase gradually (3).

User-imposed criteria. The visualization should be *well-arranged*. But there is no general understanding of what that means [14]. We have picked several criteria we believe are important when working with RDF data.

The user should be able to easily locate ancestor and descendants of a node (4). If the user follows a certain path through the tree, then the whole path should at least roughly maintain the same direction (5). Last but not least, the area required to draw the graph should not be too large (6).

Table 1. Comparison of different layout techniques

	1	2	3	4	5	6
Radial Layout	☹	part of annulus wedge	☺	☺	C	A
Vertical Layered Drawing	B	whole layer	☹	☺	to the right	☺
Horizontal Layered Drawing	B	whole layer	☹	☺	downwards	☹
Square Layout	☺	limited	☺	+/−	☹	☺
Triangle Layout	☺	whole layer	☺	☺	C	☺
Ferris-Wheel Layout	☺	not limited	0	☺	☹	☹
Spring Embedding	B	not limited	0	☹	☹	☺

A: The radial layout is best suited for drawing circular nodes. With rectangular nodes the available area can be used inefficiently if the nodes are placed onto the layer in a wrong order. If incremental navigation is used, the correct order cannot be maintained without reordering the nodes.
B: The *node merging* – can be used to handle nodes with high degree.
C: Although the path does not follow a direct route from the center, it generally follows a certain direction without significant deflections.

We have evaluated the listed drawing techniques according to the selected criteria. The results are summarized in the Table 1. The presented results are either claimed by the authors of the individual algorithms or can be easily deduced by examining the algorithms. Although this is certainly not a definitive comparison of existing tree drawing techniques, the results show that the idea of triangle layout is worth exploring. We have created an experimental implementation. There is currently no other implementation we are aware of.

The next part of the text gives a more detailed description of the triangle layout algorithm and it's properties while Section 3 is focused on the implementation.

2.2 Triangle Layout Algorithm

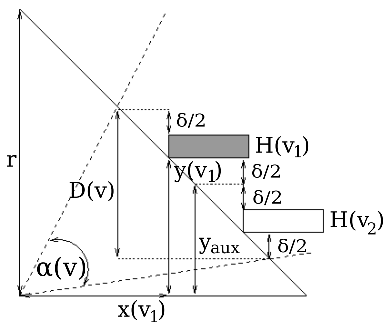
The purpose of the algorithm is to determine positions of the part of the graph that is visible at the moment. The edges of the graph that the user used to reach the visible nodes form a navigation tree T with root r_T. The children of node v are nodes that were reached by exploring edges connecting them to the node v. The order of the children is the same as the order in which they were reached. All nodes with the same distance from the root form a *layer*. A node with distance i from the root is placed in layer l_i (by $L(h)$ we denote nodes on the level h of the tree and $L(0) = \{r_T\}$). Layers are represented as lines connecting $[r_i, 0]$ and $[0, r_i]$, where the value r_i is called *radius of layer* l_i.

The nodes are drawn as rectangles $\Gamma(v)$ that are $H(v)$ pixels high and $W(v)$ pixels wide. They are labeled by URI or literal value of the node they represent and also display a list of edges that start or end in the node (for further details see Subsection 3.1). The rectangle $\Gamma(v)$ representing node $v \in L(i)$ is placed from the outside of the line representing l_i (we place the lower left corner of the vertex onto the line). The corner of the rectangle $\Gamma(v)$ that lies on the layer is denoted $\gamma_0(v)$ in the following text, while the opposite corner is denoted $\gamma_1(v)$. The radius r_i of each level is computed so that $r_{i+1} > r_i$ and to make sure there is enough space to place all nodes that belong to the layer l_i.This is influenced by the fact that we place descendants of node v into a so called *angle of influence* of the node v. The angle of influence is actually defined by two angles that define lower and upper boundary where all descendants (even indirect ones) must fit. This way each path in the tree is given a certain direction to follow, which was one of the user-imposed criteria defined in the previous section. Having $\alpha_1(v), \alpha_2(v) \in \langle 0, 90\rangle$ and radius r the *vertical range* (height of the available space in pixels) available to node v is

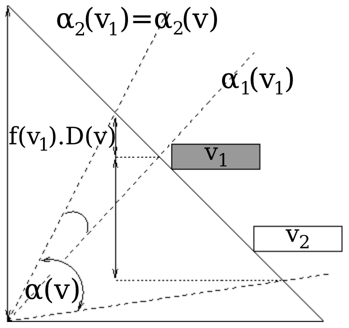
$$D(v) = r. \left(\frac{sin\ \alpha_1(v)}{sin\ \alpha_1(v) + cos\ \alpha_1(v)} - \frac{sin\ \alpha_2(v)}{sin\ \alpha_2(v) + cos\ \alpha_2(v)} \right)$$

We fit the successors of v into this vertical range. Let $v_1 \ldots v_k$ be the children of the vertex v and let v_i have a size of $H(v_i) \times W(v_i)$. If the minimal distance between vertices is δ, then the minimal required vertical space for the children of v is $\Sigma_{i=1}^{k}(H(v_i) + \delta)$. Hence the inequality $D(v) > \Sigma_{i=1}^{k}(H(v_i) + \delta)$ should hold.

The layout algorithm first displays the root on the coordinate origin (i.e. $r_0 = 0$). For each depth h of the tree (beginning with $h = 1$) the algorithm works as follows (see also Fig. 2):



(a) The vertical range



(b) Vertical range distribution

Fig. 2. Layout algorithm – explanation

Let r_{cont} be such radius, that triangle $[r_{cont}, 0], [0, 0], [0, r_{cont}]$ completely contains all vertices in layers $l_1 \dots l_{h-1}$. For each vertex $v \in L(h-1)$ the angle of influence has already been computed. Let $v_1 \dots v_k$ be the children of v and $H(v_1) \dots H(v_k)$ their heights. From the inequality $D(v) > \Sigma_{i=1}^{k}(H(v_i) + \delta)$ we compute the minimal required radius r_{min} for the children of v. Let r be the maximum of the minimal required radii and the radius r_{cont}. The vertices from $L(h)$ will be placed on layer with radius r. Radius r of the square and the angle of influence of vertex v determine the vertical range $D(v)$ for the sub-tree rooted in v. The distance $\delta(v)$ between children of v has to be recomputed from the inequality $D(v) > \Sigma_{i=1}^{k}(H(v_i) + \delta(v))$. Now, having global parameter r – radius of the layer and for each vertex $v \in L(h-1)$ the parameter $\delta(v)$, we can compute the display coordinates of children of v and their angles of influence. More formally, for each v_i with height $H(v_i)$ we determine the angle of influence of v_i and the coordinates of $\gamma_0(v_i)$.

The angle of influence of the node v is divided among the children of v according to a function $f: V \rightarrow \langle 0, 1 \rangle$ where $\Sigma_{i=1}^{k} f(v_i) = 1$.

```
LAYOUT ALGORITHM($T$)
 1  $\gamma_0(r_T) \leftarrow [0,0]$   //Place the root vertex $r_T$ to the coordinates origin
 2  $\alpha_1(r_T) \leftarrow 0,\ \alpha_2(r_T) \leftarrow 90$
 3  for each $h$ in $\{1, 2, \dots\}$
 4     do
 5         for each $v$ in $L(h-1)$
 6            do COUNT($r_{min}(v)$)
 7         $r \leftarrow max\{r_{cont}, max\{r_{min}(v) \mid v \in L(h-1)\}\}$
 8
 9         for each $v$ in $L(h-1)$
10            do COUNT($\delta(v)$)
11                $D(v) \leftarrow r \cdot \left(\frac{sin\ \alpha_2(v)}{sin\ \alpha_2(v)+cos\ \alpha_2(v)} - \frac{sin\ \alpha_1(v)}{sin\ \alpha_1(v)+cos\ \alpha_1(v)}\right)$
12         for each $v$ in $L(h-1)$
13            do $\alpha_1(v_0) \leftarrow \alpha_2(v),\ \gamma \leftarrow \alpha_2(v)$
14                for $i = 1$ to $k$
```

```
15                **do**
16                    $y_{aux} \leftarrow r \cdot \frac{sin\ \gamma}{sin\ \gamma + cos\ \gamma} - H(v_i) - \delta(v)$
17                    $\gamma \leftarrow arctg\ \frac{y_{aux}}{r - y_{aux}}$
18                    $y(v_i) \leftarrow y_{aux} + \frac{\delta(v)}{2}$
19                    $x(v_i) \leftarrow r - y(v_i)$
20
21                    $\alpha_2(v_i) \leftarrow \alpha_1(v_{i-1})$
22                    $y_{aux} \leftarrow r \cdot \frac{sin\ \alpha_2(v_i)}{sin\ \alpha_2(v_i) + cos\ \alpha_2(v_i)} - f(v_i) \cdot D(v)$
23                    $\alpha_1(v_i) \leftarrow arctg\ \frac{y_{aux}}{r - y_{aux}}$
```

2.3 Vertical Range Distribution

The angle of influence of a node is divided among its children according to the function f. Let v be a node and $u_1 \dots u_k$ children of v. The only constraint for the function f imposed by the algorithm is that $\Sigma_{i=1}^{k} f(u_i) = 1$. The choice of the function greatly affects the behavior of the visualization algorithm. In [6] we proposed the following definition of f.

$$f(u_i) = \frac{H(u_i) + \delta}{\sum_{j=1}^{k}(H(u_j) + \delta)}$$

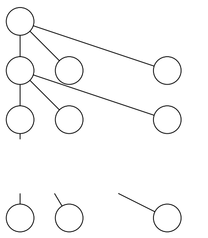
In the following text we use $r_i^{req}(v)$ to denote the minimal radius of layer l_i such that all children of node $v \in L(i-1)$ fit into the angle of influence of the node v. We also use r_i^{req} for $max\{r_i^{req}(v) \mid v \in L(i-1)\}$.

Consider a tree (see Figure 3) $T_{k,p} = (V, E)$ where all nodes are of the same size (H and W) and

$$V = \{v_{0,1}\} \cup \{v_{i,j} \mid i \in \{1 \dots p\} \ \wedge \ j \in \{1 \dots k\}\}$$
$$E = \{(v_{i,1}, v_{i+1,j}) \mid i \in \{0 \dots p-1\} \ \wedge \ j \in \{1 \dots k\}\}$$

On every level of the tree, there is a *critical node* v such that $r_{i+1}^{req}(v) = r_{i+1}^{req}$. Clearly $v_{0,1} \dots v_{p-1,1}$ are critical nodes. We denote $v_{i,1}$ as v_i in the following text.

For a critical node v_i the angle of influence covers $\left(\prod_{j=0}^{i-1} k\right)^{-1}$ of the total vertical range of level l_{i+1}. We need to place k children of v_i into this fraction of the vertical range. Thus r_{i+1}^{req} of the level l_{i+1} is $r_{i+1}^{req} = r_{i+1}^{req}(v_i) = H. \prod_{j=0}^{i} k$.


(a) $T_{k,p} = (V, E)$

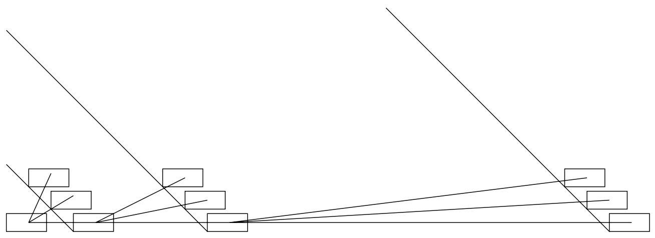

(b) Tree $T_{3,3}$

Fig. 3. Example of a tree that requires large area

The number of nodes in the tree $T_{k,p}$ is $N = k.p + 1$, so the radius of l_p is

$$r_p \geq r_p^{req} = H \cdot k^p = H \cdot \left(\frac{N-1}{p}\right)^p$$

So the area required to draw the graph grows exponentially with the number of nodes. This is not a good result from the theoretical point of view and it was also confirmed by the implementation of the algorithm using real-world data.

A better choice seems to be such function f where the value of $f(u_i)$ is the number of nodes of $T(u_i)$ (the tree rooted in u_i) divided by the number of nodes of $T(v)$. In the following text we will prove, that this function produces better drawings of the tree. The number of the nodes of $T(v)$ is denoted $N(v)$ while N denotes the number of nodes in the whole tree. First, we compute $H = max\{H(v) + \delta \mid v \in V\}$ and $W = max\{W(v) + \delta \mid v \in V\}$ and use them as the heights and widths of all nodes in the graph and use $\delta = 0$.

Lemma 1. *Every node v is assigned at least $\frac{N(v)}{N}$ of the vertical range available to the whole layer that the v is positioned on.*

Proof. For the root r_T the statement holds ($\frac{N(v)}{N} = 1$).

Let v be a child of r_T. Then v is assigned $\frac{N(v)}{N-1}$ of the vertical range and $\frac{N(v)}{N-1} > \frac{N(v)}{N}$ so the statement holds.

Let v be a child of u. We already know, that u was assigned at least $\frac{N(u)}{N}$ of the vertical range. This vertical range is divided among the children of u. The node v is assigned $\frac{N(v)}{N(u)-1}$ of the range of u, which totals to $\frac{N(u)}{N}\frac{N(v)}{N(u)-1} = \frac{N(u)}{N(u)-1}\frac{N(v)}{N}$. Since $\frac{N(u)}{N(u)-1}\frac{N(v)}{N} > \frac{N(v)}{N}$ the statement holds. $\square$

Lemma 2. *For every node $v \in L(i-1)$ the required radius $r_i^{req}(v)$ is at most $N \cdot H$.*

Proof. The node v is assigned at least $\frac{N(v)}{N}$ of the vertical range. The range has to be divided among the children of v which means at most $N(v) - 1$ nodes. Height of each child is H so the total height of the children of v is at most $H \cdot (N(v) - 1)$. The $\frac{N(v)}{N}$ fraction of the whole vertical range has to cover the height of the children and since the total vertical range is equal to the radius r_i the value of r_i must be big enough for $r_i \frac{N(v)}{N} \geq H(N(v) - 1)$ to hold. This is equivalent to $r_i \geq H\frac{N(v)-1}{N(v)}N$. The value $r_i = N \cdot H$ fulfills this condition. The condition $r_i \geq max\{r_i^{req}(v) \mid v \in L(i)\}$ implies $r_i^{req}(v) \leq r_i = H \cdot N$. $\square$

For every layer l_i of the tree, the radius r_i^{req} required to fit all children is lesser than $N \cdot H$. The actual radius of layer l_i is one of the following

- r_i^{req} if $r_{i-1} + (H + W) < r_i^{req}$
- $(H + W) \cdot i$ if the path to the root contains no layer j where $r_j^{req} = r_j$.
- $r_j^{req} + (i - j - 1) \cdot (H + W)$ where l_j is the first layer on the path to the root where $r_j^{req} = r_j$.

The maximal number of layers is $N-1$. For the last layer l_p, the r_p is one of the values:

- $r_p^{req} \leq N \cdot H$ (inequation holds due to Lemma 2)
- $(H+W) \cdot p \leq (H+W) \cdot N$ since $p \leq N-1$.
- $r_j^{req} + (p-j-1) \cdot (H+W) \leq r_1^{req} + (p-2) \cdot (H+W) \leq N \cdot H + N \cdot (H+W)$ (Lemma 2 and $p \leq N-1$).

Thus $r_p \leq N \cdot H + N \cdot (H+W) = N(2 \cdot H + W)$. The area required to draw the graph grows quadratically with the number of nodes but also with the value of H and W. Since for rooted trees the layered drawings have quadratic area requirement [15], the area is optimal. The widths of the nodes are limited by the length of the longest label in the data. The heights are limited by the highest node degree present in the data. Although both of these numbers could be potentially very large (causing H and W to be large), for practical reasons they can be limited by much lower threshold (only first part of the labels and some of the edges are displayed). The user may still be given another way of accessing the complete information. This approach is used in our implementation.

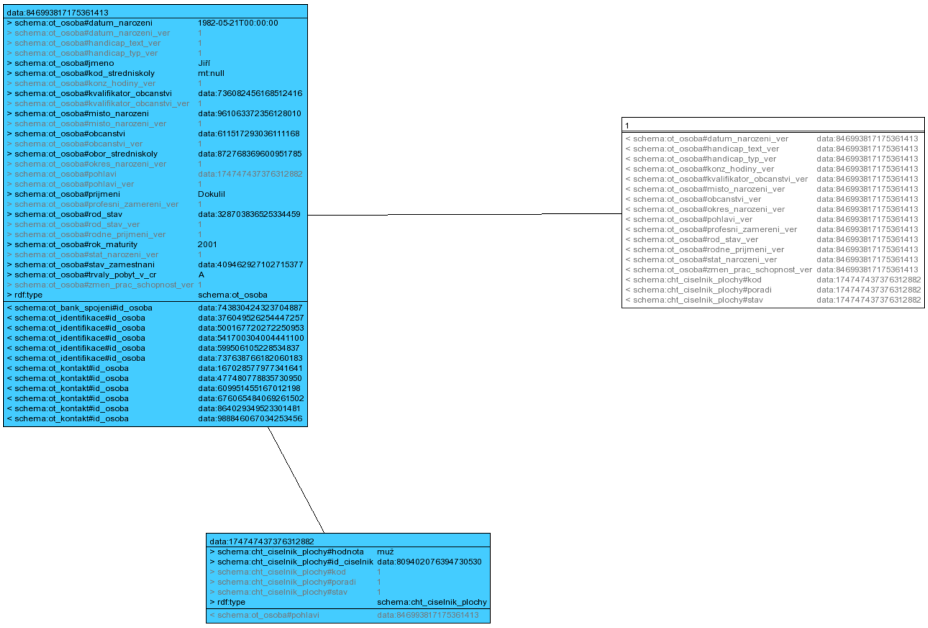
3 Implementation

We have implemented the proposed algorithm using the Semantic Web infrastructure developed at the Faculty of Mathematics and Physics of the Charles University in Prague [4,1,2].

The layout algorithm is implemented independent of the data-source and the user interface. At the moment, there is only a SDL-based user interface. This interface displays the drawing to the user and enables him or her to scroll through the whole drawing (it may not fit on the screen) and expand edges by clicking their label in a merged node. For implementation reasons, the drawing is turned upside down, so the origin of the coordinate system is in the upper left corner and the y-axis grows downwards.

3.1 Node Merging

We use our novel technique called *node merging* to help the user navigate the graph. Vertex does not contain only its label but also list of incoming and outgoing edges. This allows us to present the neighbors of the vertex to the user without using too much space. Important advantage of this approach is the fact that the user picks only the neighbors he or she is interested in and the view is then extended only by these vertices. This way we eliminate problem that a RDF node can have thousands (or even hundreds of thousands) of neighbors. Without node merging we would either have to display all of the neighbors which would hardly create a well-arranged and readable drawing of graph or the algorithm would have to pick only a few of the neighbors to display. If node merging is used



data:846993817175361413	
> schema:ot_osoba#datum_narozeni	1982-05-21T00:00:00
> schema:ot_osoba#datum_narozeni_ver	1
> schema:ot_osoba#handicap_text_ver	1
> schema:ot_osoba#handicap_typ_ver	1
> schema:ot_osoba#jmeno	Jiří
> schema:ot_osoba#kod_stredniskoly	mt:null
> schema:ot_osoba#konz_hodiny_ver	1
> schema:ot_osoba#kvalifikator_obcanstvi	data:736082456168512416
> schema:ot_osoba#kvalifikator_obcanstvi_ver	1
> schema:ot_osoba#misto_narozeni	data:961063372356128010
> schema:ot_osoba#misto_narozeni_ver	1
> schema:ot_osoba#obcanstvi	data:611517293036111168
> schema:ot_osoba#obcanstvi_ver	1
> schema:ot_osoba#obor_stredniskoly	data:872768369600951785
> schema:ot_osoba#okres_narozeni_ver	1
> schema:ot_osoba#pohlavi	data:174747437376312882
> schema:ot_osoba#pohlavi_ver	1
> schema:ot_osoba#prijmeni	Dokulil
> schema:ot_osoba#profesni_zamereni_ver	1
> schema:ot_osoba#rod_stav	data:328703836525334459
> schema:ot_osoba#rod_stav_ver	1
> schema:ot_osoba#rodne_prijmeni_ver	1
> schema:ot_osoba#rok_maturity	2001
> schema:ot_osoba#stat_narozeni_ver	1
> schema:ot_osoba#stav_zamestnani	data:409462927102715377
> schema:ot_osoba#trvaly_pobyt_v_cr	A
> schema:ot_osoba#zmen_prac_schopnost_ver	1
> rdf:type	schema:ot_osoba
< schema:ot_bank_spojeni#id_osoba	data:743830424323704887
< schema:ot_identifikace#id_osoba	data:376049526254447257
< schema:ot_identifikace#id_osoba	data:500167720272250953
< schema:ot_identifikace#id_osoba	data:541700304004441100
< schema:ot_identifikace#id_osoba	data:599506105228534837
< schema:ot_identifikace#id_osoba	data:737638766182060183
< schema:ot_kontakt#id_osoba	data:167028577977341641
< schema:ot_kontakt#id_osoba	data:477480778835730950
< schema:ot_kontakt#id_osoba	data:609951455167012198
< schema:ot_kontakt#id_osoba	data:676065484069261502
< schema:ot_kontakt#id_osoba	data:864029349523301481
< schema:ot_kontakt#id_osoba	data:988846067034253456

1	
< schema:ot_osoba#datum_narozeni_ver	data:846993817175361413
< schema:ot_osoba#handicap_text_ver	data:846993817175361413
< schema:ot_osoba#handicap_typ_ver	data:846993817175361413
< schema:ot_osoba#konz_hodiny_ver	data:846993817175361413
< schema:ot_osoba#kvalifikator_obcanstvi_ver	data:846993817175361413
< schema:ot_osoba#misto_narozeni_ver	data:846993817175361413
< schema:ot_osoba#obcanstvi_ver	data:846993817175361413
< schema:ot_osoba#okres_narozeni_ver	data:846993817175361413
< schema:ot_osoba#pohlavi_ver	data:846993817175361413
< schema:ot_osoba#profesni_zamereni_ver	data:846993817175361413
< schema:ot_osoba#rod_stav_ver	data:846993817175361413
< schema:ot_osoba#rodne_prijmeni_ver	data:846993817175361413
< schema:ot_osoba#stat_narozeni_ver	data:846993817175361413
< schema:ot_osoba#zmen_prac_schopnost_ver	data:846993817175361413
< schema:cht_ciselnik_plochy#kod	data:174747437376312882
< schema:cht_ciselnik_plochy#poradi	data:174747437376312882
< schema:cht_ciselnik_plochy#stav	data:174747437376312882

data:174747437376312882	
> schema:cht_ciselnik_plochy#hodnota	muž
> schema:cht_ciselnik_plochy#id_ciselnik	data:809402076394730530
> schema:cht_ciselnik_plochy#kod	1
> schema:cht_ciselnik_plochy#poradi	1
> schema:cht_ciselnik_plochy#stav	1
> rdf:type	schema:cht_ciselnik_plochy
< schema:ot_osoba#pohlavi	data:846993817175361413

Fig. 4. Example of a two-layered tree

and the number of neighbors is small, the neighbors can be displayed directly in the vertex. If the number is higher, the list of neighbors is displayed in a separate window with the option to filter the displayed entries, which allows handling of even nodes with large number of neighbors.

Node merging is also useful for displaying certain special type of nodes. RDF data usually contain nodes representing certain object with outgoing edges representing its properties, e.g. a person together with his or her name, date of birth, etc. Merged node for the person will contain the name and other information directly so the user can see them without expanding the neighbors. Furthermore a lot of drawing space is conserved since the user will probably be interested in these values and would expand all of the neighbors which may mean adding tens of vertices.

3.2 Animation

When the users expands an edge so that a new node is displayed a drawing of the new tree has to be computed and displayed. To improve the user's experience the transition between the old drawing and the new drawing is animated in real-time. This not only 'looks nice' but more importantly it helps the user maintain connection between objects in the old and the new drawing. Using animation between time-slices to show how nodes and edges are moved to the new positions may also assist in preserving the mental map over time [16].

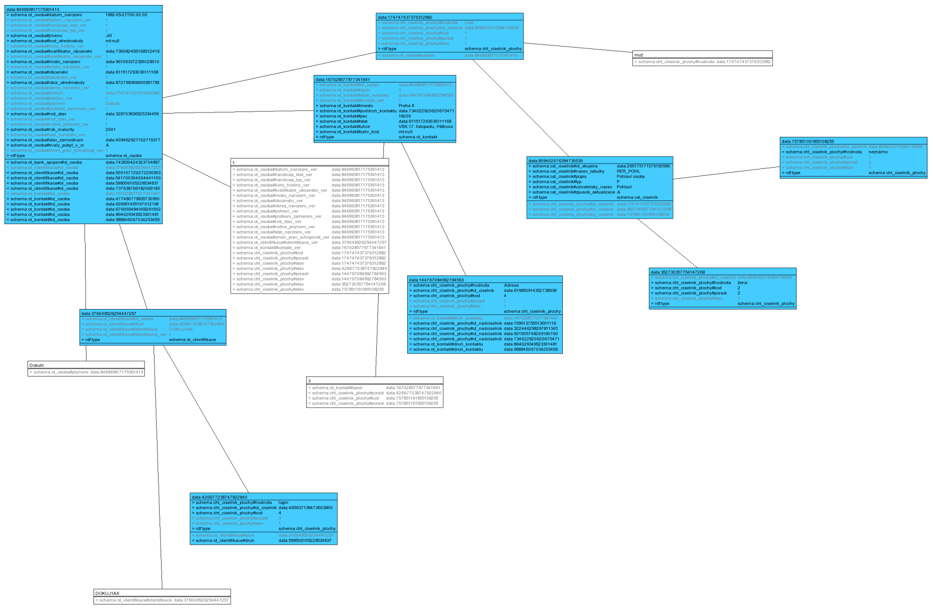


Fig. 5. Example of a large tree

The animation is a simple linear transition of rectangles that represent nodes and lines that represent edges.

3.3 Examples

We have tested the application using the data described in [17]. Figures 4 and 5 are screenshots of some of the visualizations produced by the system.

Darker nodes represent URIs while white nodes are literals. In the list of incoming and outgoing edges, the black can be clicked and new node is displayed, the gray ones represent edges that are already expanded.

4 Conclusion and Future Work

We have designed and implemented a visualization tool to supplement the semantic web infrastructure. The visualization is capable of handling even very large data. We believe it will aid in development of applications that use the infrastructure.

The visualizer could be extended to generate queries based on the displayed data (e.g. by making a query pattern that mirrors the currently displayed graph but literals are converted to query parameters) creating a kind of *query by example* system.

There is still room for improvement in the visualizer itself, especially handling of non-tree edges. Although it is not a significant issue the number of intersections between edges and nodes can be further reduced or even eliminated.

References

1. Dokulil, J., Tykal, J., Yaghob, J., Zavoral, F.: Semantic Web Repository And Interfaces. In: UBICOMM 2007 (includes SEMAPRO 2007), pp. 223–228. IEEE, Los Alamitos, California (2007)
2. Dokulil, J., Tykal, J., Yaghob, J., Zavoral, F.: Semantic Web Infrastructure. In: First IEEE International Conference on Semantic Computing, Los Alamitos, California, pp. 209–215 (2007)
3. Carroll, J.J., Klyne, G.: Resource description framework: Concepts and abstract syntax. W3C Recommendation (2004)
4. Yaghob, J., Zavoral, F.: Semantic web infrastructure using datapile. In: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Itelligent Agent Technology, pp. 630–633. IEEE, Los Alamitos (2006)
5. Herman, I., Melançon, G., Marshall, M.S.: Graph visualization and navigation in information visualization: A survey. IEEE Trans. Vis. Comput. Graph 6(1), 24–43 (2000)
6. Dokulil, J., Katreniaková, J.: Visualization of large schemaless RDF data. In: UBICOMM 2007 (includes SEMAPRO 2007), pp. 243–248. IEEE, Los Alamitos, California (2007)
7. Dokulil, J., Katreniaková, J.: Vizualizácia RDF dát pomocou techniky zlučovania vrcholov. In: Proc. of ITAT 2007: Information Technologies-Applications and Theory, Seňa, Slovakia, PONT, pp. 23–28 (2007)
8. Sayers, C.: Node-centric rdf graph visualization. Technical Report HPL-2004-60, HP Laboratories Palo Alto (April 2004)
9. Pietriga, E.: IsaViz: A Visual Authoring Tool for RDF, `http://www.w3.org/2001/11/Isaviz/`
10. Yee, K.P., Fisher, D., Dhamija, R., Hearst, M.A.: Animated exploration of dynamic graphs with radial layout. In: INFOVIS, pp. 43–50 (2001)
11. Wood, J., Brodlie, K., Walton, J.: gViz - Visualization Middleware for e-Science. In: VIS 2003. Proceedings of the 14th IEEE Visualization 2003, p. 82. IEEE Computer Society, Washington, DC, USA (2003)
12. Deligiannidis, L., Kochut, K.J., Sheth, A.P.: User-Centered Incremental RDF Data Exploration and Visualization. In: ESWC 2007 (submitted, 2006)
13. Goyal, S., Westenthaler, R.: RDF Gravity (RDF Graph Visualization Tool), `http://semweb.salzburgresearch.at/apps/rdf-gravity/user_doc.html`
14. Huang, W., Eades, P.: How people read graphs. In: Hong, S.H. (ed.) APVIS. CRPIT, vol. 45, pp. 51–58. Australian Computer Society, Australia (2005)
15. Reingold, E., Tilford, J.: Tidier Drawings of Trees. IEEE Transactions on Software Engineering SE-7, 223–228 (1981)
16. Purchase, H.C., Hoggan, E., Görg, C.: How Important Is the "Mental Map"? – An Empirical Investigation of a Dynamic Graph Layout Algorithm. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 184–195. Springer, Heidelberg (2007)
17. Dokulil, J.: Transforming Data from DataPile Structure into RDF. In: Proceedings of the Dateso 2006 Workshop, Desna, Czech Republic, pp. 54–62 (2006)