Star-like Auto-Configurable Layouts of Variable Radius for Visualizing and Exploring RDF/S Ontologies

Stamatis Zampetakis[1,2], Yannis Tzitzikas[1,2], Asterios Leonidis[1] and Dimitris Kotzinos[1,3]

[1] *Institute of Computer Science, FORTH-ICS, GREECE,*
[2] *Computer Science Department, University of Crete, GREECE and*
[3] *Department of Geoinformatics and Surveying, TEI of Serres, GREECE*
{zabetak|tzitzik|leonidis|kotzino}@ics.forth.gr

Abstract

The visualization of ontologies is a challenging task especially if they are large. In this paper we propose a visualization approach which is based on star-like graphs of variable radius which enables users to gradually explore and navigate through the entire ontology without overloading them. The star-like graphs are visualized using a Force Directed Placement algorithm (*FDP*) special suited for RDF Schemas whose configuration parameters can be adjusted interactively by the end-user via an intuitive on-screen toolbar. In addition, and since each star-like graph exhibits different graph features, we propose a novel automatic configuration method for the *FDP* algorithm parameters that is based on a number of quality metrics (area density and verticality of subclass hierarchies) and corresponding corrective actions. The experimental evaluation showed the quality of the yielded layout is significantly improved and the proposed approach is acceptably fast for real-time exploration. The user study showed that users prefer these views and perform faster various very common tasks.

Keywords: Force Directed Graph Layout Algorithms, RDFS Ontologies

1. Introduction

Understanding an ontology without the assistance of persons already familiar with it (and its associated applications), is a hard and time consuming task especially if the ontology is quite big in size (containing more than two

dozens of classes). Our objective is to alleviate this problem by providing 2D visualizations that could aid users in tasks like: selection of a suitable ontology from a corpus of ontologies, understanding the structure of one particular ontology, and understanding a number of interrelated ontologies. For a long time now, it has been recognized that the usefulness of conceptual diagrams (e.g. ER/UML/RDF diagrams) degrades rapidly as they grow in size. Some key requirements for the visualization of large in size ontologies are described in [1], including the need for semi-automatic and interactive layout facilities (the user should be able to interact with the automatic layout algorithm) and the need for filtering and complexity reduction techniques (the user should be able to filter out elements and to get diagrams of having the desired size).

In this paper we focus on (a) the support of real-time exploration through star-like graphs of variable radius since this allows users to explore large schemas while controlling the amount of displayed information on the basis of user preferences or screen-size constraints, and (b) the configuration of the force-directed placement algorithms used for rendering the star-like graphs. The latter includes simple and intuitive controls the user can enact, as well as a novel automatic layout improvement method. This approach bypasses the problem of manually configuring these algorithms which is crucial for the problem at hand since the successive star-graphs the user is getting while exploring an ontology exhibit different graph features (number of nodes and edges, kinds of edges) depending on the part of the ontology that is visualized and the selected radius, meaning that each graph requires different configuration for getting an aesthetically pleasing layout. The layout should not be too dense or too sparse and subclass hierarchies should form a top-down drawing. The experimental evaluation showed that the quality of the layout, as it is measured by a number of metrics that we introduce, is improved with the proposed automatic configuration method and the proposed approach is acceptably fast for real-time exploration. The improvements were also verified by a user study. Finally, and since ontologies usually extend and reuse elements from other ontologies, we support a number of options regarding the visualization of multiple (dependent) ontologies. A comparative (with Protégé), task-oriented evaluation showed that overall these features improve task performance which implies that the visualization techniques proposed in this work enhance the readability of the ontology. Our work has been implemented and tested over a system for visualizing RDF schemas,

called StarLion[1].

The paper is organized as follows. Section 2 discusses RDF and the generation of star-like graphs of variable radius as well as the visualization of dependent namespaces. Section 3 describes the layout algorithms and Section 4 introduces the automatic configuration of these algorithms. Section 5 reports the results of the experimental and empirical evaluation. Section 6 discusses related work, and finally Section 7 concludes the paper and identifies issues for further research.

2. Exploration through Star-like Graphs and Dependent Namespaces

2.1. Semantic Web and RDF/S

According to wikipedia [2] and W3C [3] community, the Semantic Web (SW) is an evolving extension of the World Wide Web, in which content can be expressed not only in natural language, but also in languages (e.g. RDF/S) that can be interpreted formally enabling the provision of more advanced searching, sharing and integration services. The term "Semantic Web" is often used more specifically to refer to the formats and technologies that enable it. These technologies include the Resource Description Framework (RDF), a variety of data interchange formats (e.g. RDF/XML, N3, Turtle, N-Triples), and notations such as RDF Schema (RDFS) and the Web Ontology Language (OWL), all of which are intended to provide a formal description of concepts, terms, and relationships within a given knowledge domain.

RDF Schema (variously abbreviated as RDFS, RDF(S), RDF-S, or RDF/S) is a set of classes with certain properties using the RDF extensible knowledge representation language, providing basic elements for the description of ontologies, otherwise called RDF vocabularies, intended to structure RDF resources

Let us now describe more formally RDFS ontologies. We can view an ontology as a directed labelled graph (C, R) where C is the set of classes (plus the predefined classes for literals) and R is a set of binary relationships over C. An edge $e \in R$ can be either a *property* or a *subclassOf* relationship (else noted as *isA* relationship). There is also a labelling function $l : C \cup R \rightarrow String$.

[1]Implemented in Java using the jgraph library, for more see: `http://www.ics.forth.gr/~tzitzik/starlion`

If $c \in C$ then $l(c)$ is the name of the class. If $e \in R$ then $l(e)$ is "isA" if e is a subclassOf relationship, otherwise $l(e)$ is the name of the property e.

2.2. Star-like Graphs of Variable Radius

In this section we focus on star-like graphs of variable radius not only for tackling cluttering situations (like that of Figure 1), but because we realized that almost all graph layouts of ontologies that were prepared manually by members of our group (for almost two decades) for demonstrating purposes were actually star-like graphs.

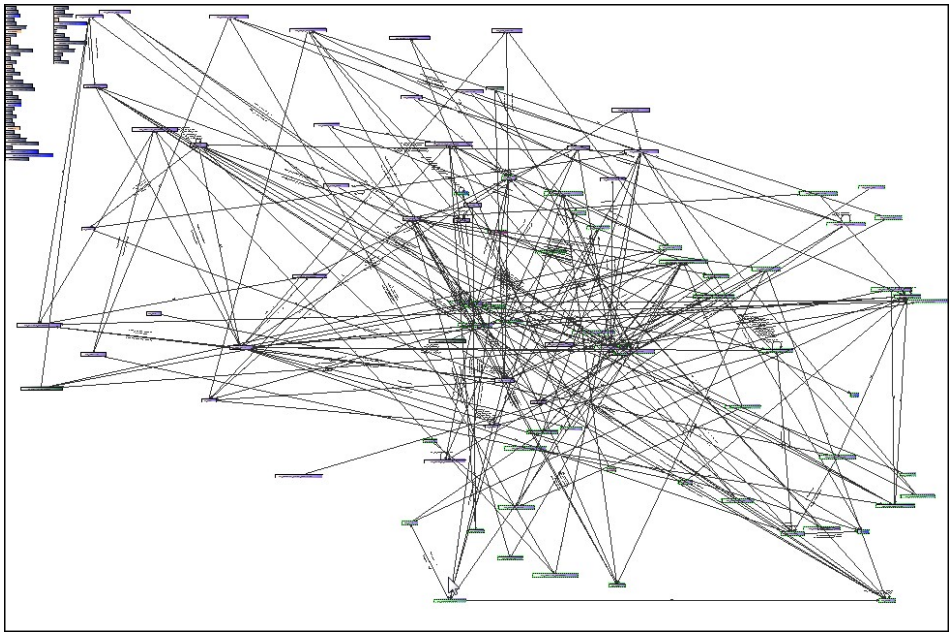

Figure 1: Visualization of Cidoc Digital with dependent namespaces

Let's first define the neighbourhood graphs of a class c. We shall use the notation $c \rightarrow_k c'$ to denote that there is an undirected path that comprises at most k edges that starts from c and ends at c'. We shall also use the notation $e \in c \rightarrow_k$ to denote that e is an edge that belongs to an undirected path starting from c that comprises at most k edges.

Def 2.1. The *plain k-star graph of* c, denoted by $\Gamma(c, k) = (N, E)$ is defined as: $N = \{\ c'\ \in C \mid c \rightarrow_k c'\}$ and $E = \{\ e\ \in R \mid e \in c \rightarrow_k\}$.

Def 2.2. The *extended k-star graph of c*, denoted by $\Gamma_e(c,k) = (N,E)$ is defined as: $N = \{\ c' \ \in C \mid c \rightarrow_k c'\}$ and $E = R_{|N}$, where $R_{|N}$ is the restriction of R on those edges that connect elements that belong to N.

This means that all relationships that connect the visualized nodes are visualized too.

The more useful (from our experience) values for k is 1, 2 and 3. These values result to layouts that are not cluttered (in ordinary screens) and the user can continue browsing by selecting any of the visualized nodes. As an example, Figure 2 (left) shows the star-graph with radius 2 on class `E37_Mark` (of `CIDOC CRM` [4]) and Figure 2 (right) what happens when we click on class `E34_Inscription`. We observe that the layouts are readable and convenient for exploring/understanding gradually the entire schema. Figure 3 shows the star-graph on `E39_Actor` of radius 3. Clearly if the radius is too big, then it's like visualizing the results of applying the (plain graph) reachability algorithm.

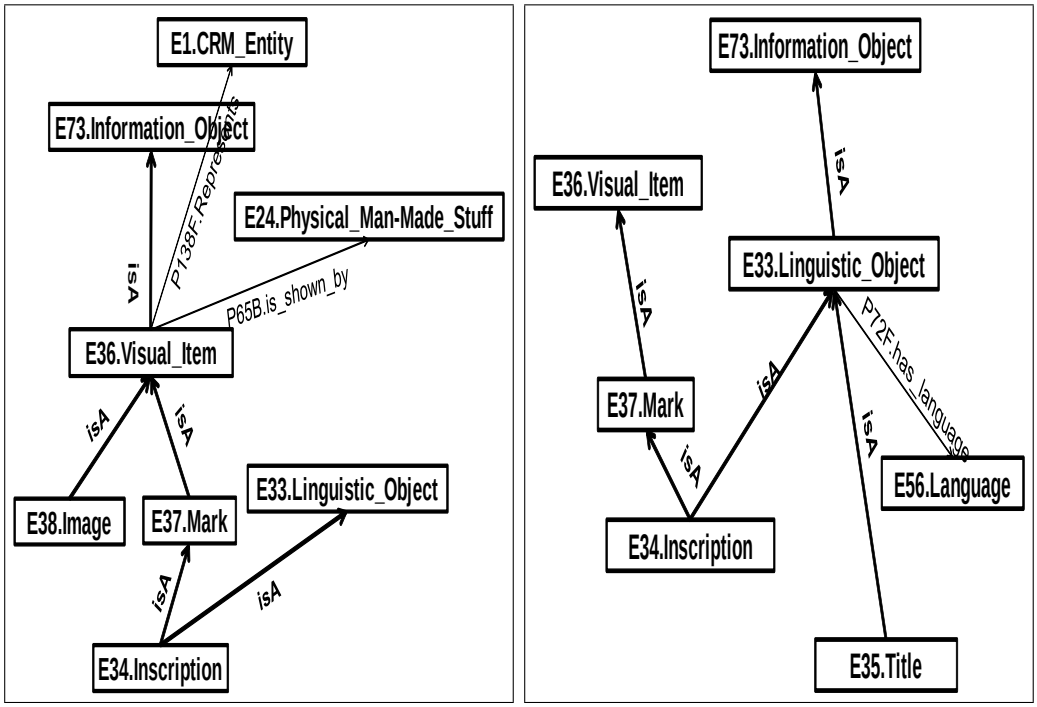

Figure 2: Star-graphs with radius 2 over `CIDOC CRM`

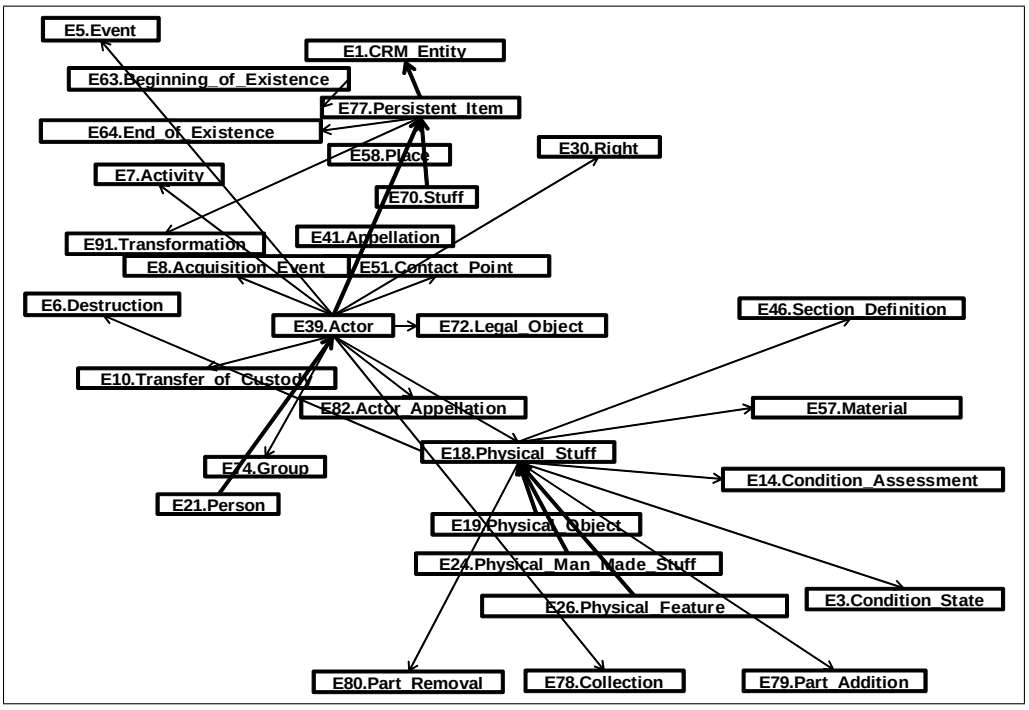

Figure 3: Star-graph on `E39_Actor` with radius 3

2.3. Visualization of Dependent RDF Namespaces

Every schema element (class or property) belongs to a specific namespace. Namespaces, as in normal XML, are used to disambiguate between elements and attributes, and in our case between classes/properties that have the same local name but belong to different schemas. A namespace can extend or reuse elements defined in other(s) namespace(s). It is often useful to load along with a schema the schemata (namespaces) on which it depends. This enhances and completes the understanding by the user for the schema (s)he is interested in, but also multiplies the visualization difficulties (increased number of visualized elements, harder separation of the elements of each schema) and makes it a cumbersome task even for experienced users. To address this problem, we propose a feature where upon loading a specific namespace NS, the user is able to see all namespaces upon which NS depends on, and select those to be visualized, while each namespace's classes are drawn using a different color. Consider the following scenario where the user loads NS1 which depends on NS2 and NS3. StarLion offers the following options:

Transitive Dependencies. Visualize NS1, those elements of NS2 and NS3 that are directly connected to an element at NS1, plus all broader elements (superclasses and superproperties) of the latter. An example is shown in Figure 4(a). Notice that such views contain all the information for understanding NS1. The aforementioned approach is considered to be the established methodology, between expert ontology developers, when manually building ontologies in various drawing tools such as Powerpoint.

Direct Dependencies. Visualize NS1 and those elements of NS2 and NS3 that are directly connected to an element in NS1 as shown in Figure 4(b).

Full Namespaces. Visualize NS1, NS2, NS3 namespaces entirely as shown in Figure 4(c).

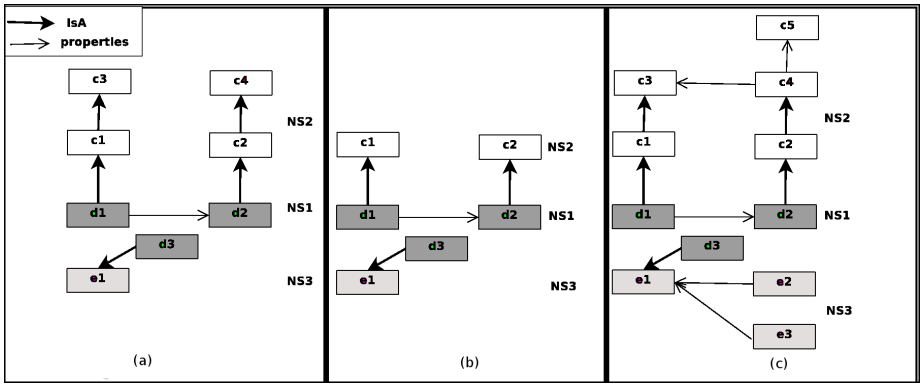

Figure 4: Visualization options for three dependent namespaces

For instance, the ontology of `CIDOC CRM` contains 78 classes, 243 properties and 7 attributes (properties pointing to literal value types), while `CRM Digital` [5] is an extension of that ontology for digital objects (consisting of six new classes and a dozen of new properties). Figure 5(a) shows the "Full Namespaces" option, while 5(b) the "Direct Dependencies" option. The latter allows someone who is already familiar with `CIDOC CRM` to understand very quickly how the six new classes (green colored) of `CRM Digital` extend the `CIDOC CRM` classes (yellow colored). The left diagram, although it provides the complete picture, it does not aid understanding and thus it is beneficial to be explored using the methods that we describe next.

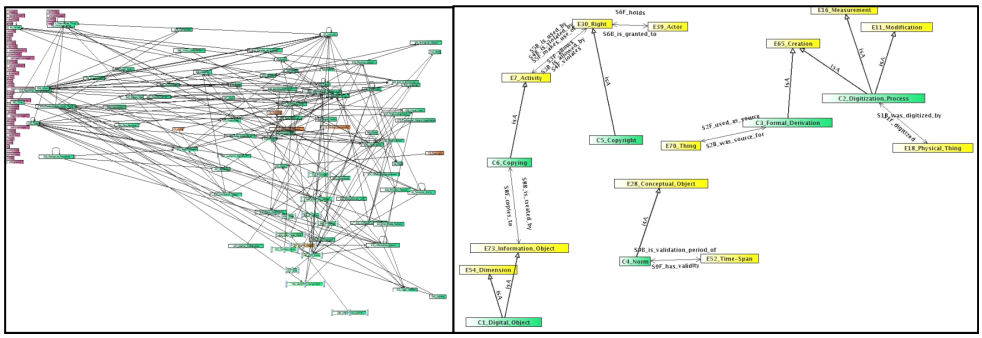

Figure 5: (a) Full, (b) Direct namespaces

3. Layout Algorithms

In this section we describe the Force Directed Placement (FDP) algorithm adopted (Section 3.1) and then the manual configuration of the layout algorithm (Section 3.2). The automatic configuration method is described in Section 4.

3.1. FDP Algorithms for RDF graphs

For deriving automatically the 2D layout we view the graphs as mechanical systems. We adopt the force model that was proposed in [6] for visualizing E-R diagrams. Specifically, that model combines the *spring-model* (proposed and developed in [7, 8, 9]) with the *magnetic-spring model* (proposed in [10, 11]). In our case we apply them on RDF/S graphs.

Nodes (in our case classes) are viewed as equally charged particles which *repel* each other. Edges (i.e. RDF/S properties and *isA* relationships) are viewed as springs that *pull* their adjacent nodes. Moreover, we assume that the springs that correspond to *isA* links are all magnetized and that there is a global magnetic field that acts on these springs. Specifically, this magnetic field is parallel (i.e. all magnetic forces operate in the same direction) and the *isA* springs are magnetized unidirectionally, so they tend to align with the direction of the magnetic field, here upwards. This is because the classical way (in semantic web, object-oriented class diagrams, Formal Concept Analysis, Hasse diagrams in discrete mathematics, etc) of presenting specialization/generalization relationships is to put the superclass above the subclass. Figure 6 illustrates this metaphor for RDF schemas, while Table 1 shows how each force is defined. The algorithm, at each iteration, computes the force on each node and then moves the node towards the corresponding

direction by a small amount proportional to the magnitude of the force. This can be continued until convergence, but in practice we limit the number of iterations to 100, as experiments showed that more iterations barely change the layout (however users are free to change the number of iterations if they wish to). Details about the force model are given next.

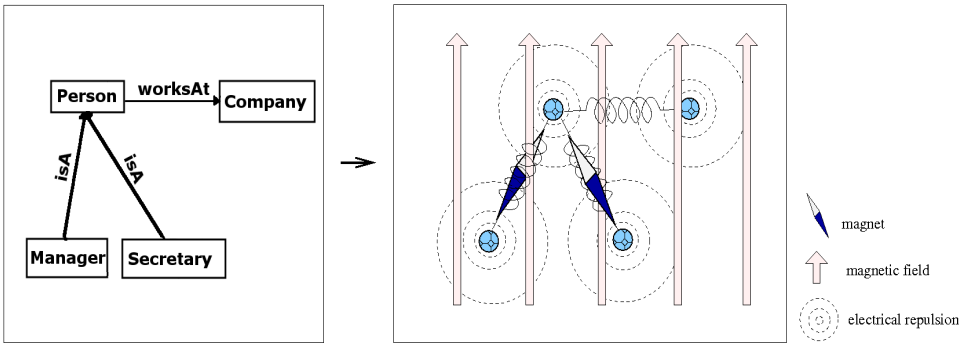

Figure 6: An RDF schema as a mechanical system

Force	Definition
Spring	$f_x(c_i) = \sum_{c_j \in conn(c_i)} K_s(d(p_i,p_j) - L)\frac{x_j - x_i}{d(p_i,p_j)}$
Repulsion	$g_x(c_i) = \sum_{c_j \in N, c_j \neq c_i} \frac{K_e}{d(p_i,p_j)^2}\frac{x_i - x_j}{d(p_i,p_j)}$
Magnetic	$h_x(c_i) = \sum_{c_j \in conn_{sp}(c_i)} K_m \frac{x_j - x_i}{L} + \sum_{c_j \in conn_{sb}(c_i)} K_m \frac{x_j - x_i}{L}$ $h_y(c_i) = \sum_{c_j \in conn_{sp}(c_i)} K_m \frac{L + y_j - y_i}{L} - \sum_{c_j \in conn_{sb}(c_i)} K_m \frac{L + y_i - y_j}{L}$
Composed	$F(c_i) = f(c_i) + g(c_i) + h(c_i)$

Table 1: Force Model (only x component is shown)

Specifically, and if (N, E) is the visualized graph, then the force on a node c_i is given by $F(c_i)$ as defined in Table 1. In the formulas of that table, if x is a node, $conn(x)$ is the set of nodes directly connected with x, while $conn_{sp}(x)$ denotes the direct superclasses of x, and $conn_{sb}(x)$ denotes the direct subclasses of x. The force $f(c_i)$ is the power exerted on c_i by the springs between c_i and $conn(c_i)$, $g(c_i)$ is the electrical repulsion exerted on c_i by all other nodes, and $h(c_i)$ is the rotational force exerted on c_i by the nodes $conn_{sp}(c_i) \cup conn_{sb}(c_i)$.

The **spring force** $f(c_i)$ follows Hooke's law, i.e. it is proportional to the difference in distance between nodes and the zero-energy length of the spring. Let $d(p, p')$ denote the Euclidean distance between two points p and

p' and let $p_i = (x_i, y_i)$ denote the position of a node c_i. The x component of the force $f(c_i)$ is given in the first row of Table 1, where L denotes the natural (zero energy) *length* of the springs. This means that if $d(p_i, p_j) = L$ then no force is exerted by the spring between c_i and c_j. Now K_s denotes the *stiffness* of the springs. The larger the value of K_s, the more tendency for the distance $d(p_i, p_j)$ to be close to L. The y component of the force $f(c_i)$ is defined analogously. The **electrical force** $g(c_i)$ follows an inverse square law. The x component of the force $g(c_i)$ is given in the second row of Table 1 where K_e is used to control the *repulsion strength* between nodes. The y component of the force $g(c_i)$ is defined analogously. The **magnetic force** $h(c_i)$ depends on the angle between the *isA* spring (that connects the nodes) and the direction of the magnetic field and it induces a rotational force on that spring. The x and y components of the magnetic force $h(c_i)$ are given in the third row of Table 1, where K_m is used to control the strength of the magnetic field. The x and y components of the **composed force** $F(c_i)$ on a node c_i are obtained by summing up, i.e., $F_x(c_i) = f_x(c_i) + g_x(c_i) + h_x(c_i)$ and $F_y(c_i) = f_y(c_i) + g_y(c_i) + h_y(c_i)$.

3.2. User Configuration of the FDP Params

The *FDP* parameters K_e, K_m, L can be configured by the user through a form which is displayed when the user requests the application of the FDP algorithm. However we realized that casual users had difficulties in changing these values because they were not familiar with the internals of the *FDP* algorithm. For this reason we introduced a toolbar (as shown in Figure 7) which lets the user to increase/decrease (a) the verticality of *isA* hierarchies, (b) node repulsion, and (c) spring length. Table 2 shows what happens after each action. We decided to use relative buttons, instead of sliders, because with a slider a user might set a value that would result in a big change of the layout what would perplex the user and would not allow him to understand the effects of each variable. Sliders by definition are state dependent, however in our case there are no states: if we restore an old value of a parameter the layout is not necessarily the same, so sliders would be misleading. Users responded positively to this change. To further improve the layout and reduce (or even vanish) the number of clicks on the above buttons, below we describe a novel method that we have developed for configuring automatically the FDP parameters.

Aspect	After pressing the **Increase**	After pressing the **Decrease**
Verticality of *isA*	$K_m = K_m * 2$	$K_m = K_m/2$
Node Repulsion	$K_e = K_e * 2$	$K_e = K_e/2$
Spring Length	$L = L + 10$	$L = L - 10$

Table 2: Relative Configuration of the Layout Parameters

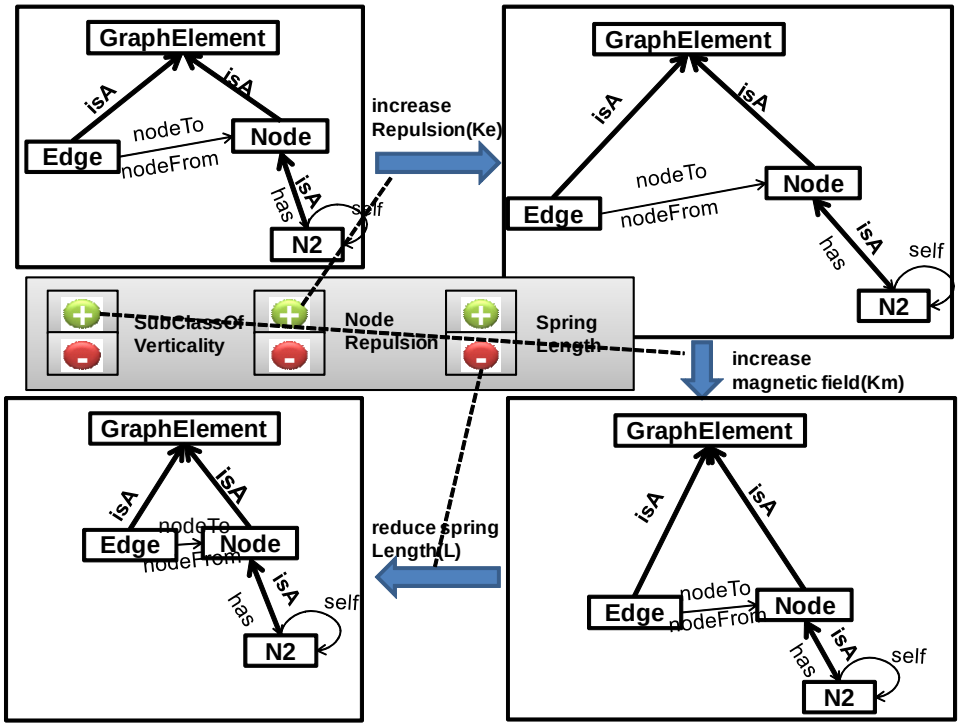

Figure 7: Relative Configuration of the Layout Parameters by a User-friendly Toolbar

4. Automatic Configuration of the Layout Parameters

We have devised, an automatic configuration mechanism which utilizes a set of quality metrics on the produced layout in order to reconfigure algorithm's parameters and improve the final layout. These metrics and the reconfiguration of the parameters are exploited in two ways:

(a) the *FDP* algorithm is applied once, the quality of the layout is measured, the parameters are reconfigured accordingly, the *FDP* is reapplied, and the resulting layout is displayed.

(b) a "Magic" button is available for starting the quality measurement and layout improvement process on demand.

Below we introduce the two quality metrics and the correcting/improvement actions.

4.1. Verticality of Specialization Hierarchies

As we discussed in section 3.1 users are expecting superclasses to be above subclasses in every occasion. The magnetic field was introduced for this reason but without any configuration the *FDP* algorithm may produce layouts without this orientation.

We can measure the quality of the layout of the *isA* relationships (i.e of the elements of R_{sub}) through a metric that ranges [-1,1], which is defined as follows:

$$Verticality(R_{sub}) = \frac{\sum_{(a,b)\in R_{sub}} \frac{y_b - y_a}{L_{a,b}}}{|R_{sub}|} \quad (1)$$

where $L_{a,b}$ denotes the length of the edge (a, b) in the current layout. For reasons of brevity we shall use V to denote $Verticality(R_{sub})$. If $R_{sub} = \emptyset$ we assume that $V = 0$. Notice that if all *isA* edges are vertical and have the desired direction, then we get a value equal to 1. If the edges have the opposite direction then we get a value close to -1. If the verticality of R_{sub} is low then we could improve the layout by strengthening the magnetic field. We will analyse this in more detail later on.

However the above metric does not take into account the morphology of R_{sub}. For instance, Figure 8 shows five graphs each having five nodes. We can observe that an "ideal" w.r.t. verticality layout should not necessarily have $V = 1$, e.g. see the rightmost diagrams.

For this reason we introduce a factor called IV (where IV comes from IdealVerticality) which aims at expressing the desired verticality value of a layout given its morphology.

The desired verticality for the layout depends on what users consider as ideal layout. Even in the simple case where the graph consists of only one node and its subclasses, the decision about the best layout is not obvious (see Figure 9). The most common layout for a node and its subclasses used from the majority of graph drawing algorithms and graph related applications is shown in Figure 9(a) and it is the one that we adopted as ideal position of nodes.

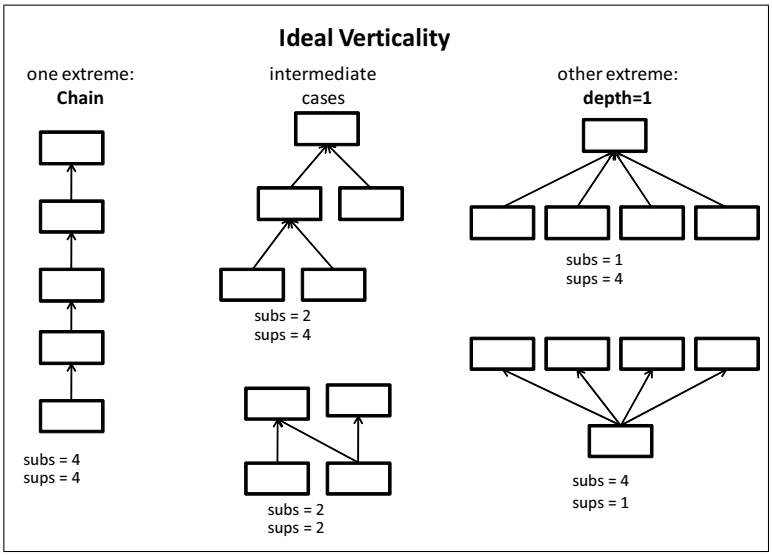

Figure 8: Ideal Verticality

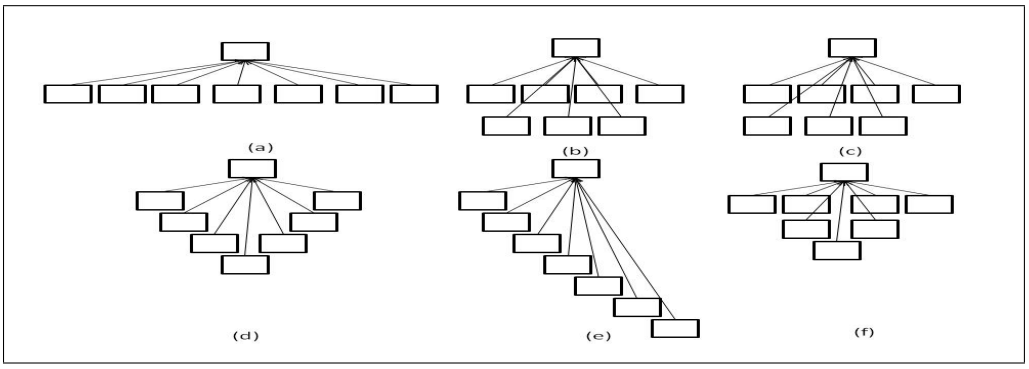

Figure 9: Possible Layouts for a Class and its Subclasses

First we will define IV for a single class c and then extend the definition for the entire graph ($\forall c \in N$). Figure 10 presents how verticality is changing by adding subclasses. The nodes along with their subclasses can be represented by equilateral triangles with side a, height h and base $b = 2 * b_{width}$ where b_{width} is the average width of the boxes which are used for visualizing nodes. Every time we add a new subclass, the base of the triangle grows, specifically the base of the triangle is an arithmetic progression starting from b and having step equal to $b/2$. The side a of the equilateral triangle that

presents a class with n subclasses is given below

$$a_{n-1} = \sqrt{h^2 + (\frac{b + (n-1)\frac{b}{2}}{2})^2}$$

According to the ideal positioning that we chose (Figure 9(a)) we define Minimum Verticality (MV) that is desired for a class c and its subclasses, as

$$MV(c) = \frac{h}{a_{|conn_{sb}(c)|-1}}$$

which comes directly from the definition of verticality, where h is equal to L (the length of the springs). We use the word "minimum" because this formula is based on the layout of Fig.9(a), and as we can see all other possible layouts (Fig.9(b)-(f)) have greater verticality than that of Fig. 9(a), assuming that all edges have length at least equal to L. It is therefore reasonable to say that Verticality should be between MV and 1, so we define $IV(c)$ as the average of them:

$$IV(c) = \frac{MV(c) + 1}{2}$$

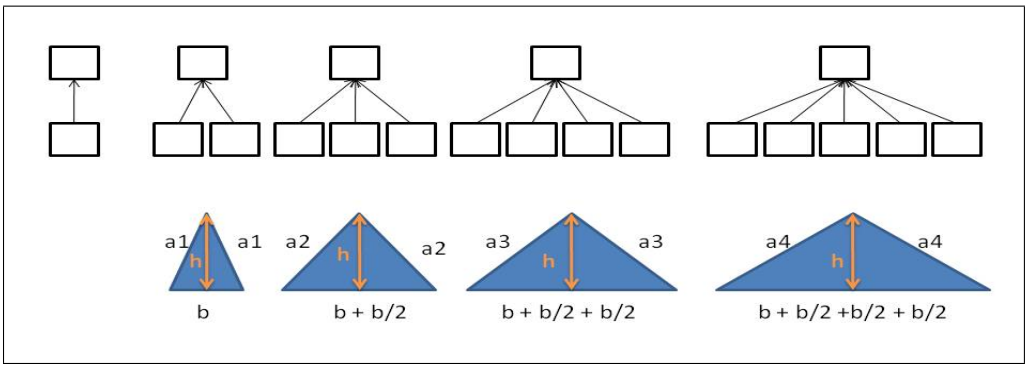

Figure 10: A Node with its Subclasses as Equilateral Triangles

A class apart from subclasses it can also have superclasses. To capture such cases we have to modify the definition of MV. The MV for a class c with both subclasses and superclasses is determined from the *maximum* of $|conn_{sb}(c)|$ and $|conn_{sp}(c)|$. This is shown in Figure 11 where verticality of sub-graph containing the nodes $\{c\} \cup conn_{sb}(c)$ is greater than the verticality of the sub-graph containing the nodes $\{c\} \cup conn_{sp}(c)$. Specifically, we define:

$$MV(c) = \frac{h}{a_{\max(|conn_{sb}(c)|,|conn_{sp}(c)|)-1}}$$

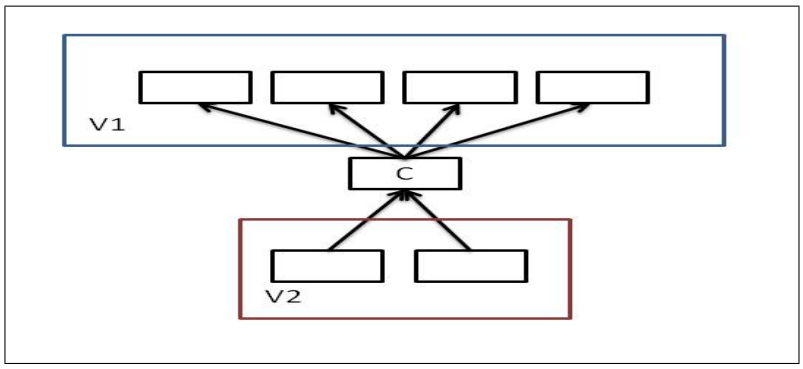

Figure 11: A class C with 4 superclasses and 2 subclasses positioned ideally

We can now define the Ideal Verticality for the entire graph. The only difference is that we must find the *minimum* verticality for every class ($\forall c \in N$). Specifically, if $MaxSubs = \max\{\ |conn_{sb}(c)|\ |\ c \in N\}$ and $MaxSups = \max\{\ |conn_{sp}(c)|\ |\ c \in N\}$, then

$$MV = \frac{h}{a_{\max(MaxSubs, MaxSups)-1}}$$

Finally, we define IV as the average of MV and 1, i.e $IV = \frac{MV+1}{2}$.

To evaluate the accuracy of the metric we created some test schemas (capturing frequently occurring morphologies) and we manually positioned their nodes according to the ideal layout. Then we measured both verticality (V) and ideal verticality (IV) for those graphs. The closest the difference $V - IV$ is to zero the better the metric is because we assumed that the manual position of the nodes is the ideal so V and IV must be almost identical for these graphs. Figure 12 shows the created layouts and Table 3 the values of V and IV for these layouts.

Layout	V	IV	$V - IV$
Figure 12(a)	0.943	0.958	-0.015
Figure 12(b)	0.990	0.996	-0.006
Figure 12(c)	0.964	0.951	0,013
Figure 12(d)	0.958	0.947	0,011
Figure 12(e)	0.827	0.858	-0,031
Figure 12(f)	0.908	0.899	0,009

Table 3: Verticality and Ideal Verticality for Test Layouts

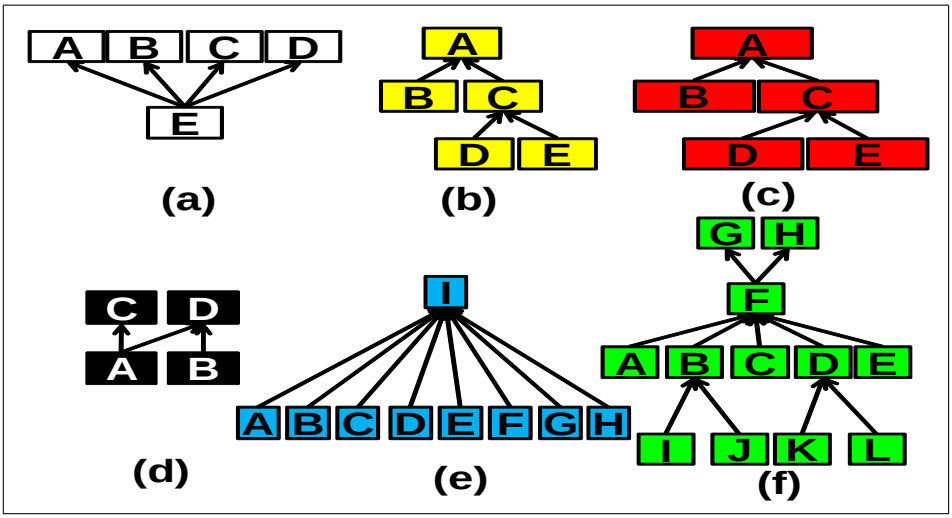

Figure 12: Manually created Layouts for measuring Verticality and IdealVerticality

4.2. Layout Area

Now we will introduce a metric for identifying whether a layout is sparse or dense, based on the area it occupies and the number of nodes and edges of the depicted graph. Let $minX$ and $maxX$ denote the minimum and maximum X coordinates of the layout, and let $minY$ and $maxY$ the minimum and maximum Y coordinates. It follows that the area occupied by the layout is:

$$Area = (maxX - minX) * (maxY - minY)$$

Let $BoxArea$ denote the average area that is occupied by a node, i.e. $BoxArea = b_{width} * b_{height}$ where b_{width} and b_{height} is the average width and height of the boxes which are used for visualizing nodes.

Clearly, the minimum space required for visualizing only the nodes of the graph in a non overlapping way, is $N * BoxArea$ (assuming that no free space exists between node boxes). However our graphs have edges which are labelled with one or more strings. Suppose for the moment that each edge is associated with one string. We shall use $|R|$ to denote number of *visible edges*, i.e. if there are several properties that connect the same pair of classes, they contribute 1 to $|R|$ because they are visualized as a single line segment which has multiple labels. The minimum space required for visualizing the nodes and the labels of the edges in a non overlapping way is:

$$MinArea = |N| * BoxArea + |R| * L * b_{height}$$

where L is the spring length and we assume that the width of the edge labels is less than the length of the spring.

Since users prefer less dense diagrams, we consider as "ideal" area a multiplication of the minimal area, i.e.:

$$IdealArea \;\; = \;\; 5 * MinArea$$

Note that the maximum area is unlimited since springs are elastic.

4.3. Corrective Actions

Table 4 shows the actual parameter values before any improvement action.

Parameter	**Value**
Initial K_m	50
Initial K_e	500000
Natural Length of Spring L	150
FDPA Iterations	100
Base B for K_m improvement	250
Base B for K_e improvement	400

Table 4: Actual Parameter Values

Improving Verticallity.

Recall that V ranges [-1,1] where 1 is the optimal value. However for the layout at hand the desired value is IV which ranges [0,1]. We can improve the verticality of *isA* relationships by changing the strength of the magnetic field K_m, aiming at getting a V closer to IV. Specifically we propose updating K_m as follows:

$$K_m = K_m * B^{IV-V}$$

where the base B in our setting is 250. This value works well along with the other parameters shown in Table 4.

Improving Area.

Having measured the occupied space of the layout, i.e *Area*, and having computed the ideal area based on the morphology of the graph, i.e $IdealArea$, we now define the metric:

$$A = \frac{IdealArea - Area}{\max(IdealArea, Area)}$$

IV	V	IV-V	Resulting Action
1	-1	2	$K_m = K_m * B^2$
1	0	1	$K_m = K_m * B$
1	1	0	$K_m = K_m$
0.5	0.5	0	$K_m = K_m$
0.5	0	0.5	$K_m = K_m * \sqrt{B}$
0.5	1	-0.5	$K_m = K_m * \frac{1}{\sqrt{B}}$

Table 5: V, IV values and the resulting actions

Obviously, A ranges [-1, 1] and we get $A = 0$ if the area has the ideal size, $A > 0$ if the layout is dense, and $A < 0$ if the layout is sparse.

We can improve the size of the layout by changing the strength of the node repulsion (K_e) on the basis of the value A. Specifically:

$$K_e = K_e * B^A$$

where the base B in our setting is 400 as shown in Table 4.

Improving both Area and Verticality.

We have observed that whenever the repulsion strength increases, we have to increase also the magnetic strength if we want to retain the same verticality level. In general, we have noticed that the values of K_m and K_e should not be treated independently. Therefore, one approach to improve both aspects of a layout is the following: first revise K_e, then compute and derive the new layout, then measure again V, then revise K_m, and finally draw again the diagram. However this process requires applying the *FDP* algorithm twice. A remedy for applying the *FDP* algorithm only once is to update both K_m and K_e in one shot but using different B values for the correction. This is the reason why we use $B = 250$ for K_m, and $B = 400$ for K_e.

Now we propose an alternative approach aiming at improving both aspects more accurately by a single application of the *FDP* algorithm. At first, the plot in Figure 13 shows how we can retain the same verticality with different combinations of K_m and K_e. This plot was derived as follows. Over a small schema we were varying the values of K_m and K_e and we were keeping only those combinations that lead to a layout of the test schema that has the same verticality. The plot can be approximated with the equation of a line of

the form $y = ax + b$ or specifically with $y = 17000x - 350000$. The proposed method exploits this correlation, and consists of the following steps:

Alg. 1

1. $\Delta_{K_m} = ImproveVerticality(K_m) - K_m$
   The $ImproveVerticality(K_m)$ revises K_m based on V and IV as we described earlier (i.e. $K'_m = K_m * B^{IV-V}$). From the revised value we substract K_m to compute Δ_{K_m} which can be positive or negative.
2. $K'_e = ImproveArea(K_e)$
   This step revises K_e to improve area as we described earlier i.e. $K'_e = K_e * B^A$.
3. $sK_m = (K'_e + 350000)/17000$
   This step computes the K_m that keeps verticality stable for the specific K'_e.
4. $K'_m = sK_m + \Delta_{K_m}$
   This step revises again K_m based on the results of step 1.
5. Apply *FDP* algorithm with K'_e and K'_m

The results of the measurements after the above improvements are reported in Section 5.2.

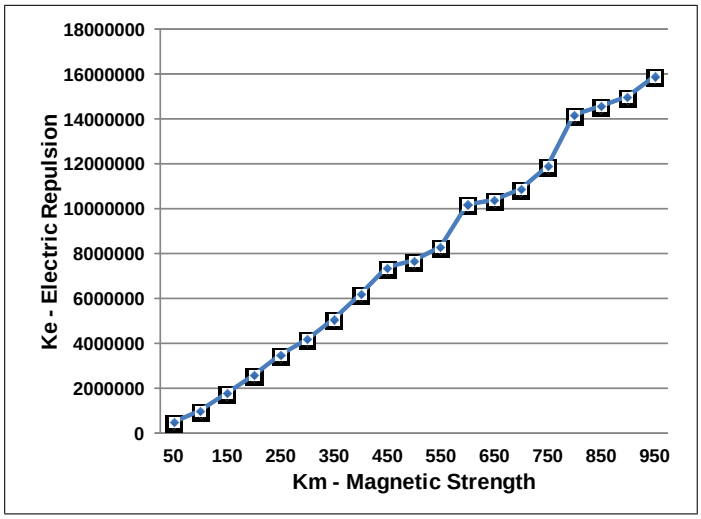

Figure 13: Different combinations of K_m and K_e with the same verticality

5. Evaluation

This section reports experimental results regarding the efficiency and the effectiveness of the auto-configuration method as well as the accuracy of the quality metrics. Furthermore, it contains the results of a comparative user-based evaluation with Protégé.

All experiments were conducted in StarLion. StarLion can load ontologies expressed in ".rdfs" files or stored remotely at the SWKM[2], and offers both textual and graphical views. It supports a semi-automatic layout process (where the user can change node positions, nail down nodes, apply layout algorithms, etc), and the Top-k diagrams (proposed in [12]), for aiding the process of understanding large in size ontologies. In addition, it supports multiple windows, edge label visibility options and scaling.

5.1. Processing Time

Performance is crucial for on-line visualization. We conducted a number of experiments over several ontologies. The experiments were conducted using an ordinary PC[3]. Table 6 shows the results. $|N|$ is the number of nodes contained in the visualized (part of the) schema, and $|E|$ is the corresponding number of edges. The column *FDP* shows the times (measured in ms) needed for applying the *FDP*, while the column "metric" reports the times for calculating the layout quality metrics. We can see that the proposed approach is fast enough for real-time browsing. Although star graphs should not be very big (if we want them to remain readable by the user), we have included in the table one row that corresponds to the entire ontology of `CIDOC CRM` (it is actually the extended star-graph of the class `E39_Actor` with radius twelve). We can see that even for this scenario, which in normal circumstances does not appear, the time for calculating the new positions of nodes and the calculation of metrics remains very low (less than 1 second).

5.2. Quality Metrics and Automatic Configuration

In order to evaluate the effectiveness of the automatic configuration based on the quality metrics we selected an initial layout and applied some correcting actions. The values of the metrics for various actions over a test schema that resembles a star graph of radius 2 are shown in Figure 14.

[2]FORTH-ICS Semantic Web Knowledge Middleware, http://athena.ics.forth.gr:9090/SWKM.

[3]Pentium IV 3.4GHz, 2GB Memory with Java 1.6.0_07 installed.

Schema			Execution Times (ms)	
			Appl. of FDP	Comput. of Metrics
Name	$\|N\|$	$\|E\|$		
Event(R=1)	7	6	5	0.1471
Event(R=2)	36	35	60	0.3025
Event(R=3)	61	60	169	0.3020
Actor(R=1)	15	14	10	0.0643
Actor(R=2)	48	47	101	0.1947
Actor(R=3)	66	65	206	0.2671
Actor(R=12)*	80	184	327	3.6844

Table 6: Execution times for ontology visualization

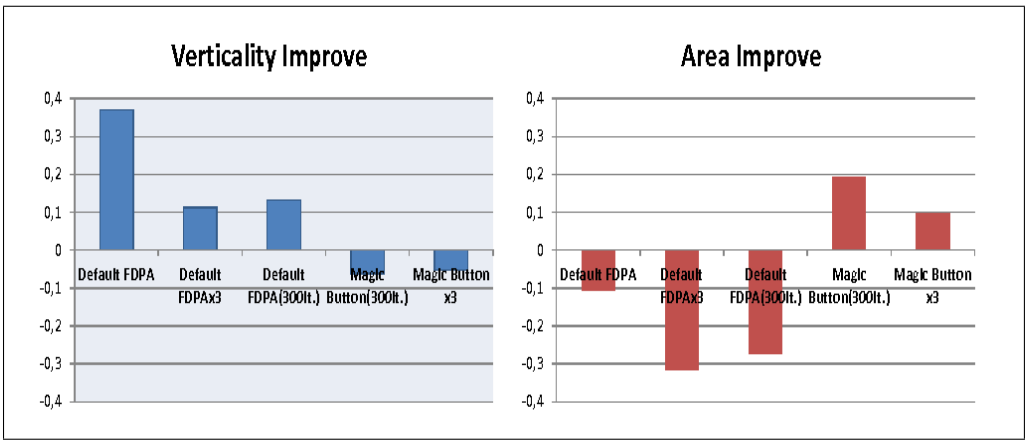

Figure 14: Quality Metrics after different Correcting Actions

The closer these values are to 0 the better the corresponding layouts are. The correcting actions that were applied are the following: (a) one application of the *FDPA* without any configuration (lasting 100 iterations), (b) three consecutive applications of the *FDPA* without any configuration (lasting 100 iterations each), (c) one application of the *FDPA* without any configuration (lasting 300 iterations), (d) three presses of the "Magic" button (*FDPA* + 100 iterations) and (e) one press of the "Magic" button (*FDPA* + 300 iterations). The results after (b) and (c) are almost identical which was expected since both results yield 300 iterations in total. By applying the correcting actions (d) and (e) the different number of iterations affects the layout. By pressing the magic button 3 times we measure and correct the layout 3 times (one every 100 iterations) in contrast with (e) where we measure the quality of the layout only at the beginning and then we run the correcting algorithm more time (300 iterations). The provision of feedback to the algorithm (about

the quality of the layout) allows reconfiguring its parameters progressively and this yields better results and smoother transitions between presses of the magic button. By observing the diagrams we can see that the automatic configuration (i.e. the magic button functionality) provides the best results (values closer to 0).

We should stress that the two metrics are not independent and an improvement of verticality might affect negatively the area (and the vice versa). In Figure 15 we present the two metrics plotted together and this demonstrates the overall improvement of the layout. Although the area improvement after 3 presses of the magic button seems to be the same with a single execution of the *FDPA*, the verticality is not good after a single execution of the *FDPA*. We can observe that with three presses of the magic button the quality of the layout (by considering both area and verticality) is better from the previous ones.

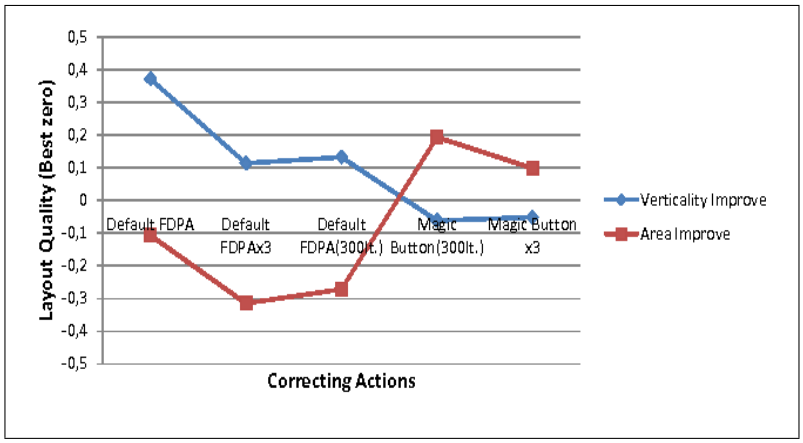

Figure 15: Overall layout improvement after different correcting actions

Figures 16 and 17 show the values of the layout quality metrics for some schemas (specifically for the star-graphs of the classes `E5_Event` and `E39_Actor` for radius 1..3), and the values after the correcting actions. The numbers inside the parentheses reveal how many times we applied the correcting action (auto-configuration of the parameters and *FDPA* application). VI stands for Verticality Improvement and AI for Area Improvement. We can see that the quality of the layout is improved after changing the parameters and reapplying the algorithm, as the quality metrics improve (i.e. get closer to 0). We observe that both verticality and area tend to 0 after consecutive

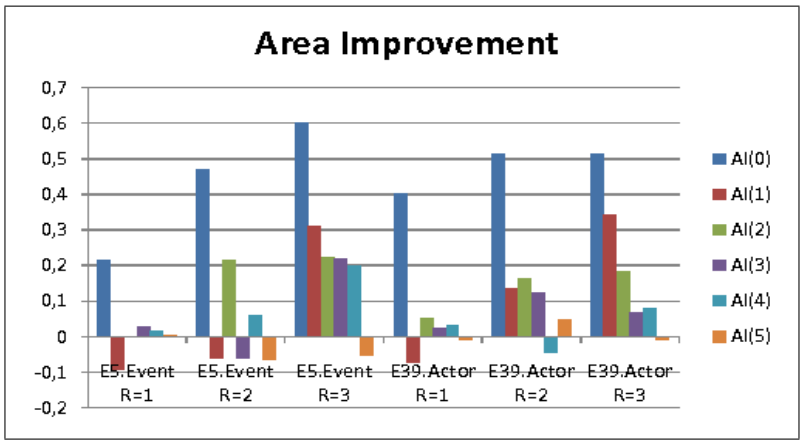

Figure 16: Area improvement after automatic configuration

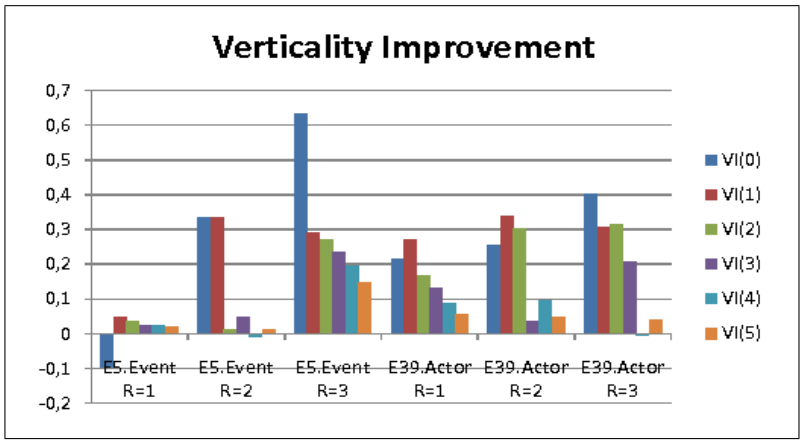

Figure 17: Verticality improvement after automatic configuration

presses of the magic button.

5.3. User Evaluation of the Auto-Configuration Method

The results of Section 5.2 prove that the proposed method improves the layout as regards the adopted quality metrics. To verify the improvement in real world situations with real users, we conducted a user study. Specifically we derived the layouts for 15 graphs. They were star graphs of various nodes of `CIDOC CRM` with radius 1, 2 or 3. For each graph we derived two diagrams: one using the *FDP* and one with *FDP+* Autoconfiguration (AC) and we printed them on paper (on the same page). To ensure that users were

unbiased, we did not write anything on page regarding the applied actions, and we randomly decided the position of diagrams on the page. Consequently we asked from 20 persons to express their preference (*I prefer (a), I prefer (b), Both are the same to me*) for every one of the 15 graphs (=30 graph views). Table 7 summarizes the results

Diagram with radius (R) = X	***FDP***	***FDP+*** **Autocon-figuration**	**Same**
E38.Image with R = 1	10%	75%	15%
E38.Image with R = 2	60%	5%	35%
E38.Image with R = 3	20%	80%	0%
E34.Inscription with R = 1	0%	15%	85%
E34.Inscription with R = 2	45%	15%	40%
E34.Inscription with R = 3	55%	45%	0%
E5.Event with R = 1	45%	5%	50%
E5.Event with R = 2	5%	90%	5%
E5.Event with R = 3	0%	100%	0%
E6.Destruction with R = 1	5%	95%	0%
E6.Destruction with R = 2	55%	25%	20%
E6.Destruction with R = 3	0%	100%	0%
E37.Mark R = 1	10%	10%	80%
E37.Mark R = 2	90%	5%	5%
E37.Mark R = 3	0%	85%	15%
Average	27%	50%	23%

Table 7: The Percentage of users that chose each layout

As depicted in the results, in almost 50% of the cases, users considered the graph generated by the Force-Directed & Autoconfiguration tool as a better choice, while in 30% of the cases the Plain Force-Directed graph was selected instead. Finally, only in the rest 20% of the cases users show no special preferences between both resulted graphs.

Nonetheless, eliminating graphs with radius either 1 or 2, which contain very few nodes, the aforementioned results are significantly improved. In 80% of the cases the graphs generated by the Force-Directed & Autoconfiguration tool were considered as a better choice, while only in 20% of the cases the Plain Force-Directed graph was eventually selected.

5.4. User Evaluation: StarLion vs Protégé

In this section we compare StarLion with Protégé in order to prove the usefulness of star-graphs, dependent namespaces and the proposed layout algorithms. The evaluation is *task-oriented* and emphasizes on the best possible understanding of the ontology from the user. Better time in completing a

task implies better understanding of the ontology (i.e efficient visualization). Protégé is one of the main representatives in the area of ontology management and supports a large range of visualization options. In this evaluation Jambalaya plug-in was used. It is important to note that we are not comparing these tools in general but only their visualization facilities. For the evaluation, we selected 10 users and asked them to perform the following tasks, over the ontology `CRM Digital`:

(T1) Find all direct superclasses of the class "Digitization Process"

(T2) Find all direct properties related with the class "Formal Derivation"

(T3) Find all superclasses of the class "Norm"

(T4) Find all properties (directed or inherited) related with the class "Digital Object"

(T5) Find all classes which are related (through subclass or property relationships) with the class "Copying" and for each of these classes find their direct superclasses.

(T6) Prepare a layout with the neighborhood of the class "Norm" in order to describe the derivation of the class.

The tasks chosen correspond to very common actions a typical user usually executes in the phase of understanding and exploring a schema.

5.4.1. User Selection

A total of ten users with no previous knowledge of the schema were selected. The first four users had not any experience regarding ontologies (however they had a minimum experience with conceptual modelling and E-R diagrams) and they were totally new to the tools that were used for completing the tasks. The reason for this selection was to see how users with no experience at all interact with our tool and how user friendly it is in comparison with Protégé. The remaining six were familiar with ontologies in general and two of them had already used StarLion and Protégé in the past.

5.4.2. Evaluation Procedure

The users were asked to use StarLion and Protégé's Jambalaya plugin to complete the above tasks. At the beginning we trained each user for 30 minutes (for both systems). To be sure that our results are valid, we asked the users to complete the tasks, first in StarLion and then in Protégé. By doing this we gave a small advantage to Protégé since the users became familiar with the schema and they knew the expected answers. During the procedure there were two observers along with the user providing some help when necessary so that all users complete all tasks. We emphasize on the time each user needs to complete one task and not in the correctness of answers (the answers are obvious). Only in task 6 (T6) we did not measure time because the correctness of the result is subjective. The purpose of this task was first to see if the users will use star-view with variable radius (R = 2, 3 expected) to complete the task and second which system the users preferred for completing this task.

5.4.3. Evaluation Results

The evaluation results are summarized in Tables 8, 9, 10. Tables 8, 9 present the times required by each user to complete the tasks in StarLion and Protégé correspondingly. Table 10 summarizes the aforementioned tables, displaying the time spend to complete each task in Protégé as a percentage over the total time spend in both tools. The values reported in the table are greater than 50% which means that almost every task was completed faster in StarLion by all users. The majority of the users fulfilled the requested tasks by exploring the schema with "star-view" and "show neighborhood" (star-view in Protégé). The FDPA that we adopted along with our "star-view" approach of variable radius lead to a very natural way of viewing and exploring the schema providing a clear understanding of the ontology and this was experimentally verified by the speed advantage we observe in the measurements. At this point we should bring back the fact that the users were restricted to use only jambalaya plug-in and not Protégé's complete suite since we are comparing only the visual representation of the ontology.

For task 6 (T6) all users who participated in the survey ranked StarLion higher than Protégé mainly due to the missing functionality of configurable radius. The configurable radius was proven to be very convenient for getting an initial layout since 80% of the users have used "star-view" with radius 2 or 3 (all users tried to start with a layout of "star-view" with radius 1). Then the user was able to further improve the layout depending on his needs

by hiding unwanted classes and properties and re-positioning some nodes. The initial layout depicted by StarLion was in most of the cases satisfactory without many manual interventions from the user. In conjunction with the automatic configuration that is supported, manual re-positioning was totally eliminated for 60% of the users.

It is worth noticing that a 20% of the users decided to give their answers in task 1 to 4 (T1-T4) without using "star-view" or "show neighborhood". StarLion's dependent namespace support along with the FDP algorithm proposed, made this task extremely easy resulting in very good times. Protégé's display algorithm seem to confuse the users who were trying to understand where are the superclasses and subclasses. In addition the missing functionality of dependent namespaces was an important drawback since it made the users re-load any related ontologies.

Finally, and to enforce our claims that the auto-configuration proposed in this work is significantly helpful, we were counting the times the users pressed the magic-button (auto-configuration). Overall, each user pressed the button 1 or 2 times in average something that verifies the usefulness of auto-configuring the layout parameters.

Tasks	**User Times in StarLion (s)**									
	U1	**U2**	**U3**	**U4**	**U5**	**U6**	**U7**	**U8**	**U9**	**U10**
T1	0.481	0.380	0.080	0.184	0.451	0.283	0.117	0.530	0.360	0.450
T2	0.200	0.258	0.220	0.304	0.252	0.575	0.168	0.334	0.160	0.390
T3	1.300	3.260	1.120	1.150	1.400	3.500	0.172	1.254	0.430	1.325
T4	2.460	2.350	1.270	0.400	2.000	1.030	1.310	2.700	2.270	2.550
T5	3.070	1.520	0.574	1.060	2.300	2.030	1.370	1.820	1.400	1.550

Table 8: Times for executing tasks in StarLion

Tasks	**User Times in Protégé (s)**									
	U1	**U2**	**U3**	**U4**	**U5**	**U6**	**U7**	**U8**	**U9**	**U10**
T1	1.100	1.570	0.483	0.574	1.050	1.560	0.140	1.120	0.590	1.150
T2	0.300	0.338	0.178	0.355	0.400	1.030	0.272	0.296	0.210	0.420
T3	3.300	5.350	2.550	3.350	3.550	4.490	1.390	2.562	1.350	3.050
T4	2.000	4.000	1.560	2.220	2.040	2.400	1.590	2.120	2.550	2.240
T5	3.150	5.210	2.370	1.490	3.200	3.000	5.280	3.452	1.110	3.220

Table 9: Times for executing tasks in Protégé

6. Related Work

There has been a lot of work over the years in the area of ontology visualization in general (regardless of using or not using an FDP layout algorithm).

Tasks	User Times in Protégé (% of total time "StarLion + Protégé")										
	U1	**U2**	**U3**	**U4**	**U5**	**U6**	**U7**	**U8**	**U9**	**U10**	**AVG**
T1	69.58	80.51	85.79	75.73	69.95	84.64	54.47	67.88	62.11	71.88	72.25
T2	60.00	56.71	44.72	53.87	61.35	64.17	61.82	46.98	56.76	51.85	55.82
T3	71.74	62.14	69.48	74.44	71.72	56.20	88.99	67.14	75.84	69.71	70.74
T4	44.84	62.99	55.12	84.73	50.50	69.97	54.83	43.98	52.90	46.76	56.66
T5	50.64	77.41	80.50	58.43	58.18	59.64	79.40	65.48	44.22	67.51	64.14

Table 10: Percentage of time to complete a task in Protégé

Most of these works refer to RDF/S ontologies but there are efforts that deal also with OWL, like [13]. The majority of these works try to lay out a schema in the best possible way but the subsequent actions by the user are mostly manual. There are efforts like [14] where the authors propose an extended FDP approach by adding extra *"particles"* to each node so as to gain volume and *"push"* away the neighborhood nodes; thus resulting in more sparse visualizations, but keeping near to each other nodes that are semantically similar. Moreover color density is exploited to provide information hints. Compared to ours this proposal lacks the ability for extending the star like views beyond radius 1 and to automatically support users' further steps after the initial placement. Other efforts, like *CropCircles* [15] employ tree cased views with emphasis in topology by placing the child nodes of a node inside that node. This simplifies the overall layout but hinders the representation of other relationships among the nodes apart from subclassOf (*isA*) relationships. [16] proposes the use of views, a well known feature of relational databases to simplify the schema to be visualized by specifying more accurately the part, density and complexity of the schema to be visualized. Similarly *OntoTrack* ([17, 18, 19]) tries to adjust the size of the presented information by using clustering techniques to preserve graph's density and keep the information amount low. It also uses additional presentation techniques to make this information available like pop-up windows, textual information panels, etc. Both these works focus on how we can limit the size of the schema to be visualized and visualize the outcome of this process. In that sense they differ from our work, since we try to also tackle the problem of self adjusting the initially proposed visualizations by the system.

Apart from the above efforts there is a wealth of tools that implement similar approaches and are part of everyday working systems. One could name here, *HOMER* [20] which is a system that supports ontology alignment and uses a window split in two to present two radius-like graphs to the user, who can work in independent (changing each graph separately) or linked

(changing both graphs at the same moment) mode. On the other hand GViz ([21]) is a general purpose visualization tool for RDF graphs. The tool has a graphical environment and allows also external scripting in order to automate tasks. One of the most popular open source tools is the *Jambalaya* plug-in of the Protégé tool (http://protege.stanford.edu), which offers a set of different algorithms (trees, radial, grids) and allows the user to select the type of objects they want to visualize (e.g. classes only, etc.). But it offers no automatic ability to restructure the visualization. *Touchgraph* (http://www.touchgraph.com/navigator.html), a commercial product, offers a Star-like view where the selected node is automatically located at the center with only its directly connected nodes visible, but only radius 1 is supported. The user is able to expand any of the available nodes but upon selecting a new center node the graph is reorganized. *Welkin* (http://simile.mit.edu/welkin/) provides a layout algorithm based on a force directed model but it limits users interaction to configuration of the layout and presentation parameters only. *ISWIVE* [22], incorporates the topic features from Topic Maps into RDF and thus into the corresponding visualizations. Finally, *RDF-Gravity*[4] provides a standard but non-configurable force directed layout with zooming facility, while the exploration is achieved only through filtering and textual information presentation. RDF-Gravity relies on Jena for the underlying graph storage and manipulation.

Compared to ours the aforementioned tools lack the ability of automatic configuration for the re-placement of the visualized schema based on metrics computed in real-time, after the initial effort and also lack the ability to extend visualizations beyond the radius 1 star-like views, a capability that proved to be valuable especially for visualizing large schemas in RDF. Most of the tools do not target specifically large ontologies and thus exhibit cumbersome behaviour when the schemas become very large, like the inability to distinguish between edges or the necessity of the users considerable intervention to replace nodes in order to correctly visualize the intended part of the graph. Table 11 compares these systems using a number of criteria.

Plain-Graph Drawing. The field of graph drawing and visualization is very broad. There are many works like [23, 24, 25, 26] using FDP algorithms and some of them also support star-like views with variable radius. All of these works refer to general (plain) graphs and they are not RDF-specific

[4]http://semweb.salzburgresearch.at/apps/rdf-gravity/

Feature	**StarLion**	Protégé 3.4 (Jambalaya)	Welkin	RDF-Gravity	TouchGraph
Visualization of Dependent Namespaces	√	No	No	No	It is possible to include file, but dependency is not evident
Star-view exploration	√	√	No	No	√
Star-view with variable radius	√	No	No	No	No
Auto layout	FDPA	Spring-layout among other alternatives (trees or radial)	FDPA-based (not sure about which one)	No	Spring-based
Interactive Improvement	Manual change, nail ,unail, +/- of parameters	Manual graph edit, not of the layout parameters	Manual graph edit, layout parameters, Nail/Unail	Manual graph edit, not of the layout parameters	Manual graph edit, not of the layout parameters, node expansion
Auto configuration of layout params	√	No	No	No	No
Top-K diagrams	√	No	No	No	No
Animation Stepping	√	Limited transition animation between alternative layouts	√	√	√
Configurable Node Length for Name Readability	√	On specific layouts length is adequate, but no configuration is available	No	No	No
Edge Name Readability	Customizable	On mouse hover	No	Customizable	No

Table 11: Related Systems

and for that reason are out of the scope of this paper. RDF graphs contain more information than plain graphs and have more visualization needs (*e.g.* subclass hierarchies must be vertical). For this reason we did not rely on such algorithms but we designed a dedicated force directed algorithm. Apart from this, the notion of namespaces does not exist in plain graphs where in RDF graphs plays a very important role and provides great help to the user if it is visualized right. Last but not least we have to tackle edge labels. The

readability and visibility of the labels is crucial in RDF graphs in contrast with plain graphs where in many cases they can be omitted. The tight integration between star-like views and placement algorithms used by StarLion helps to deal with the above issues.

Graph visualization libraries. Regarding graph visualization there are many general purpose libraries which provide frameworks for aiding graph drawing. Most of the tools discussed so far are based to some kind of generic library. Libraries provide a large quantity of interfaces and classes with rich functionality but in order to use them coding is always necessary. The only axis that we can compare StarLion with a visualization library is on the layout algorithm introduced. StarLion was built over JGraph [27] library. This library provides a generic *FDPA* where mainly repulsion between nodes is considered. There is no magnetic field and as a result there is no specific way for drawing class hierarchies (in fact in the library layer class hierarchies do not exist). Prefuse [28] and yFiles [29] libraries provide many drawing algorithms and they both support a simple form of *FDPA* which does not capture hierarchies. Automatic configuration of parameters for the *FDP* is not provided and the developer is responsible for setting up the parameters if they are not static. Star-view exploration is a modification of a BFS/DFS algorithm which all graph libraries provide. Top-K diagrams, dependent namespaces, node and edge configuration are domain specific issues and cannot be compared with general purpose libraries.

7. Concluding Remarks

In this paper we focused on (a) providing star-like graphs of variable radius, and (b) configuring automatically the parameters of a force-directed placement layout algorithm [5] based on quality metrics appropriate for RDF/S ontologies. The star-like graphs of radius 2 and 3 proved very effective for exploring real RDF schemas while the interactive adjustment of the layout parameters through the toolbar resulted in a user friendly interaction. Regarding the automatic configuration of the layout parameters, we proposed two quality metrics, namely, *verticality of specialization hierarchies*, and *area density* which take into account the characteristics (morphology of subclassof relationship, labels) of the visualized graph. The experimental evaluation

[5] Supporting *springs, electrical repulsion* and *magnetic field*.

showed that these metrics indeed capture the quality of the layout, and the corrective actions that we introduced (which actually adapt the strength of the electrical repulsion and the strength of the magnetic field) indeed improve the quality of the layout (as measured by the metrics). The improvements were verified also empirically by a user study (80% of the users preferred the corrected layout for graphs with radius equal to 3). The comparative (with Protégé), task-oriented evaluation showed that almost every task was completed faster in StarLion by all users.

In future we plan to apply these views also for exploring/visualizing ontology-based descriptions since star like-graphs, due to their ability to restrict the scope of the diagram, seem very suitable for the visualization of graphs consisting of class and property instances. For the same reason, they could be proved successful for browsing the LOD (Linked Open Data) cloud [30], which is in principle distributed, and thus the restriction of the scope is advantageous in terms of efficiency.

Another direction for future research is to investigate how to integrate the proposed visualization method with methods which are not graph-based. For instance, with menu or tree-based exploration methods like those of the interaction paradigm of *dynamic taxonomies and faceted search* [31], which are quite similar to those methods which are based on *Formal Concept Analysis* (FCA), or methods that combine information retrieval and FCA techniques, e.g. [32]. These methods allow the user to explore gradually a query answer and assist him/her in decision making. We should also note that the interaction paradigm of faceted exploration has been generalized to capture also the case of RDF/S knowledge bases (ontologies and ontology-based descriptions), e.g. see [33], and the case of fuzzy ontology-based descriptions [34].

References

[1] Y. Tzitzikas, J. Hainaut, On the visualization of large-sized ontologies, in: Procs of the 8th Intern. Conf. on Advanced Visual Interfaces, ACM, 2006, pp. 99–102.

[2] WIKIPEDIA: The Free Encyclopedia, http://en.wikipedia.org/ (2001).

[3] W3C: The World Wide Web Consortium, http://www.w3.org (1994).

[4] The CIDOC Conceptual Reference Model: ISO 21127, http://www.cidoc-crm.org/ (2006).

[5] M. Theodoridou, Y. Tzitzikas, M. Doerr, Y. Marketakis, V. Melessanakis, Modeling and Querying Provenance by Extending using CIDOC CRM , Journal of Distributed and Parallel Databases 27 (2010) 169–210.

[6] Y. Tzitzikas, J.-L. Hainaut, How to tame a very large ER diagram (using Link Analysis and Force-Directed Drawing Algorithms), in: ER, Springer Berlin/Heidelberg, 2005, pp. 144–159.

[7] P. Eades, A heuristic for graph drawing, Congressus Numerantium 42 (3) (1984) 146–160.

[8] T. Kamada, On visualization of abstract objects and relations, Ph.D. thesis, Dept. of Information Science, Univ. of Tokyo (1988).

[9] T. Fruchterman, E. Reingold, Graph drawing by force-directed placement, Software - Practice and Experience 21 (11) (1991) 1129–1164.

[10] K. Sugiyama, K. Misue, Graph drawing by magnetic-spring model, Journal on Visual Lang. Comput. 6 (3) (1995) 217–231.

[11] K. Sugiyama, K. Misue, A simple and unified method for drawing graphs: Magnetic-spring algorithm, in: Graph Drawing, Springer Berlin/Heidelberg, 1994, pp. 364–375.

[12] Y. Tzitzikas, D. Kotzinos, Y. Theoharis, On ranking rdf schema elements (and its application in visualization), Journal of Universal Computer Science 13 (12) (2007) 1854–1880.

[13] S. Brockmans, R. Volz, A. Eberhart, P. Loffler, Visual modeling of OWL DL ontologies using UML, Lecture Notes in Computer Science 3298 (2004) 198–213.

[14] K. Tu, M. Xiong, L. Zhang, H. Zhu, J. Zhang, Y. Yu, Towards imaging large-scale ontologies for quick understanding and analysis, Lecture notes in computer science 3729 (2005) 702–715.

[15] T. Wang, Parsia, Cropcircles: Topology sensitive visualization of OWL class hierarchies, in: Intern. Semantic Web Conf., 2006, pp. 695–708.

[16] N. Noy, M. A. Musen, Specifying ontology views by traversal, Lecture Notes in Computer Science 3298 (2004) 713–725.

[17] O. Noppens, T. Liebig, Interactive visualization of large owl instance sets, in: Proc. of the Third Int. Semantic Web User Interaction Workshop, 2006.

[18] T. Liebig, O. Noppens, Ontotrack: Combining browsing and editing with reasoning and explaining for owl lite ontologies, in: International Semantic Web Conference, 2004, pp. 244–258.

[19] O. Noppens, T. Liebig, Understanding large volumes of interconnected individuals by visual exploration, Lecture Notes in Computer Science 4519 (2007) 799–808.

[20] O. Udrea, L. Getoor, R. Miller, Homer: Ontology alignment visualization and analysis, in: 6th Intern. and 2nd Asian Semantic Web Conf. (ISWC2007+ASWC2007), 2007, pp. 111–112.

[21] F. Frasincar, A. Telea, G.J.Houben, Adapting graph visualization techniques for the visualization of rdf data, Visualizing the Semantic Web: Xml-based Internet And Information Visualization (2006) 154–173.

[22] X. Chen, C. Fan, P. Lo, L. Kuo, C. Yang, Integrated visualization for semantic web, in: Intern. Conf.on Advanced Information Networking and Applications, 2005, pp. 701–706.

[23] E. Adar, Guess: a language and interface for graph exploration, in: Procs of the SIGCHI conf. on Human Factors in computing systems, ACM New York, 2006, pp. 791–800.

[24] A. Perer, B. Shneiderman, Integrating statistics and visualization: case studies of gaining clarity during exploratory data analysis, in: CHI, 2008, pp. 265–274.

[25] M. A. Smith, B. Shneiderman, N. Milic-Frayling, E. M. Rodrigues, V. Barash, C. Dunne, T. Capone, A. Perer, E. Gleave, Analyzing (social media) networks with NodeXL, in: C&T, 2009, pp. 255–264.

[26] P. Shannon, A. Markiel, O. Ozier, N. Baliga, J. Wang, D. Ramage, N. Amin, B. Schwikowski, T. Ideker, Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks, Genome Research 13 (11) (2003) 2498.

[27] J. Bagga, A. Heinz, JGraph: A java based system for drawing graphs and running graph algorithms, in: Graph Drawing, Springer, 2002, pp. 459–460.

[28] J. Heer, S. Card, J. Landay, Prefuse: a toolkit for interactive information visualization, in: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM, 2005, pp. 421–430.

[29] R. Wiese, M. Eiglsperger, M. Kaufmann, yFiles: Visualization and automatic layout of graphs, in: Graph drawing: 9th international symposium, GD 2001, Vienna, Austria, September 23-26, 2001: revised papers, Springer Verlag, 2002, p. 453.

[30] C. Bizer, T. Heath, T. Berners-Lee, Linked data-the story so far, International Journal on Semantic Web and Information Systems 5 (3) (2009) 1–22.

[31] G. M. Sacco, Y. Tzitzikas, Dynamic Taxonomies and Faceted Search: Theory, Practice, and Experience, Springer, 2009.

[32] D. Poshyvanyk, M. Gethers, A. Marcus, Concept Location using Formal Concept Analysis and Information Retrieval, ACM Transactions on Software Engineering and Methodology(to appear).

[33] S. Ferré, A. Hermann, Semantic search: reconciling expressive querying and exploratory search, The Semantic Web–ISWC 2011 (2011) 177–192.

[34] N. Manolis, Y. Tzitzikas, Interactive Exploration of Fuzzy RDF Knowledge Bases, Proceedings of the 8th Extended Semantic Web Conference (ECSW'2011) (2011) 1–16.