SETIT 2005
3rd International Conference: **S**ciences of **E**lectronic, **T**echnologies of **I**nformation and **T**elecommunications
March 27-31, 2005 – TUNISIA

Visualization of Subsumption Hierarchies in Ontologies

Anis Zarrad[1], Volker Haarslev[2]

[1]Ottawa University, SITE,
[2]Concordia University, Department of Computer Science and Software Engineering,

azarrad@site.uottawa.ca

haarslev@cse.concordia.ca

Abstract: The RACER system is a knowledge representation system that implements description logic reasoning. It offers reasoning and evaluation services for multiple concepts (TBox) and multiple individuals (ABox) as well. The RACER system responds to taxonomy queries related to description logic. The body of the response contains information about a relational structure called a concept hierarchy or subsumption hierarchy. In this paper, we propose an efficient algorithm to visualize the concept hierarchies by producing several geometric representations. The display of the concept hierarchy in a single screen has been proven to be useful and helpful for ontology designers. In fact they can easily identify the hidden relations that were discovered during the Tbox classification process.

Key words: ontology; layered graph; autonomous sets , drawing algorithm

1 Introduction

Graphs drawings [12] are used to represent information that can be modeled as objects and connections between those object. The visualization of the concept hierarchies allows the user to better understand the domain ontology [9]. It essentially takes nodes and relations from the taxonomy file and creates a layout for them on the screen. Graph theory and order theory may play a major role in getting a good layout that can help the user to understand the application.

The main idea of the classification is to compute the ancestor (Parents) and descendant (Childrens) concepts for each concept name. The taxonomy is defined as a categorization of concepts based on a certain criteria, it lists for each concept it's most specific sub-concepts and super-concepts.

In description logic systems it is often necessary to make legible taxonomy information about the knowledge bases expressing concept and concept hierarchy. There are two major reasons for this: firstly, retrieve the anomalies in the ontology design, secondly, to better understand the ontology's domain.

The user can receive the taxonomy information as a text file. The text file is created when the user instruct the RACER system to answer the taxonomy query which is a classification ontology process to compute the subsumption relationship between all concepts names mentioned in a TBox. The result is referred as the taxonomy of a TBox and gives for each concept name two sets of concept names listing its "parents" (direct subsumers) and "children" (direct subsumees). The result may be given in text format:

Result example
1:(TOP NIL (AGE ANIMAL CAT))
2:(AGE (TOP) (BOTTOM))
3:(ANIMAL (TOP) (BOTTOM))

The ontology classification process requires a specific kind of drawing to understand the ontology design (concepts and their subsumption relationships) which is called the hierarchy graph or layered graph [31].
A layered graph is an oriented acyclic graph. The vertices of a layered graph can be partitioned into sets (L0={s}, L1, L2, etc.), called levels or layers, such that each edge of the graph, that is an ordered pair of vertices, has the first component in a level Li and the second component in a level Lj where i<j. Hence, there are no edges between two vertices in the same layer.

The interpretation process has to first apply rules for layer 0 as long as possible, then rules for layer 1, etc. This makes it easily possible to specify a hierarchical view of ontology concepts.
The rules assign concepts without parents to layer 0, and then place the children of the concepts in layer 0 into layer 1, and so on.

Using a hierarchical approach to represent the taxonomy file has a number of advantages.

-When we examine the graphical representation of a model we use our visual cognitive apparatus which has some millions of years of evolutionary advantage over our text-reading abilities. The two-dimensional representation of a diagram is a lot more expressive than text, which is typically scanned from left to right and from top to bottom. Diagrams can be viewed following different directions to gain distinct insights:

- Ergonomically friendly presentation of the ontology;
- Easy browsing and navigation through the ontology;
- Efficient retrieval and searching of a specific concept name;
- Identify the hidden relations.

The layout must satisfy several aesthetics to create a "good" layout. Aesthetics differ from one application to another. To achieve the aesthetic goal we may adopt three major drawing parameters for the concepts hierarchy display:

Straight line drawing: each connection is drawn as a straight line segment.

Aesthetics: properties applied to the drawing achieve the understandability and readability of concepts. The major important element in this parameter is to reduce edge crossing. This problem was first studied by Warfield [43] and similar methods were discovered by Carpano [44], Sugiya, Tagawa, and Toda [45]. There has not been many results dealing with this problem, probably due to the difficulty of the problem. In this work we follow Di Battisa and Tamassia's approach [12].

Space minimization: the concepts hierarchy display should fit in one screen, if the output is larger than the screen width, many techniques may be applied to reduce the display width.

2. Drawing Algorithms

2.1 Introduction

Before we apply the drawing step we need a crucial step which is the parsing. The taxonomy file should be parsed using the recursive descent parsing to crate an abstract data structure that must be used as input for the drawing process. The running time for the parsing process is polynomial to the size of the taxonomy file.

It is important to take into consideration some properties of the concept hierarchies, to decide the best drawing approach that must be used.

A concept hierarchy is a relational structure, consisting of a set of concepts (classes) and relationships between those concepts. This structure is modeled as a graph G=(V, E) .

Definition: Let G = (V, E) be an acyclic directed graph. A vertex is called maximal if it has no parents in G. A vertex is called minimal if it has no children in G. It is obvious that for acyclic directed graphs there are always minimal and maximal elements.

Definition: Let G = (V, E) be an acyclic directed graph. We define the Top-to-Bottom decomposition into levels as follows: the level L0 consists of all vertices without parents (or set of maximal elements in G). For every i>0, the level Li consists of all maximal elements in G-{L0, L1, …, Li-1}.

Note that for every node v in a non-empty layer Li, there exists at least one simple path Pv starting from v and passing through every level Lk for $0 \leq k \leq i$. That is a sequence of vertices vi = v, vi-1... v0 such that $vk \in Lk$, for every $k \leq i$ and (vk , vk+1) is an edge in G.

A similar decomposition could be done from Bottom-to-Top: The level L0 consists of all vertices without children (or set of minimal elements in G). For every i>0, the level Li consists of all minimal elements in G-{L0, L1, …, Li-1}.

Clearly, any vertex in layer (or level) Li should have at least an ancestor in layers Li+1.

In a layered (or leveled) drawing, all vertices of the same layer will be drawn on a horizontal line and the edges are represented by straight lines between different layers. Of course, and since the structure is an acyclic graph because the RACER would collapse a cycle in the concept hierarchy during the reasoning process, it is impossible to have edges between vertices in the same layer.

Note that we could have edges connecting vertices from non-consecutive layers. These edges are called Bypassing edges. Direct Edges are defined as edges that connect two nodes in a consecutive layer.

We did notice however that in the examples we dealt with, most of the outgoing crossing edges are from the Top vertex or the Bottom vertex depending on whether we use the Top-to-Bottom or the Bottom-to-Top layout strategy.

The Bottom node in the hierarchy can be viewed as an imaginary virtual node; it is used for encapsulating and covering the ontology domain. The deletion of this concept does not affect any ontology parameters.

2.2 proposed drawing algorithm

Our goal is to produce a readable drawing of a given taxonomy file by analyzing its properties and then making a decision about which drawing algorithm should be applied. For instance, one major problem is caused by the crossing edges which could seriously affect the readability of the drawing. Reduction of crossing between edges seems to be the most crucial parameter to minimize. It creates the biggest problem in readability, especially when the graph is large.

The computation of the number of crossing edges is needed, since it is used as a major parameter whenever the readability of a drawing algorithm is analyzed.

It is very common not to draw the direction of the edges for an acyclic directed graph. Instead, the position of the vertices on the plane will indicate the directions. With a vertical (also called upward drawing) orientation and for an edge (u, v) in the

graph, the vertex y will be drawn higher up than x (the y-coordinate of u is strictly smaller than the y-coordinate of v). However, for the horizontal drawing the vertex y will appear at the right of the vertex (the x-coordinate of u is strictly smaller than the x-coordinate of v).

Relationships between concept hierarchies are drawn with a vertical orientation, so the directions of the edges are omitted but the y-coordinates of the nodes on the screen define the direction (upward). Vertices are drawn as circles or rectangles (for groups of vertices) and edges are drawn as straight-line segments.

The number of concepts related only to the top or only to the bottom is usually very large and these concepts could increase the complexity of the drawing, although there are simple ways to deal with theses relationships.
The size of the graph could be very large and it becomes difficult if not impossible to fit the display in a single screen.
Designing an approach that solves all of the above mentioned problems and which could be used for all taxonomy files is known to be a very hard task. However we believe that our approach is useful to obtain an acceptable drawing in many cases.
Our proposed drawing approach has eight major steps or sub-algorithms:

1. Test the hierarchy by removing the non essential edges (Cycle remove step).

2. Layer decomposition for both strategies orientation Bottom-to-Top and Top-to-Bottom: this step consists of assigning a layer for each node.

3. Distribution decision between the two strategies: is defined as choosing the best strategy which produces a homogenous distribution between layers.

4. The autonomous set decomposition: Consists of grouping several nodes that have the same properties into one virtual node.

5. Layer permutation: Deals with the order of nodes that must be chosen for each layer.

6. Cross reduction for each two consecutive layers using the layer by layer sweep algorithm. In this step we discard the crossing edges that pass through more than one layer.

7. Layer Bifurcation: Consists of finding the layer number where bypassed edges need to be bifurcated.

8. Draw the bypassed edges: show the technique used to draw the crossing edge with respect to the node order already made in step number 5.

2.3 Testing the hierarchy for non-essential edges

An edge (u, v) is called a redundant transitive edge if there exists a directed path of edges from u to v, that does not contain the edge (u, v).
Although concept hierarchies are not supposed to contain transitive edges, it is important to check if there are any and delete them. In upward drawings the transitive edges, if they exist, they could be deleted without losing any data. The existence of such an edge will create a problem in defining the layers. For instance, if (u, v) is a transitive edge, then the vertex v will belong to different layers at the same time.

2.4. Layout algorithm

The algorithm consists of assigning a layer to each node. To determine the layer of each node we may use two different strategies and then decide the orientation.
The layout procedure first identifies all the roots nodes or the leaf nodes in the graph depending on the orientation of the drawing.
The first step of our layout algorithm is to assign nodes with no parents on the first level and then remove those nodes from the adjacency list, next place the children of the previous nodes on the second level. Third level nodes are the direct parents of nodes of the second level and so on. This process continues until all nodes have been assigned a level.
The layout algorithm computes the number of layers in the graph as well as the number of nodes on each layer. These parameters are useful to decide about the final drawing strategy to be used.

Algorithm Layering (G);
Input: an adjacency list L that represents the graph G= (V,E) and a ParAdj list that contains the parents node for a given node name.
Output: layering of G as a linked list L1, where each element contains a node name and layer number.

```
Begin
NumLayer=0;
Initially all nodes are unlabeled
Repeat
        For each node u in L
                If (ParAdj[u] is empty)
                        Add the u node and the NumLayer to the list L1
                        Label the vertex u.
                End if
        End for
        NumLayer++;
        Remove the labeled vertex u from the list L.
Until no vertices are left in the List L
End
Return L1
```

2.4.1 Complexity analyses

The total running time of the layout algorithm is O(n2), where n is the number of nodes in the graph.

2.5. Layout orientation decision

The distribution of the vertices on the different layers could be more or less proportional depending on the drawing strategy we use. The layout orientation decision is based on two drawing parameters: The number of nodes on each layer and the number of layers. Taking into consideration those parameters we can conclude the best display orientation. Using a statistical measure of dispersion. The Layer variance (noted as σ2) is the average square distance from the mean node:

σ2= ((x1 - μ)2 + (x2 - μ)2 + ... + (xn - μ)2) / n , where μ = (x1 + x2 +…….+ xn) / n and

xn is the number of node in the n th layer.

The layer standard deviation (noted as σ) σ= √ σ2 . This parameter must be close to the mean layer μ to get a proportional layer distribution.

A simple procedure OrientationGraph has been implemented to calculate the σ parameter for both layout strategies. Then we calculate a new parameter Δ = |μ-σ|. The best approach with the smallest Δ is chosen. In some case both strategies give the same number, then the Top Bottom approach is chosen.

We noticed that in most examples we worked with, the Top-Down design approach seems to be more appropriate to create a drawing for the ontology. The number of crossing edges in each file has proven very small for the Top Bottom layout with the omit Bottom concept option. Moreover, with this approach the distribution of the nodes on the different layers seems to be more proportional than with the Bottom Top layout.
Of course, in a Bottom-to-Top drawing there will be no crossing edges coming from the Bottom, since all nodes connected to the Bottom will be in the first layer. Similarly removing the Top in a Top-to-Bottom drawing will not decrease the number of crossing edges.

```
Algorithm OrientationGraph (R1, R2) ;
Input: R1, R2 arrays that contains the number of nodes in each layer for each respective strategy (Top-to-Bottom and Bottom_to-Top) .
Output: BottomTop or TopBottom
Begin
BT=0; TB=0;
        For (all element x in R1)
                BT=BT + x
        End for
        μ1= BT / R1.size ();
        For (all element x in R1)
                BT=BT + (μ1-x) 2
        End for
        σ1 2= BT / R1.size ();
        For (all object x in R2)
                TB= TB + x
        End for
        μ2= TB/R2.size ();
        For (all element x in R2)
                TB= TB + (μ2-x) 2
        End for
        σ2 2= TB/R2.size ();
        Δ1=| μ1- √ σ1 2 |
        Δ2=| μ2- √ σ2 2 |
        If (Δ1<=Δ2)
                Return BottomTop
        End if
        Else
                Return TopBottom
        End else
End
```

2.5.1. Complexity analysis

The algorithm uses a simple naïve process with a run time O(N+N') where N and N' is the number of layers for each strategy.

2.6. Autonomous sets Decomposition algorithm

The second major technique used to simplify the drawing without losing data is a decomposition technique. It combines any group R of nodes that have the same relationships with all nodes outside of R.
Definition: Let G= (V, E) be a directed graph. A non-empty set of nodes A is called autonomous set and denoted Aut (v,w) if for every node u in A and v in V-A, there exists an edge (v, u) and if and only if there is an edge (u, w) where w in V-A for every u in A. (Figure 1 and 2)

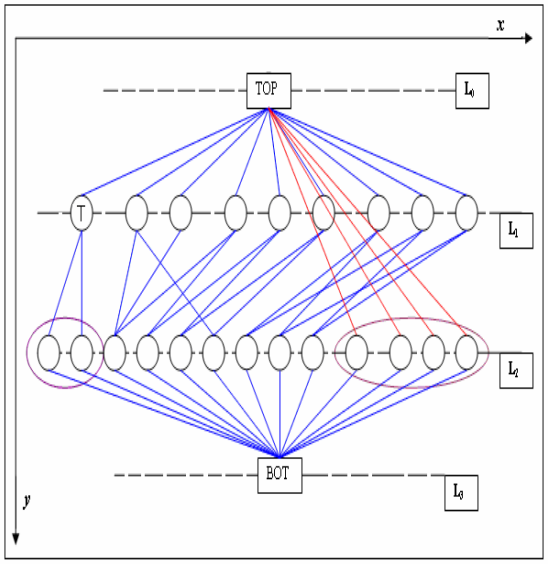

Figure 1: The oval marker shows the nodes that must be joined

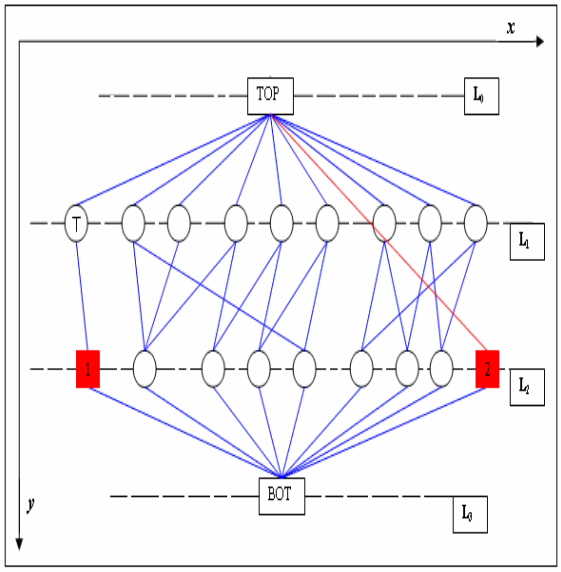

Figure 2: The gray rectangle represents the Autonomous set Aut (T, BOT) and Aut (TOP, BOT)

By grouping all nodes of an autonomous set into one virtual node, we greatly simplify the drawing without losing any data.
The goal of this part is to create a data structure with a reduced number of vertices without affecting the displayed information. This technique is related to the large graph aesthetic parameters because many nodes are joined and the size of the graph may be reduced.
Once the nodes are placed in their layer, the procedure AutonomousNodes is called.
Here is the algorithm which finds all the autonomous sets in the graph G.
CandidateNode: procedure checks if the node may be a candidate to be joined into the autonomous set. It returns a Boolean. There are two conditions that are checked for a node: the node must only have one parent and if the node is already in an autonomous set, then it is discarded.
UpdateAdjacencyList: procedure that updates the adjacency list. If a join decision is made, we remove all nodes that must be joined and replace them by a new single node.
Input: L a Link List (Adjacency List), N1, N2 nodes must be joined
Output: L a new Link List (Adjacency List) with homogenous nodes
The algorithm will use an adjacency list as input. The process is done for all nodes and starts from the top to bottom or bottom to top depending on the orientation of the graph.
Adj[u] is defined as: the set of children of node u.

Algorithm: AutonomousNodes (G);
Input: L a Link List (Adjacency List represents the concept hierarchy graph) and Adj[u] that represent the children set for a given node u.
Output: L a new Link List (Adjacency List with autonomous sets)
Begin
For each node u in the L
For each element xi in Adj[u]
If (CompareLists (Adj [xi], Adj [xi+1]) =true)
If (CandidateNode (xi) =true and CandidateNode (xi+1) =true)
1-Create autonomous set with xi and xi+1
2-UpdateAdjacencylist (L, xi, xi+1)
End if
End if
End for
End for

Return L
End

2.6.1. Complexity analysis
The maximum number of iteration that must be made is related the number of pairs node.
Suppose the number of nodes is n then, the total number of iteration is n*(n-1) / 2.
The running time of the autonomous sets algorithm is O(n2) where n is the number of nodes in the graph.

2.7. Layer permutation algorithm

Unfortunately, the algorithm used in the crossing step uses a random order of nodes for each layer, and especially for the first layer. The challenge of this part is in choosing an optimal ordering within the layers that could be useful and helpful to draw pretty concept visualizations. The choice of the order is on the computation of the direct degree. Let us denote the direct degree of a node u as Deg (u) = n where n is the number of nodes v, where u∈Li, v ∈ Li+1, and (u,v) ∈

E. Once the computing of the children is done for a specific layer, a sort procedure could be applied to sort them using their direct degree number in an appropriate data structure. First we place the nodes with the highest degree in the middle layer as a pivot, then place the rest of the nodes on each pivot side starting from the left until all nodes are placed. The layer permutation must be applied to each layer.

Algorithm LayerPermutationNodes();
Input: random node order as a linked list L for a selected layer.
Output: a new permuted node order as a linked list L.
Begin
For each element u in L
    Compute the direct degree n
    Add n to the list Lst
End for
    Sort Lst using the descendent order.
    Place the first node v in the Lst in the middle of the layer fixed as a pivot P. Remove v from the list Lst.
For each element ui in the Lst
    If (i is even)
        Place ui in the left of P
    Else
        Place ui in the right side of P
End for
End

5.7.1. Complexity analysis
This simple procedure runs in linear time O(n) where n is the number of nodes in the graph.

2.8. Crossing edges reduction algorithm

The minimization of the number of crossing edges is a very difficult problem in general. In fact, it is a NP-complete problem even if the graphs contain only two layers [30]. However, in the research literature there is a number of heuristics that have been developed for this problem.

Of course the relative positioning of the nodes within every layer will decide about the number of crossings in the drawing.

We use the layer by layer swap algorithm [28]. It is described as follows. First we choose a concept ordering of a layer L1. Then for each layer i= 2, 3, 4….h, the concept ordering of layer Li-1 is held fixed when we reorder the vertices in layer Li. Unfortunately the “two layer crossing problem” is NP-complete [29].

The ordering technique is based on the number of crossings. The number of crossings between two nodes u and v in the same layer Li is denoted as Cuv and defined as: the number of crossing that edges incident with u make with edges incident with v when xu<xv. More formally, for u,v∈Li, u≠v, xu < xv, Cuv is the number of pairs (u, w), (v, z) of edges with xz < xw, where z and w ∈ Li-1, and xu, xv, xz and xw represent respectively the coordinate for the nodes u, v, z and w.

This procedure is called to place the nodes in the vertical line with respect to each other. It is done for all nodes in the top (bottom) layer depending on the orientation, then for all nodes in the next layer and so on until the bottom (top) layer is done. If any node changed position in any layer during the process, the whole procedure is repeated again. This continues until either no node moves in a pass or until a specified number of iterations are reached.

Algorithm LayerByLayerSwap(G, x1)
Input: two layered graph G = (L1, L2, E) and a fixed vertex order x1 for L1
Output: vertex order x2 for L2
Begin
We choose a random order for L2
Repeat
    Scan the vertices of L2 from left to right, exchanging an adjacent pair u, v of vertices, whenever Cuv>Cvu

Until the number of crossings remain unchanged.
End

2.8.1 Complexity analysis
The computation of the crossing number Cuv depends on the relative position of u and v. Each scan in the repeat loop can be done in O (|L2 |) and there are O (|L2 |) scans. The time complexity of the layer by layer swap algorithm is O (|L2 |2). The total running time is O (n |L2 |2), where n is the number of iterations to get a stable crossing number.

2.9. Layer Bifurcation algorithm

Once we have done all the previous steps we may get an acceptable drawing with an optimal number of crossings. Unfortunately, in many cases the bypassed edges (u, v) may reduce the visibility and the readability by causing some node shape intersections. We compute the upper layer number that represents

the beginning of the bifurcation for each bypassed edge.
This technique must be used for each crossing edge (u,v) ∈ E where u∈Li, v ∈ Lj and j>i+1. We try to find the layer number where we have to bifurcate the crossing edge. The idea is based on two parameters, xu and xv, that represent respectively the x coordinate for u and v. This procedure is done for each two consecutive layers starting from the layer Li until the layer Lj.
Calculate the crossing number for the edge (xu,i and xv,i+1), for a given drawing. xu,i and xv, i+1 respectively represent the xu coordinate in the layer Li and the xv coordinate in the layer Li+1. We compute the crossing number for each iteration and then we save the smallest one with the layer number.

Algorithm LayerBiffurcation()
Input: a Bypassed edge (u,v) with u= (x1, y1) and v =(x2, y2)
Output: an integer y representing the upper layer number
Begin
n=GetLayer(u) // return the layer number that the vertices u belongs.
m = GetLayer(v)
C=+∞ // initialization for a big number
For (i=n to m)
Create an imaginary node u' with coordinates (x1,i)
Create an imaginary node u" with coordinates (x2,i+1)
Create an imaginary edge (u', u")
Compute the crossing number Cu" w for each node w in the layer i+1,

If (C> Cu" w) Then
C= Cu" w
y=i
End if
End for

Return y.
End

2.9.1. Complexity analysis
The running time of this algorithm is dominated by the compute crossing procedure (step 4). The crossing complexity is O (|L|2) where L is the number of edges between two consecutive layers. The total running time for one bypassed edge is |L|2*(m-n-1). Where m and n are respectively the upper and lower layer number.

2.10 The Bypassed edges draw algorithm

The basic idea is drawing the bypassed edges as a straight line through all layers, but with respect to the nodes that intersect with the edges using the layer bifurcation technique already described in the previous step (Figure 3 and 4). This algorithm adopts the polyline drawing edges with a minimum number of bends to provide flexibility instead of curved edges which may be difficult to track by the users.
Having the layer number where we should draw the bifurcations done, we start drawing a vertical straight line between two consecutive layers from xu,i. until we reach the layer bifurcation, then we cross the line and keep drawing the vertical straight line between two consecutive layers until we reach the node xv, j .
In most cases the drawing of a vertical straight line between two consecutive layers may pass through a node (circle with a radius of 15 pixels). Three cases must be considered.

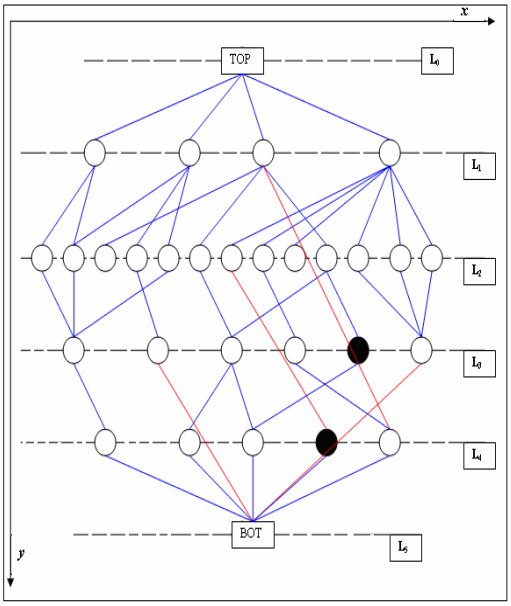

Figure 3: The black node shows the intersection between bypassed edges

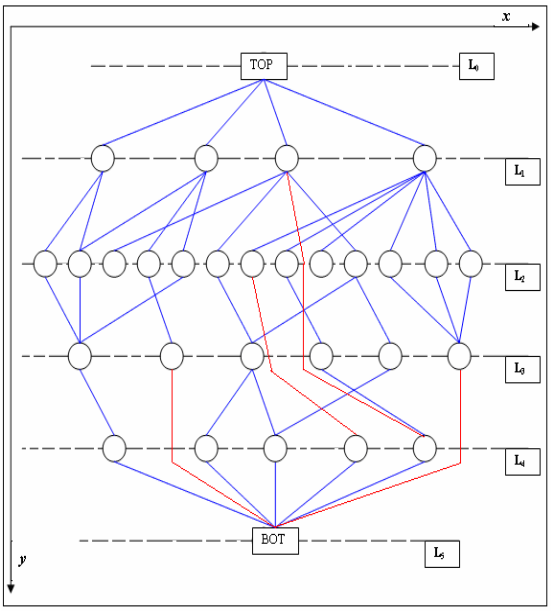

Figure 4: The bypassed edges drawing using the layer

bifurcation and “draw bypassed edges “algorithm

```
Algorithm BypassedEdgeDraw(T)
Input: a drawing T with optimal number of crossings
Output: a new drawing T‘ with a bypassed edge drawing
Begin
For each bypassed edge e=((x1, y1),(x2, y2))
        Integer i = Layer Bifurcation (e)
        If a layer number#i
                Calculate the intersection distance D between the line and the node.
                        If D<= 15 then diverge the line with D pixels to the right.
                        If D> 15 then diverge the line with D pixels to the left.
                        If D=0 then draw a straight line

        End if
        Else
                Bifurcate the line from the (x1, y ) coordinate to the (x2 ,y+1)
                Coordinate. //y and y+1 represent respectively the coordinate for the layer Li and
                Li+1
        End else
End for
End
```

2.10.1 Complexity analysis

The time complexity is strongly related to the layer bifurcation procedure already computed in the previous step. The total running time of this algorithm is O(K*Q) where K in the number of Bypassed edges in the drawing and Q =|L|2*(m-n-1). Where m and n are respectively the upper and lower layer number for each bypassed edges and L are the number of crossing between two consecutive layers.

3. Application description

We designed a user interface that communicates with the RACER [8] system through a TCP/IP protocol to respond to a specific query Taxonomy with is a classification process to compute subsumes and subsumees for each concepts. The response is saved in a text file called taxonomy file, it can be visualized as a concept hierarchy using graph drawing algorithms.
The main task for the application is to generate a hierarchical graphical view using the Taxonomy file generated by the RACER system.

We should be very careful about minimizing the area of the screen used for the display. We require a global view for the user. The best situation to display the concepts hierarchy would be to use a single screen even for large data sets, because scrolling has shown to be a big distraction to the user. We adopt several conventions for the graphical display such as: we use a fixed sized shape for the nodes, we display only the first three characters of the concepts name; the full name will be displayed using the tool tips mouse option. With a rectangular screen, the horizontal lines could accommodate a larger number of vertices. In most cases the number of layers of the hierarchy is not so big; however, the sizes of the layers could be very large. So it is more appropriate to use a horizontal orientation in our drawings.
The application is meant to be suitable for visualizing large hierarchical view as well. For a useful display, there is a need for a set of flexibilities and features to be developed: zoom, rotations, improved animation, etc. This leads to a better interactivity and an easier browsing of the hierarchy.

4. Application architecture

The application design is based on a two-tiered system. An applet will be downloaded at the user terminal and will communicate with the RACER system that resides on the server. The application architecture uses the layered design (Figure 5) because of the following reasons:

- The reusability and maintenance of the system; and
- To simplify and separate the major components in the design system.
- To provide simplicity, flexibility, and adaptability to the application

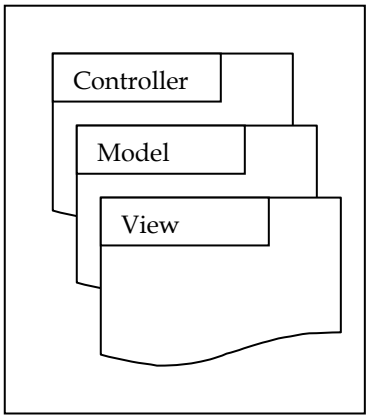

Figure 5: Application architecture.

We define the services that can be offered by each layer:

First layer: A View provides graphical user interface (GUI) components for a model. This class coordinates the appearance of the GUI. It decides where all the controls and allocated displays (labels, text fields, buttons, concepts Box, relation line, etc.) are positioned. When notified by the model that it has changed its state, the view updates itself by calling methods in the model that return all the information that the view must display. In Java the views are built with AWT or SWING components.

Second layer: The Model is the central class that represents the most important part in the design because it carries out huge tasks and operations. It has a set of public functions that can be used to achieve all of its functionality. Some of those functions are query methods that permit a "user" to get information about the current state of the model. Others methods permit the state to be modified and the rest deal with the parsing process, data collection, drawing algorithms, and network connections.

The model layer handles the most important methods for the system and has a complex structure design. We may use the layered approach [19] in the model class itself (Figure 6).

The first subclass represents the network connection. It delivers the connection between the application and the RACER system to classify the ontology and to respond to queries. This layer is related to the network service and to devices that require connectivity. In this layer, we use the TCP/IP protocol as the underlying transfer protocol.

The system calls the socket function to create a socket and connect to the racer socket once the connection is in place. The client sends the ontology file to RACER through this connection and asks the receive function to get the taxonomy query response from the RACER application. When all the data has been received, the system calls the close function to close the socket.

The default port number for the RACER system is 8088.

The second subclass creates an appropriate data structure that must be used for the upper layer through a set of procedures such as the data collection process and the parsing algorithm. The use of appropriate data structures is required in order to have more efficient algorithms.

The third subclass contains all the drawing algorithms for constructing the geometric concept hierarchy graph. Several result and graph proprieties must be applied to implement such an algorithm in a sophisticated and efficient way to reflect the major thesis contributions.

Finally, the last subclass manages the behavior of the data. It should be able to register the views, notify all of its registered views when any of its function causes its state to change.

In Java, the View Model Class consists of one or more classes that extend the class java.util.Observable.

To make the system code more usable and comprehensible by the developer, each layer must interact with the other layer specifically through well defined interfaces. We use the down to top approach (*Figure2*) to interact between the **Model** subclasses.

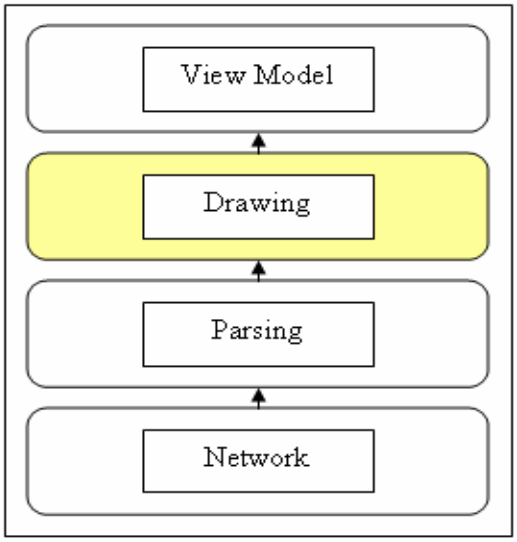

Figure 6: Model subclasses.

Third layer: this represents the controller class and updates the model as necessary when a user interacts with an associated view. The controller can call update methods of the model to get it to update its state, in which case the model will notify all registered views that a change has been made, so they will update what they display to the user as appropriate.

In java, the controller is presented as listener; it can be used and called directly from the event structure java library.

5. The application interface

The challenge for the application is to create an acceptable user interface that incorporates multiple inputs to produce a layered graph viewed as output. The application interface is a tool for prototyping the concept hierarchy display in user interfaces. This tool creates a hierarchical view using a bar menu icon or the file menu depending on the user ergonomics. The user can interact with the system interface using the mouse to sketch some query and operations for a specific concept name selected by the user to better understand the hierarchy.

To facilitate the transition from a global view to a local view, a pop up menu was built using the right mouse click with an option for accumulating ideas and understanding the appropriate local view.

Figures 7 to 10 show a series of screen captures of the main display area with some principles scenarios. They Show progressive steps to display and interact with the hierarchy concept. The user can use the mouse and the menu icons to retrieve the necessary information about the selected concept name.

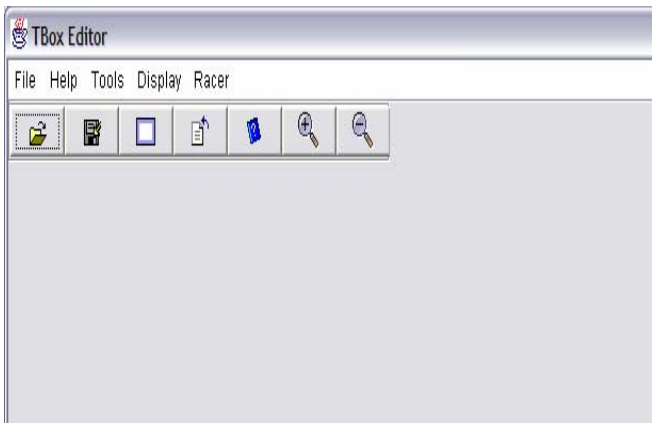

Figure 7: The bar and the file menu of the system interface.

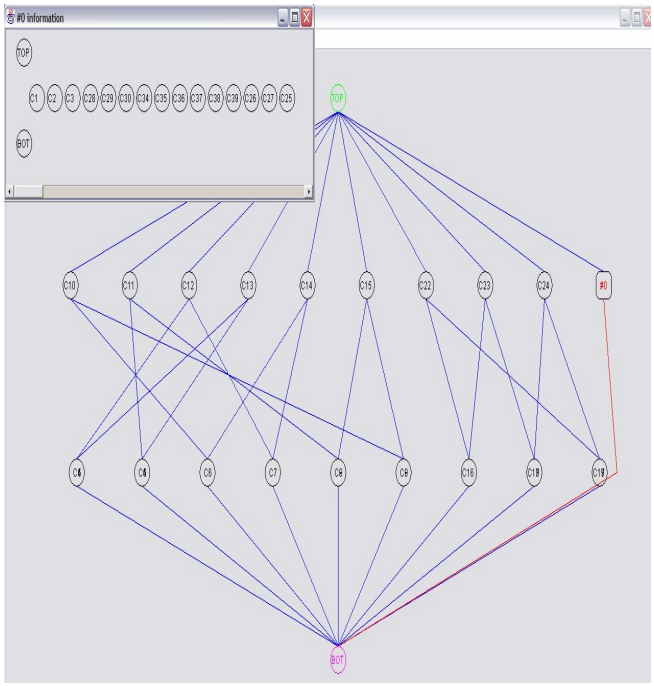

Figure 8: Zoom Option for the autonomous set.

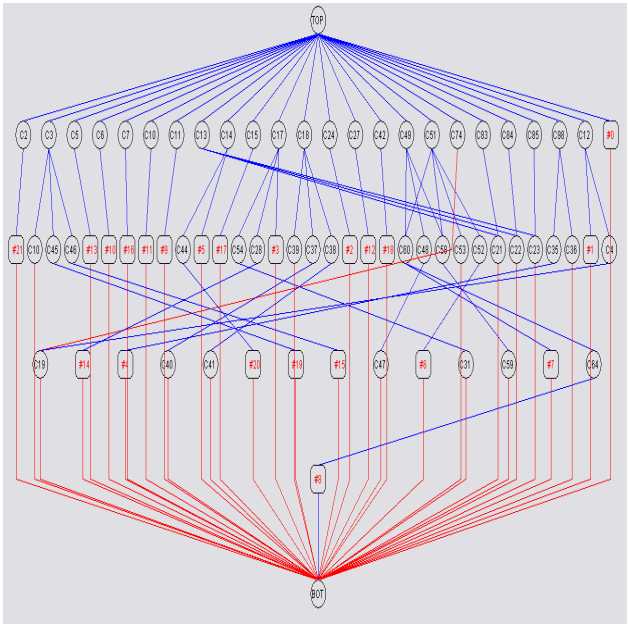

Figure 9: The Top to Bottom display using the autonomous set algorithm.

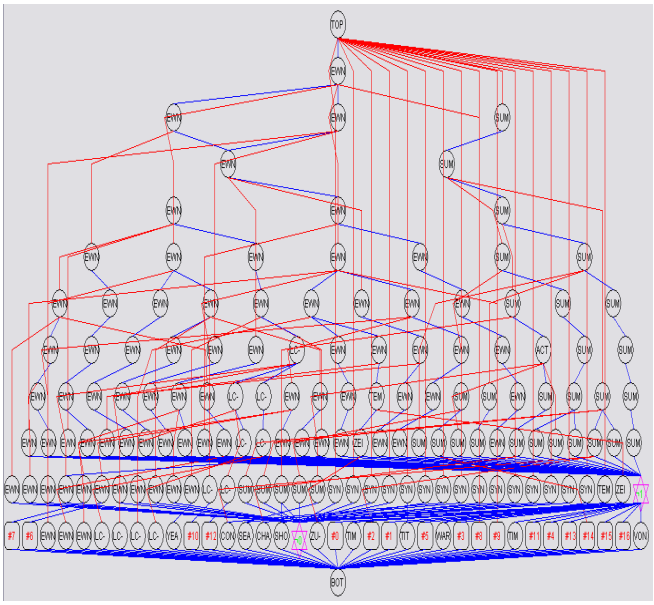

Figure 10: The Bottom to Top display with the autonomous node option

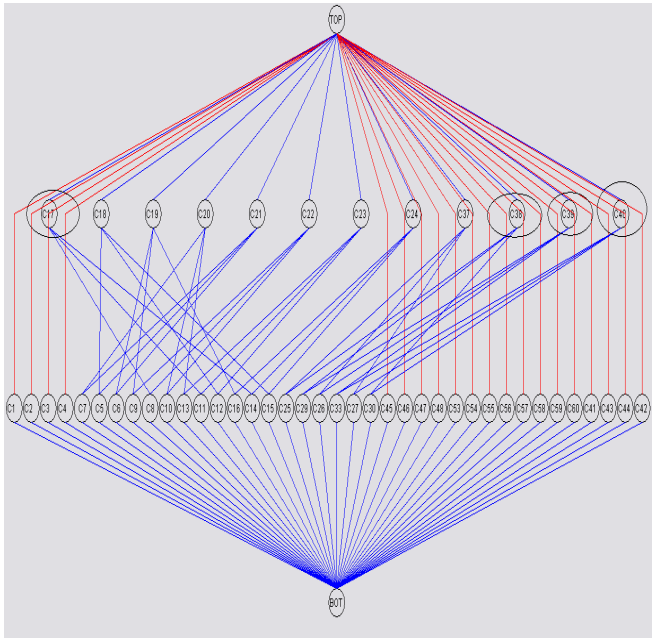

Figure 11: Display using the crossing algorithm. The ellipses show that the bypassed edges still pass through nodes.

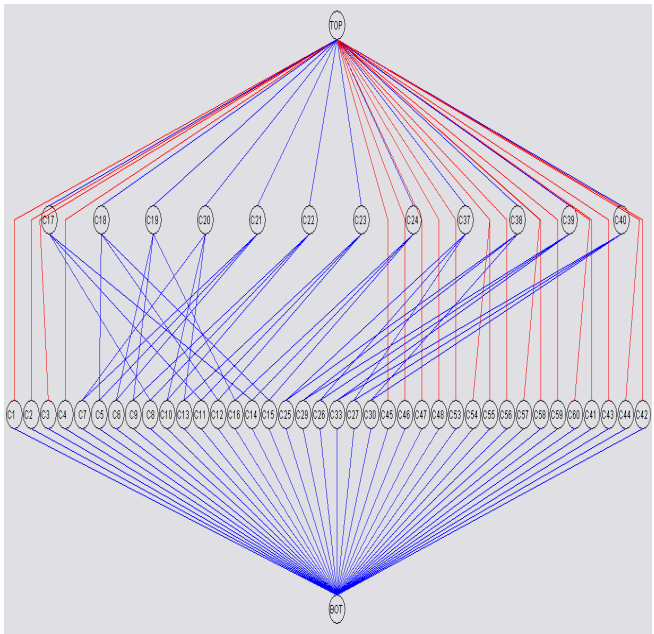

Figure 12: The previous problem is resolved using the modified bypassed edges Layer bifurcation algorithm.

6. Conclusion and Open Problems

In this paper we presented the techniques of visualizing subsumption hierarchies. We have presented two main parts: In the first part we announced an approach for best viewing the data. In the second part we presented the application it self to communicate with the RACER.
Much more work remains to better visualize a large taxonomy file size with a minimum number of edges crossing. This can be an area for future research.
There are still some issues which have not been addressed yet and that we considered as an open problem, here we describe three of them:

1- The data structure is very large.
2- Minimize the crossing number even for the bypassed edges that goes from the top or the bottom.
3- An efficiency search tool.
4-Visualization in 3-D can be explored.

References

[7]: I.Horrocks, U.Sattler, and S.Tobies. “Reasoning with individuals for the description logic SHIQ”. In David MacAllester, editor, Proceedings of the 17th International Conference on automated Deduction (CADE-17), number 1831 in lecture Notes in computer science, Germany, 2000.

[8]: V. Haarslev, and R. Moller. “Racer System Description .” Proceedings of International Joint Conference on Automated Reasoning, Springer-Verlag, June 18-23, 2001, Siena, Italy.

[9]: F. Baader, I. Horrocks, and U. Sattler.”Description Logics as Ontology Languages for the Semantic Web”. Lecture Notes in Artificial Intelligence, 2003.

[12]: G.D. Battista, P. Eades, R. Tamassia, and I.G. Tollis. “Graph Drawing: Algorithms for the Visualisation of Graphs”. Prentice Hall, 1999.

[19]: F. Buschmann. “Pattern-Oriented Software Architecture”. Vol 1. Wiley & Sons, 1996.

[28]: G. Di Battista, P. Eades, R. Tamassia, and I.G Tollis. “Graph drawing, Algorithms for the visualization of graphs”: Chapter 9, 280–290, 1999.

[29]: P. Eades, and N. Wormald. “Edge crossings in drawings of bipartite graphs” Algorithmica. 11(4):379–403, 1994.

[31]: E. Makinen. “Experiments of Drawing 2-leveled Hierarchical Graphs”. Technical Report A-1988-1, Department of computer science, University of Tampere, Jan 1988

[43]: J. Warfield. “Crossing theory and hierarchy mapping”. IEEE Transactions on Systems, Man and Cybernetics, SMC-7(7):502--523, 1977.

[44]: M.J. Carpano. “Automatic display of hierarchized graphs for computer aided decision analysis”. IEEE Trans. on Syst. Man Cybern., SMC-10,no.11,705-715,1980.

[45]: K. Sugiyama, S. Tagawa, and M. Toda. “Methods for Visual Understanding of Hierarchical Systems”. IEEE Trans. Sys Man Cybern., SMC-11,no.2,109-125,1981.