Latour - a tree visualisation system

I. Herman, I. Herman, G. Melançon, M.M. de Ruiter, M. Delest

Information Systems (INS)

INS-R9804 April 30, 1999

Report INS-R9804
ISSN 1386-3681

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Latour — a tree visualisation system

I. Herman, G. Melançon, M.M. de Ruiter
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
email: {Ivan.Herman,Guy.Melancon, Behr.de.Ruiter }@cwi.nl

M. Delest
LaBRI, Université Bordeaux I
351, cours de la Libération, 33405 Talence Cedex, France
email: Maylis.Delest@labri.u-bordeaux.fr

ABSTRACT

This paper presents some of the most important features of a tree visualisation system called Latour, developed for the purposes of information visualisation. This system includes a number of interesting and unique characteristics, for example the provision for visual cues based on complexity metrics on graphs, which represent general principles that, in our view, graph based information visualisation systems should generally offer.

1991 Computing Reviews Classification System: D.2.2, G.2.1, G.2.2, H.5.2, I.3.6, I.3.8.

Keywords and Phrases: information visualisation, tree visualisation, graph visualisation, user interfaces

Note: The work was carried out under the INS3.1 project “Information Visualization”. The on–line version of this report contains the figures in colour.

1. INTRODUCTION

Information visualisation is one of the relatively new areas of research and development in computer science; its fundamental goal, i.e., the ability to visualise and to navigate in large, abstract datastructures, is often regarded as one of the crucial tasks in bringing computers closer to the general public[3]. Visualising graphs plays a very special role in this area, because they can often be used to visualise abstract datastructures. Practical examples include hypermedia structures (like the Web), database query results, or organisational charts of companies. Experimental systems to visualise large graphs have come to the fore in the last years; the NicheWorks system of Wills[23], the *fsviz* system of Carrière and Kazman[4], or daVinci of the University of Bremen[8] are just some typical examples[1]. These systems usually draw on the rich research heritage in the graph drawing community which, over the years, has explored some of the mathematical problems related to graph drawings, like optimal placement algorithms, complexity issues, planarity, etc.[2]. Putting these research results into practice is not a simple task, however. Practical issues raised by, for example, the large size of graphs in information visualisation, the need for navigation and interaction, user interface and ergonomic issues, etc., create new challenges, or cast a new light on well–accepted practices[18]. Consequently, none of the current graph drawing systems could claim to be complete; experiences with these systems are still to be gathered to gain a better understanding of the kind of drawing and navigation facilities which are necessary for a really successful system.

The goal of this paper is to contribute to this “gathering”. It describes an application framework called Latour[2], whose goal is to incorporate interactive graph (primarily tree) visualisation and navigation techniques into other applications. At present, Latour is used or tested as a toolkit to visualise, and to interact with, abstract data for the following applications areas:

---

[1] There are a number of Web site which contain further links to such systems; our site, http://www.cwi.nl/InfoVisu, beyond providing additional information on Latour and the current state of our work, also includes such links.

[2] The reader should not look for a sophisticated abbreviation behind the term “Latour”. Because the framework has been developed in co–peration with the University of Bordeaux I, France, we decided to name the framework after a venerable claret, Chateaux Latour…

- internal data structures of programs (in our case, the internals of a configurable compiler called COSY, produced at ACE b.v., in Amsterdam);
- deployment results of large Petri nets;
- trace information produced by running a massively parallel application based on a co–ordination language;
- result of text and content retrieval systems;
- evolution of genetic algorithms;
- Web site content.

Other application areas are still to come.

While developing this framework, some of the practical problems required more concentrated research efforts, which also led to interesting and general results; these have been presented in separate papers or reports[10,11,16]. The goal of this paper is different; its aim is to describe a number of issues which, albeit not deserving separate articles by themselves, together constitute a body of experiences which we felt is worth sharing with R&D community.

2. GRAPH/TREE LAYOUT

The classic survey of Battista *et al.*[1], which is already 5 years old, lists more than 300 papers, mostly on various graph layout algorithms. New results have been published since this survey, and a recent book by the same authors gives a very comprehensive overview of the field[2]. In spite of all these results on graph drawing, it is not simple to choose a specific algorithm for information visualisation. Information visualisation, which is inherently interactive, raises a number of issues that are not necessarily covered by the classical research on graph drawing. Apart from obvious problems such as speed (in the case of a graph with 3–4000 nodes, the display of the graph should not take more than a second), there remain two important aspects:

- *Predictability*. Two different runs of the algorithm, involving the same or similar graphs, should not lead to radically different visual representation. This is very important if the graph is interactively changed, for example by (temporarily) hiding some nodes or making them visible again. If, as a result of such interaction, the graph drawing algorithm creates a radically different view of the graph, the user will be "lost", and the application may become unusable[1]. Consequently, great care should be taken on which layout algorithm is chosen. For example, a number of graph layout algorithms use optimisation techniques; if the graph changes, a new local minimum may lead to a dramatically different visual representation, which is unacceptable for interactive use.
- *Navigation on large or unusual graphs*. The size of the compiler data structures, related to a simple "Hello World" C program, might contain 50–60 nodes already. Practical applications lead to thousands, or possibly tens of thousands of nodes. To cope with such numbers, navigation tools, search facilities, hierarchical views, etc., are necessary. The implementation of such tools may also require the usage of suboptimal layout algorithms. This should not be considered as a major problem: a rich navigation environment is more important than a pretty layout.

The bulk of the Latour system concentrates on trees, where the usual layout algorithms are quite predictable and fast. Although this section concentrates on tree layout algorithms only, the requirements above should nevertheless be kept in mind for more general graphs, too (see also Section 4).

It was not the goal of Latour to develop new layout algorithms; instead, the goal was to concentrate on the issues raised by data exploration and interaction. Three different tree layout algorithms have been implemented: a (classical) hierarchical view, a radial view, and a so–called "balloon" view. The reason for having these views was simply user demand; various user communities have their own traditions, habits, or requirements, and an application framework cannot impose one single view on its users. In what follows, a short overview of these views will be given. Other layout methods based on cone trees[19], or treemaps[13], could also be considered in future; the modular nature of Latour makes it easy to include new layout algorithms.

[1] The term "preserving the mental model" is also used to describe this requirement, see [15].

2.1. *Hierarchical view*

The hierarchical view of the tree is based on the well–known algorithm of Reingold and Tilford[20] revisited by Walker[22]. The layout algorithm is simple, fast, and completely predictable (in the sense described in the previous section). It has trivial variations, as depicted on Figure 1. All these variations are mathematically identical and implementors may be tempted to include arbitrarily one of these variations only. This would be a mistake: one should recognise that the way of looking at trees may depend on the application areas. For example, the top–down grid view is the widespread way of looking at family trees, whereas biological evolution schemas often use a left–to–right grid. The conclusion for an application developer is simple, albeit important: give the user the choice; he/she should be able to choose among the different views (or customise the range of available views for a specific application area).

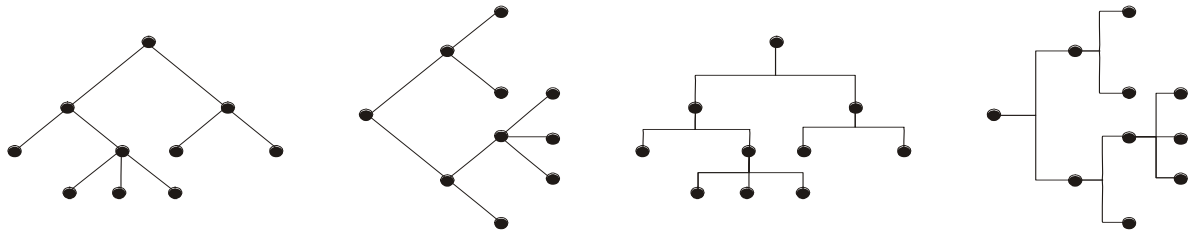

Figure 1 Different hierarchical views of a tree

2.2. *Radial view*

The radial view (see Figure 2) is based on an algorithm described in Eades[7] (see also in di Battista *et al*[2]). This algorithm recursively places the children of a subtree into circular wedges; the central angle of these wedges are proportional to the width of the respective subtrees, i.e., the number of leaves. If this was the only layout rule, additional edge intersections would occur if the angle on the node became too large; to avoid this, a "convexity constraint" is introduced which, essentially, forces the wedge to remain convex. Such, or similar, view is favoured, for example, by some web site viewers, which do not want to overemphasise the role of a root.

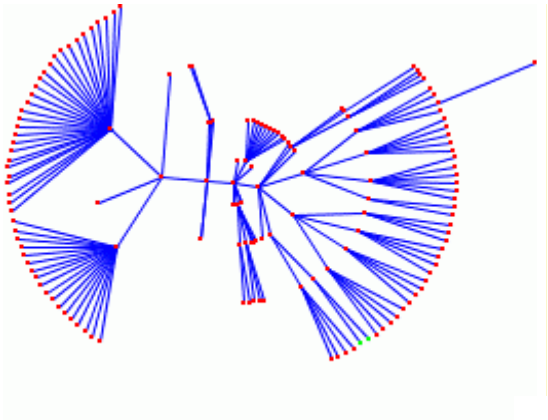

Figure 2 Radial view with convexity check

The algorithm is very simple, but it is not optimal in using the available space (this can clearly be seen on the figure). We spent some time in trying to optimise the algorithm. The idea was to use the statistical distribution of the width of a subtree at a node, which can be approximated with a normal distribution (see the paper of Drmota[6]). Using this distribution one can "predict" whether a subtree is excessively large or narrow, and one can therefore modify the default re–distribution of a wedge at a node. If a subtree is "large", which means that it has, statistically, many leaves compared to its size, it gets a higher share of the wedge. If, conversely, the number of leaves is unusually low compared to its size, its share is reduced. (For details of the statistics, the reader should consult [10] where these formulae were used for other purposes). The improvements were not significant, however; this turned out to be the consequence of the relative "strength" of the convexity constraint whose effect seems to dominate other optimisation attempts.

A possibility to overcome this problem is to simply drop the convexity. Although this is not mathematically correct, the occurrence of extra intersections is not very frequent after all. On the other hand, the image fills the available space much better. Figure 3 shows the same tree as on Figure 2 with the statistically improved radial placement but without the convexity check. This shows that, in some cases, it is not necessary to look for a mathematically perfect algorithm for a graph layout; the mathematical "faults" may not be significant in practice. The weaknesses (like the extra intersections) of the algorithm become apparent only if very big trees are used, where the few extra intersec-

tions are not really disturbing any more (the image is very complex anyway). Problems with navigation, zooming, etc. (see the next section) should become predominant in that case, and it is not really worth to optimise the layout any further. A fast, better looking, albeit mathematically incorrect algorithm might sometimes be the good choice after all. (It is interesting to note that, for example, the NicheWorks system of Wills[23] uses a similar radial algorithm, but *without* the convexity constraint; the same seems to be true for [5].)

For the sake of completeness, we decided to include both the optimal (i.e., with convexity check) and the, shall we say, sub–optimal radial layout algorithm into Latour.

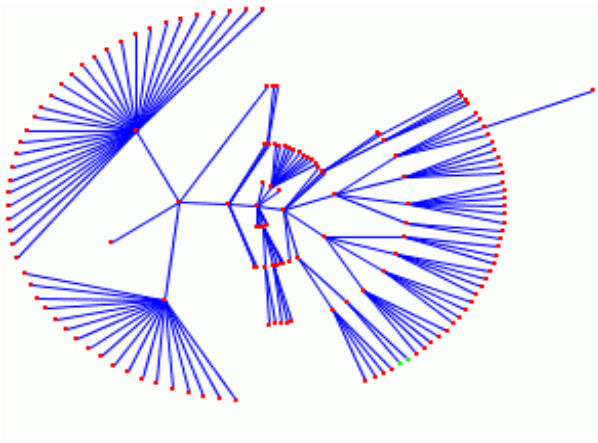

Figure 3 Radial view without convexity check and with statistical modifications

2.3. Balloon view

The request for a "balloon" view (see Figure 4) came from an application dealing with the retrieval of keywords and their relations from a database. The notion of a "root" is temporary for such application: the user should be able to move from one node to the other interactively, and the tree on the screen should reflect the relationships using this temporary focus. Neither the hierarchical nor the radial view was seen as appropriate; both emphasise the role of the root in a way that was seen to be misleading for the users. The balloon view seems to fulfil these needs.

The balloon view is the only layout algorithm, which has been developed during the work on Latour. The detailed explanation of the algorithm would go beyond the scope of this paper; the interested reader should consult a separate report on the subject[16]. The balloon view gives satisfactory results for well–balanced trees; other placement algorithms (for example, by drawing the projection of a spatial cone tree placement, see, for example the approach used in *fsviz* and described in [4]) could also be used. The important issue to remember at this point is the necessity for a "re–root" facility: i.e., that the user can interactively pick an arbitrary node on the screen, and reorganise the full tree with the newly picked node as a root. This ability of Latour is essential for large classes of applications.

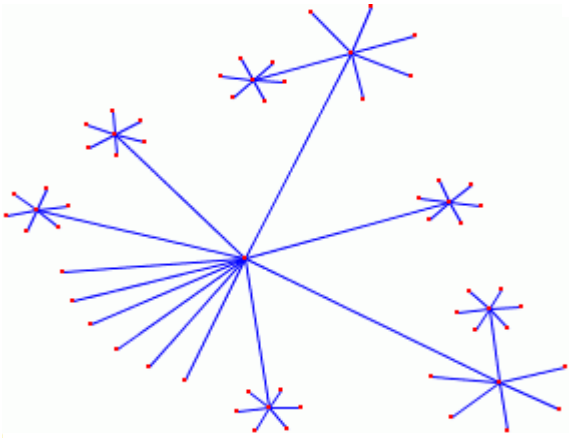

Figure 4 Balloon view

3. Interaction and Navigation

Information visualisation is an inherently interactive application; the user has to move around in information space, explore details, hide unnecessary parts of a tree, etc. Obviously, a good system must offer a whole range of tools in order to make the exploration of a graph easy, or indeed possible.

3.1. Zoom, pan, fish–eye

Some of the techniques, implemented in Latour now are standard: zoom, pan, fish–eye (the latter based on the paper of Sarkar and Brown[21]). As much as possible, the factors controlling these effects (e.g., the distortion factor of the fish–eye view) are settable interactively by the end–user.

The fish–eye view has one drawback, though, which implementors should be aware of. The essence of a fish–eye view is to distort the position of each node, using a convex function applied on the distance between the focal point and the node's position. The function can be a relatively simple rational polynomial. However, if the distortion were to be applied faithfully, the edges connecting the nodes should be distorted, too. Mathematically, the result of

this distortion is a general curve. Usual graphics systems (e.g., the Java's AWT or Java2D packages) do not offer the necessary facilities to transform lines into these curves easily (they can be, mathematically, fairly complex). The implementer's only choice is, therefore, to approximate the original line segments with a high number of points, transform those points, and display a polyline to approximate the ideal, transformed curve. The problem is that the number of approximating points must be relatively high if a smooth impression is sought (on average 60 points per edge), which leads to a prohibitively large amount of calculation and make the responsiveness of the system sink to an unacceptably low level. The only viable solution is to apply the fish–eye distortion on the node co–ordinates only, and to connect the transformed nodes by straight–line edges (this is the approach taken by Sarkar and Brown, too[21]). The consequence of this inexact solution is that new edge intersections might occur, for example when the radial or the balloon views are used. Though inelegant, this brute force approach did not prove to be disturbing in practice.

The approach of Sarkar and Brown might be characterised as an "image space algorithm", to borrow a terminology from computer graphics: the distortion occurs once the geometric positions of the nodes are already determined by the layout algorithm. Another form of distortion may be achieved by controlling the parameters governing the layout itself, thereby producing a fish–eye like distortion. For example, in the case of the balloon view (see Section 2.3), one can interactively "inflate" a circle determined by a particular node; the algorithm would automatically adjust (i.e., deflate) the other circles, yielding a fish–eye view like image. Not all layout algorithms are appropriate for such control, though.

3.2. Complexity visual cues

The well–known problem in using zoom and pan is that the user looses the "context". This is why fish–eye view is used: it provides a "focus+context"[14] view of the tree. However, when the tree is large, zoom and pan cannot be avoided and other techniques become necessary, too. A unique feature of Latour is a technique to provide visual cues based on the structural complexity of the tree. This technique works as follows.

A so–called "metric" value is calculated for each node of the tree. This metric should represent the complexity of the subtree stemming at the node. Several different metric functions are possible and, ultimately, the choice among these should be application dependent. Examples for such metric include the width of the subtree (i.e., the number of leaves), the sum of the lengths of all paths between the node and the leaves of the subtree, the so–called Strahler numbers derived from a complexity measure widely used in combinatorics (see [10]), or the "degree of interest" function used by Furnas[9]. These (structural) complexity metric values can be controlled further by assigning an application dependent weight to each node (either interactively, or via the background application); this weight is then taken into consideration, too, when the final complexity values are calculated.

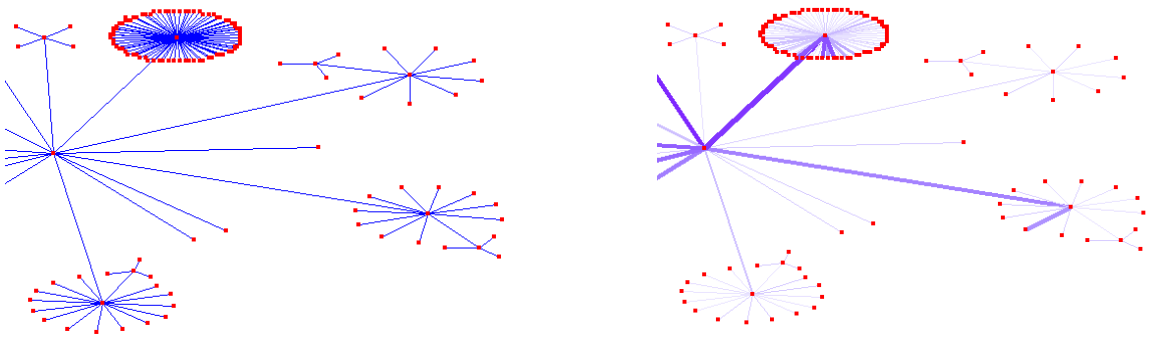

Figure 5 Effects of complexity cues

Using these metric values, and visual tools like colour saturation, linewidth, etc, Latour can highlight the "backbone" of a tree, i.e., those edges which hold larger, more complex subtrees. The effect is clearly visible on Figure 5. The image on the left is a simple zoomed–in image: the user barely knows where to move with the pan, if complex areas are searched; it is also not clear whether the node on the left which looks like a root node is indeed the root of the tree or not. The right–hand image shows the same portion of the graph, with the metric based visual cue. It

clearly shows, for example, that the node on the left is indeed the root and that one of the edges going toward the left leads to a complex portion of the tree, whereas the other one is probably less interesting.

Another possible usage of the metric numbers is presented on Figure 6: this is the so–called *schematic view* of a tree. Based on the complexity metrics of the nodes, Latour displays, in a "traditional" graph form, only those nodes whose metric value is greater than a specific cut–off, yielding what we have called the *skeleton* of the tree. All other nodes are encapsulated in schematic "shapes" (triangles in this case), whose size and geometry is proportional to the hidden portion of the tree. Obviously, the cut–off value can be controlled interactively by the user. The result is a better overall view of the tree which, combined with other navigation techniques, provides a powerful interactive tool to the user to explore the graph. This schematic view approach bears some resemblance with the so–called generalised fish–eye views of Furnas[9] although our schematic views can be applied to any metric and gives a general view of the whole tree instead of focusing on one node.

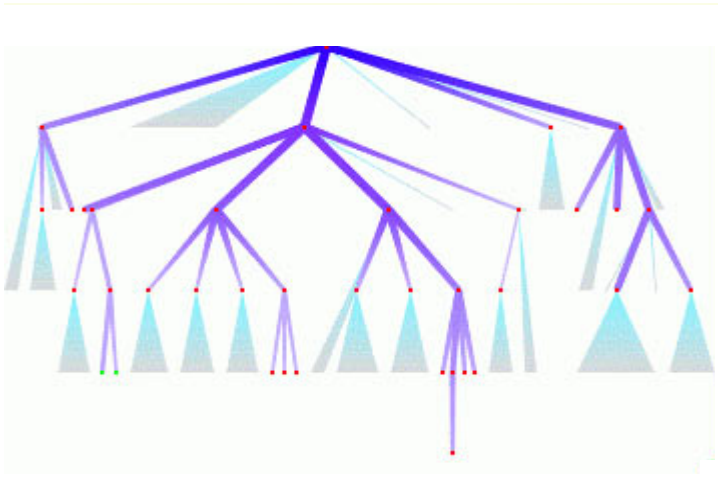

Figure 6 Schematic view of a tree

It is worth noting that, although all our examples so far were for trees, the visual cue techniques based on a complexity metric represent a general principle which can be applied for more general graphs, too; the interested reader should refer to [11].

3.3. Animation

Latour is an interactive system; the user navigates in different portions of the tree, zooms, pans, etc. Some of these actions result in an immediate, real–time feedback (for example, zoom), some other actions may lead to a more radical reorganisation of the screen (for example, folding a subtree into a node, or unfolding a folded subtree). These steps may be notoriously disturbing for the user, who may easily loose track, forcing him/her to "relocate" on the screen searching, for example, for a particular node. To reduce this problem, Latour gracefully animates all possible changes from one view to the other, avoiding any radical changes as far as possible. The animation step itself is straightforward: for example, if the layout algorithm is changed, each node "moves" along a linear path to its new position, creating a movie–like effect (other systems have also adopted the similar approaches; see, for example, the paper of Huang *et al*[12]).

Although originally only included in Latour to reduce possible ergonomic problems, this basic animation feature turned out to be a very useful tool for various applications, too. There are indeed applications whose goal is to explore a sequence of trees, instead of a single one, often representing the evolution of data in time. This is the case, for example, of the application exploring genetic algorithms, or the traces of parallel program runs. Therefore, the input possibilities of Latour have been extended: it can not only accept the description of a single tree, but also a "generation" of trees, i.e., a basic tree plus a sequence of difference trees. This sequence of trees can then be visualised systematically with again a graceful animation at each change.

3.4. Miscellaneous techniques

Latour offers some other interaction tools, too, which are worth mentioning without going into further details:

- interactive tools to fold/unfold a subtree at a node (note that the predictability of the layout is of an utmost importance for this);
- ability to visualise a subtree in a separate, pop–up window;

- “non–geometric” navigation tools, like a search on the name or each node, or on a set of application–dependent attributes which can be assigned to each node;
- re–root of the tree (this has already been mentioned, in relation to the balloon view).

4. BEYOND TREES

Latour is primarily a tree visualisation tool but, obviously, applications may want to handle more general structures, too. Although we are currently working on the implementation of a more general tool, some extensions have been added to Latour already. These added features are interesting because they show that, through a moderate amount of extra work, a tree visualisation system may become useful for a larger class of applications, too.

4.1. Packed forests

Packed forests are, in fact, special data structures. The need for these data structures have arisen through the application concerned with the visualisation of the internal data structures of compilers, but has proven to be useful in general, too. (The name for the structure originates from its usage in computational linguistics.)

Instead of giving an abstract definition, the concept is presented through an example. For a compiler, the standard internal representation of a string is a list. The leaves of the list represent the individual characters of the string, and intermediate nodes are used to build up a list structure. Such list can be represented as a simple tree, like the left–hand one Figure 7.

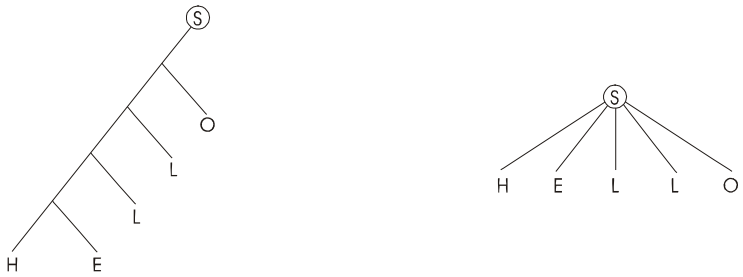

Figure 7 Example of a packed forest

However, such a representation may be too “verbose”. An expert in compiler technology knows the internal representation for a string and does not necessarily need the full list version of the relevant portion of the graph; the tree on the right–hand side of Figure 7 is enough to convey all the necessary information. What the user wants is to be able to “switch” between the two representations, the two “alternatives” (as they are called within Latour), interactively. A switch between the two alternatives means, mathematically, to change between two trees differing in the subtree of a specific node only. Latour has the possibility to store, internally, a set of such alternatives for each node, and offers interactive means to switch among those. In effect, what Latour stores as a data structure is a forest packed in one entity.

Managing packed forests require the adaptation of the navigation tools, the visual cues, and other interactive facilities. For example, if a node has several stored alternatives, additional metric values can be calculated representing the average of the complexity values for each alternative taken individually (of course, these calculations should be performed recursively). Non–geometric navigation tools are also provided: for example, an attribute can be assigned to each alternative in the forest, and the changes among alternatives can be controlled globally (a simple example might be to change from, say, ‘expert’ mode to ‘novice’ mode, and make a global change from the succinct string representation to the more verbose one, where applicable).

Packed forests turned out to be extremely useful in practice. As a slightly extreme example, some of the demonstration graphs used by our compiler builder partner is, initially, a tree consisting of 2–3 nodes only. However, when the same graph has all its most complex alternatives extended, it turns into a tree of about 100 nodes. Similar data structures are used routinely in computational linguistics; the concept of “level of details”, of an utmost importance in virtual reality scenes, is another example which can be represented through these structures. In other words, packed forests provide a very efficient, and application dependent, way of imposing a manageable hierarchy on the visualised data structures, hence their importance.

4.2. Dag's

Dag's (Directed Acyclic Graphs) represent the next logical step when trying to generalise from trees. This is achieved by a simple extension of Latour, which allows the storage of additional links ("stepchildren" and "ancestor" links) for each node of the underlying tree. This means, mathematically, that a spanning tree is provided, and Latour uses its tree–related structure to visualise the dag by simply adding the additional links to the tree picture. The spanning tree may have two origins: either the application generates it, or the spanning is tree is calculated for the dag.

4.2.1. Spanning trees provided by the application

Requesting the application to generate a spanning tree is not such a strong requirement as one might think. For a number of applications, there *is* an inherent tree structure in the data, and visualising this tree, with the additional edges added to the tree, yields a natural representation of the dag. Munzner, for example, argues in her paper[17] that a large number of Web sites do have an inherent tree structure, and a Web visualiser should take advantage of this. Although this might not be true for all Web sites, our experiences concur with hers for a large number of cases. Figure 8, which indeed represents the structure of a (small) Web site, differs from Figure 2 by having some additional edges added to the picture. These additional edges do not represent any difficulties in navigating through the graph.

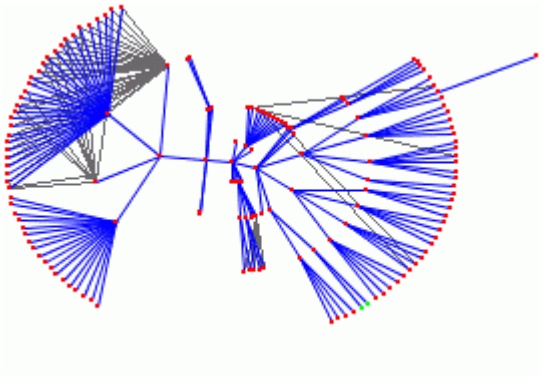

Figure 8 A tree with added links

Figure 9 shows another example where a spanning tree is used to visualise a dag. The interesting feature is that the spanning tree consists of three branches and all "non–tree" edges are used to connect these branches. We can refer to such graphs as "multipartite" trees. Similar, bipartite trees occur when describing virtual reality scenes, for example (where one branch describe an object hierarchy, the other the real instances). These "multipartite" graphs occur frequently in applications, and constitute a set of examples where the simple extension of Latour works out very well in practice.

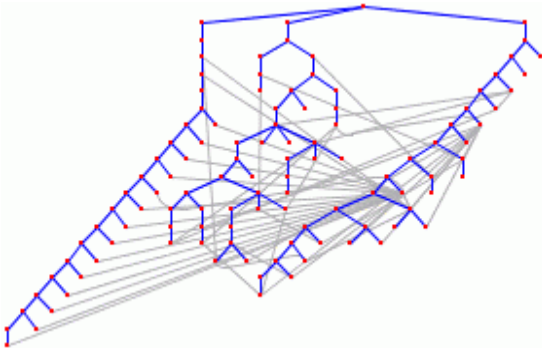

Figure 9 A tripartite tree

4.2.2. Automatic generation of spanning trees

Relying on a spanning tree to layout a large graph is common practice and is indeed often a necessity in situations where interactive time response is critical. Automatic computation of a spanning tree can be done in many different ways, mainly because of the high degree of liberty one has for computing one. Our main concern was to compute a spanning tree that would lead to a best layout for a dag, based on the choice of a spanning tree.

Many spanning tree algorithms try to find a tree, which is optimal with respect to a certain criterion, such as minimising the sum of weights of the nodes in the tree. To achieve this, one obvious way is to consider all the neighbours of a node already in the tree, and to choose among those the one with the corresponding minimal edge. By repeating this process iteratively, a spanning tree is generated.

Nodes of a dag are implicitly assigned a layer number, namely the maximal length of a path connecting it with a top node of the dag. Using this value as a weight, and applying the simple algorithm described above, we were able to produce spanning trees for dag's in an efficient and simple way.

4.2.3. Adapting the tree layout to dag's

The Reingold and Tilford algorithm makes use of the depth of nodes in the tree; all nodes with equal depth will actually appear on a horizontal line on the screen. We had to adapt this feature of the R&T algorithm so that a node

would be placed on its proper layer in the dag. Indeed, in most cases the depth of a node in the spanning tree does not coincide with its layer number. To handle this problem, we insert dummy nodes in the spanning tree, i.e., nodes that are used by the R&T algorithm, but are not displayed as nodes on the final image. Incidentally, as test cases we used dag's extracted from graphs submitted at Graph Drawing contests. Surprisingly enough, the layouts we obtained compared quite well with the winner layouts. Figure 10 shows an example based on sample B of the GD '95 contest.

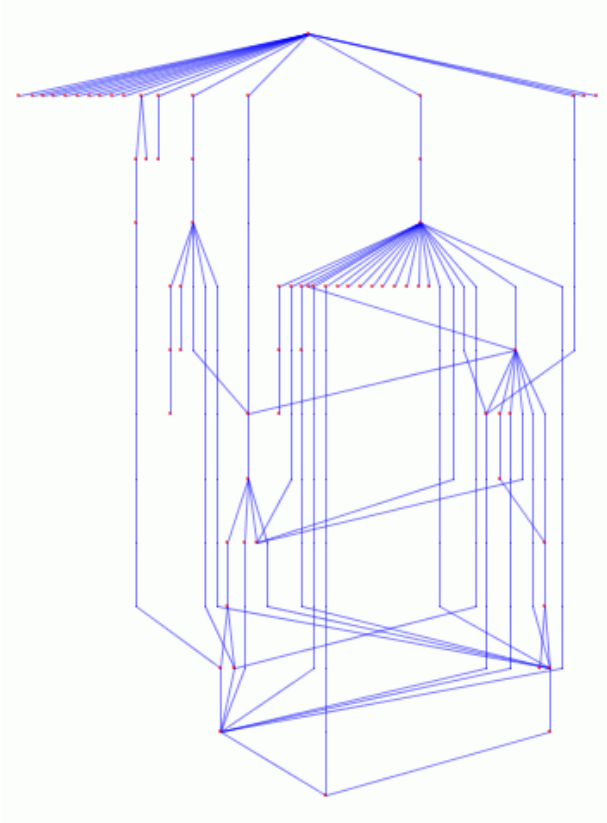

Figure 10 A dag drawn with a generated spanning tree

5. Implementation

Latour has been implemented as a stand alone Java application. The reason of choosing Java was to achieve the highest possible portability among various Unix platforms, Windows and Macintosh. The famous slogan of Sun "write once, run everywhere" did not quite work out in practice: when porting the system from the original Unix development platform to Windows NT, the behaviour of the user interface (based on the AWT toolkit) was slightly different (mainly due to a small difference in handling mouse), which required some further testing and adjustments. Nevertheless, porting was indeed a matter of a few days only and we routinely work today on different platforms without problems. Such a level of portability would have been much more difficult to achieve, had we used C++, for example.

Beyond portability, Java offered some additional tools, which have proven to be extremely useful. It was our goal to provide a very high level of flexibility in using Latour, to be able to adapt it to the end user's needs and taste easily, and that such adaptation should also be doable by any expert user. The dynamic loading facilities of Java have proven to be of invaluable help in this respect: by providing the *name* of a class, a Java program can easily load a new class and use it as if it was part of the original distribution. This means that the user of Latour can write his/her own class implementations for specific aspects of the system, use a property mechanism to convey the name of these classes to a Latour system already executing, and the system will use these classes instead of the default ones. Of course, a proper specification of some standard super–classes was necessary, but this was routine work. Here are some aspects of Latour, which can be adapted by the user:

- colouring of edges, nodes;
- display of individual nodes (replacing the node by an icon, controlling the appearance of the node's name, etc.);
- details of the animations (speed, linear or not linear, etc.);
- tree layout algorithms;
- application–dependent complexity metric functions;
- global control of alternatives in a packed forest.

The class containing the final menus can also be overridden by the user; for example, a sub–set of every metrics in the system can be chosen, hiding those that are irrelevant for a specific application area. Moreover, most of these aspects can be controlled on a graph by graph basis, too: if a certain graph is better adapted to a balloon view, for example, than this view can be set as its default. This kind of flexibility makes it quite easy to incorporate Latour into new applications; this is why Latour can be considered as a framework rather than a single application.

Of course, not everything was easy with Java. The biggest problem we have today is the disappointingly bad performance of the Java2D implementation in Java 1.2. Although the features offered by Java2D, which provide a much better control over the final outlook of the image, would be extremely valuable for us, it is way too slow for interactive use. We are therefore forced to continue using the basic graphics of Java 1.1, in the hope that a next release of Java 1.2 will overcome this problem.

6. CONCLUSIONS

The implementation of Latour has resulted in a very flexible system, which is well adaptable to various user communities. It concentrates on interaction and visual feed–back, rather than complicated layout algorithms, which makes it one of its strengths. It has also taught us some important lessons: that a proper balance has to be found between the mathematical correctness and the requirements of navigation and interaction, that the end–user has to have a maximal control over the appearance and the attributes of the visual representation, we learned about the importance of metric functions on graphs in general. In developing a more general graph–based information visualisation framework these experiences will become of an utmost importance.

ACKNOWLEDGEMENTS

The development of Latour was based partly on a co–operation within CWI and the University of Bordeaux, within the framework of the Franco–Dutch intergovernmental agreement “van Gogh”, and partly on a co–operation with ACE b.v., Amsterdam and with other research groups at CWI, within the framework of the Dutch national funding “High Performance Visualisation”. Let us thank Job Ganzevoort (ACE), Farhad Arbab and Scott Marshall (CWI), and Jean–Philippe Domenger (LaBRI) for their valuable comments. Finally, our thanks to Peter Eades (Newcastle, Australia) for pointing at some recent results in the area of tree placement; he has saved us some unnecessary detours.

REFERENCES

1. G. di Battista, P. Eades, R. Tamassia, and I.G. Tollis: “Algorithms for drawing graphs: an annotated bibliography”. In: *Computational Geometry: Theory and Applications*, **4**(5), 1994.
2. G. di Battista, P. Eades, R. Tamassia and I.G. Tollis*: Graph drawing: algorithms for the visualisation of graphs*, Prentice Hall, 1999.
3. S.K. Card, J.D. Mackinlay, B. Shneiderman (eds): *Readings in Information Visualization*. Morgan Kaufmann Publishers, 1999.
4. J. Carrière, R. Kazman: “Interacting with huge trees: beyond cone trees”. In: *Proceedings IEEE Information Visualisation '95*, IEEE CS Press, 1995.
5. M.C. Chuah: “Dynamic Aggregation with circular visual designs”. In*: Proceedings of the IEEE Symposium on Information Visualisation (InfoVis'98)*, G. Wills and J. Dill (eds.), IEEE CS Press, 1998.
6. M. Drmota: “Systems of functional equations”. In: *J. Random Structures and Algorithms*, **10**(1–2), 1997.
7. P. Eades: “Drawing free trees”. In: *Bulletin of the Institute for Combinatorics and its Applications*, **5**, 1992.
8. M. Fröhlich, M. Werner: “Demonstration of the interactive graph visualization system daVinci”. In: *Proceedings of DIMACS Workshop on Graph Drawing '94*, R. Tamassia, I. Tollis (eds.), Lecture Notes in Computer Science No. 894, Springer Verlag, 1995. (See also the project's current homepage for the current status: http://www.informatik.uni-bremen.de/~davinci/).
9. G.W. Furnas: “Generalized fisheye views”. In: *Proceedings of the ACM CHI’86 Conference*, ACM Press, 1986.
10. I. Herman, M. Delest and G. Melançon: “Tree visualisation and navigation clues for information visualisation”. In: *Computer Graphics Forum*, **17**(2), 1998.
11. I. Herman, S.M. Marshall, G. Melançon, D.J. Duke, M. Delest, J.P. Domenger: “Skeletal images as visual cues for graph visualisation". In: *Proceedings of the Joint Eurographics–IEEE TCCG Symposium on Visualization*, Vienna, eds. E. Gröller and W. Ribarsky, Springer Verlag, Wien, 1999. Also in: *Reports of the Centre for Mathematics and Computer Sciences (CWI), INS–R9813*, ftp://ftp.cwi.nl/pub/CWIreports/INS/INS-R9813.ps.Z, 1998.
12. M.L. Huang, P. Eades, and R.F. Cohen: “WebOFDAV — navigating and visualizing the Web on–line with animated context swapping”. In*: Computer Networks and ISDN Systems (Proceedings of the 7th World Wide Web Conference)*, **30**(1-7), 1998.
13. B. Johnson and B. Schneiderman: “Tree–maps: a space–filling approach to the visualisation of hierarchical information structures”. In: *Proceedings of IEEE Visualisation ’91*, IEEE CS Press, 1991.
14. J. Lamping, R. Rao, and P. Pirolli: “A focus+context technique based on hyperbolic geometry for viewing large hierarchies”. In: *Proceedings of the ACM CHI'95 Conference*, ACM Press, 1995.

15. K. Misue, P. Eades, W. Lai, and K. Sugiyama: “Layout adjustment and the mental map”, *Journal of Visual Languages and Computing*, **6**, 1995.
16. G. Melançon, I. Herman: “Circular drawing of rooted trees”. *Reports of the Centre for Mathematics and Computer Sciences (CWI), INS–R9817*, http://www.cwi.nl/InfoVisu/Papers/circular.pdf, 1998.
17. T. Munzner: “H3: Laying out large directed graphs in 3D hyperbolic space”. In*: Proceedings of the 1997 IEEE Symposium on Information Visualization*, IEEE CS Press, 1997.
18. H. Purchase: “Which Aesthetic has the Greatest Effect on Human Understanding?”, In: *5th International Symposium, Graph Drawing ‘97*, Rome, Italy, Lectures Notes in Computer Science 1353, Springer Verlag, 1997.
19. G.G. Robertson, J.D. Mackinlay, and S.K. Card: “Cone trees: animated 3D visualizations of hierarchical information”. In: *Proceedings of the ACM SIGSHI Conference on Human Factors in Computing Systems*, ACM Press, 1992.
20. E.M. Reingold, J.S. Tilford: “Tidier drawing of trees”. In: *IEEE Transactions on Software Engineering*, **SE–7**(2), 1981.
21. M. Sarkar, M.H. Brown: “Graphical fisheye views”. In: *Communication of the ACM*, **37**(12), 1994.
22. J.Q. Walker II: “A node-positioning algorithm for general trees”. In*: Software ── Practice and Experience*, **20**(7), 1990.
23. G.J. Wills: “Niche Works ── interactive visualization of very large graphs”. In*: 5th International Symposium, Graph Drawing ‘97*, Rome, Italy, Lectures Notes in Computer Science 1353, Springer Verlag, 1997.