Three Dimensional UML Using Force Directed Layout

Tim Dwyer

Basser Department of Computer Science
Madsen Building F09
University of Sydney
NSW 2006
Australia
Email: dwyer@cs.usyd.edu.au

Abstract

There is evidence to suggest that three dimensional representations of connected graphs have a number of advantages in conveying information to users over their two dimensional counterparts. In this paper we explore the use of a Force Directed Algorithm to layout three dimensional UML Class diagrams representing the architecture of object oriented software systems. We describe a simple evaluation and usability study conducted on the proposed system and present our findings which indicate some unexpected benefits to a user's perception of the visualised architectural structure.

Keywords: UML, Force Directed Algorithm, Three Dimensional, Software Modelling

1 Introduction

One of the major problems facing software engineers is that software can be incredibly complex, often involving millions of lines of code in a single system, and since the 'workings' of the software are ethereal electronic signals that occur in nanoseconds the operation of the software is completely invisible [Mellor, 1994]. We can look at the source code to a large system one editor windowful at a time but this only shows us the extreme detail and tells us little about the operation of the system as a whole. We are like doctors trying to understand the workings of the entire human body by looking only at individual cells through a microscope. The key to understanding such a complex system is being able to view it at different levels of abstraction, to find 'meta-structure' or patterns from minute structure.

What we need are tools to help us visualise software at every level. We need the X-Rays, CAT Scans, biopsies, blood tests, stethoscope and a pair of good hands to feel for lumps. This is the inspiration for this paper, to investigate the technologies necessary to develop a new generation of tools for the visualisation of software.

This situation is recognised in the software engineering field and a number of tools and methods for modelling, visualising and communicating software have been in use since it began. The most prominent in recent years has been the Unified Modelling Language, UML. In a medium used for communicating artifacts as complex and critical as software designs a widely used standard makes the engineer's job much easier. UML goes a long way towards being a complete visual language for modelling software however, most tools for manipulating UML models present it only in the form of 2 Dimensional (2D) diagrams. This mode of visualisation has a number of limitations that are discussed below.

UML is a very large language and provides several different *views* of a system model, holistically capturing every aspect of the system's design and dynamic execution. In this work we will consider primarily the diagrams from the *Design View* which show *static structure*, or the relationships between elements of the system[Rumbaugh et al., 1999]. We examine how the Class and Object diagrams that make up this view can be better understood through 3D visualisation. It should be noted however that the same concepts can be extended to many other parts of UML.

Now we pose some of the questions which must be asked when considering this new paradigm for UML diagrams:

2 Is Modelling Software in 3D Better than in 2D?

This question has been asked at least once before regarding UML[Gogolla et al., 1999] and speculation about the effectiveness of 3D visualisations of other types of informational graphs has been rampant, most notably [Ware and Franck, 1994, Kumar and Fowler, 1994, Reiss, 1994, Cohen et al., 1994]. This section, therefore, will describe briefly what has been said in the past on this topic. Later sections will comment on the correlations between this prior work and my own study in the specific domain of UML.

It has been claimed[Rumbaugh et al., 1999], with good supporting evidence, that graphical methods for describing software designs highlights the structure and relationships between software elements. However, there are also difficulties in visualising such designs graphically:

- As systems grow in complexity their diagrammatic representations also expand, sometimes exponentially, making comprehension of the overall structure of the software more difficult.
- As diagrams grow in size and complexity optimal layout becomes increasingly difficult and time-consuming to calculate and problems like trying to remove edge crossing become intractable, even for automated algorithms. These problems belong to the domain of the Graph Drawing field, whose definitive text is [Battista et al., 1999].
- In hierarchical or layered structures, components in the same layer must be displayed at the same level in the hierarchy. In 2D this implies a linear arrangement for each level which rapidly consumes space. In 3D such levels can be laid out on planes.

[Ware and Franck, 1994] express the amount of information that can be perceived in a given n-dimensional display using the notation: I_{nD}. Using this notation they express the relationship between the amount of information that can be displayed linearly (in a 1-dimensional space) against that of a 2-dimensional space as:

$$I_{2D} = (I_{1D})^2$$

They go on to suggest that following the same logic the increase in information that can be perceived in 3D over 2D is:

$$I_{3D} = (I_{2D})^{3/2}$$

However, they concede that since in reality we view a 3D model with only two relatively close set eyes this relationship is probably closer to:

$$I_{3D} = C \times I_{2D}$$

where C is a constant for which they obtained a value of roughly 3 from experimentation on human subjects. [1]

The indication is that subjects were able to perform tasks requiring good comprehension of a graph's structure better when the graphs were presented to them in 3D than when in 2D. Taking this evidence we ask how we can best utilise the 3rd dimension for software design and analysis models.

2.1 How are UML models best automatically arranged in 3D?

Traditional tools for drawing UML diagrams, such as Rational Rose (`http://www.rational.com`), require that the diagrams be laid out by hand. Manually trying to find the best arrangement of elements in a diagram, both aesthetically and to clearly display the design, is not a very good use of the designer's time. Having to do so in 3D, especially if state of the art 3D interfaces such as data-gloves and stereo headsets are not available, would be impractical. So an automatic layout method is clearly necessary to free the designer to think about more important issues without worrying about presentation. [2]

One study of the effectiveness of the most popular layout algorithms for conveying information to humans [Purchase, 1998] found that force-directed algorithms (FDAs) were at least as effective as any other. FDAs, which were introduced into graph drawing by [Eades, 1984], use heuristics based on balancing a set of simulated physical forces between the elements of the graph to find optimal layout. They have several other advantages for a 3D UML visualisation system:

- It was found that FDAs scale up to 3D space well, simply by replacing 2D vector arithmetic in the conventional algorithms with 3D vector arithmetic.
- FDAs simulate forces found in nature, that is, in the 3D physical world in which we live. For example the 3D FDA discussed in this paper uses approximations of natural laws, Hooke's spring law and Newton's gravitation law. Therefore, it is possible to assume that they are quite familiar, natural and comprehensible to the viewer when used to layout models. If this is true then animation dictated by natural, predictable physical forces should help to preserve the user's 'mental map'[Misue et al., 1995] of the graph after layout changes.
- FDAs work very well to remove edge crossings in 3D. Edges can pass each other from in front or from behind as well as from above and below. This is one of the key criteria for effective graph drawing as proposed by [Battista et al., 1999].
- UML diagrams can have both directed and undirected edges. Force directed algorithms can work both, see 4.3.
- The Layout can be used to automatically suggest the semantics of the model. We discuss this further in Section 5.

[1] This value was found when stereo vision and hand-coupled animation (the ability to rotate the model using a mouse) was available. Lower, but still significant, values of C were found when such facilities were not available.

[2] Other speculations on the use of UML in 3D have not addressed this issue or have left it as further work: for example [Gogolla et al., 1999] built static 3D UML models in VRML simply as a proof of concept. Therefore, the choice and design of such a method is considered here from first principles.

3 Abstraction and Clustering

UML Class diagrams support the concept of a 'package' or a collection of classes grouped logically into a sub-diagram. This concept is important when modelling a large system since atomic entities in the system can be abstracted into packages and the model can be viewed at a courser granularity. Thus it becomes possible to study the higher level interactions between groups of classes when previously they may have been lost in the detail.

In graph theoretic terminology such groupings of related elements in a graph are called 'clusters'. The graph engine developed as part of this study supports clustering through the use of transparent spheres and the 'Origin Force' described below.

Another study [Ware et al., 1997] also visualised nested software systems in 3D. Our approach differs as follows:

- We translate UML notation to 3D to provide a more familiar paradigm for software engineers and hence ease the transition to 3D
- We use a more "Physically correct" layout model which we feel is more intuitive
- We enclose clusters in transparent spheres rather than boxes. [Wiss and Carr, 1999] found that the inefficient use of space in nested 'box' style visualisations slowed user's response times in their usability study.

4 Force Directed Layout Algorithm For UML Class and Object Models

Force Directed Algorithms (FDA), also known as Spring Algorithms or Spring Embedded Algorithms, work by simulating repulsive and attractive forces between the elements of a graph so that their layout must reach a state that minimises these forces. Many approaches to finding this 'minimal energy' state have been applied to FDA's. Methods from Numerical Analysis[Kamada and Kawai, 1989] and Simulated Annealing[Davidson and Harel, 1996] have been applied with varying degrees of success. Our basic algorithm most closely follows that of [Fruchterman and Reingold, 1991]. [Huang et al., 1998] explored the animation of such an algorithm in 2D and the same approach is easily extended to 3D. Another interesting animated implementation is that of [Brandes et al., 2000] for visualising WWW structures in 3D. The method involves simply converting the net forces on each node

to incremental displacements in order to produce a smooth animation with a 'natural' appearance. Such animation of the layout process helps to preserve the user's 'mental map', allowing them to track the movements of elements of the model as changes are made interactively[Misue et al., 1995]. In Ware and Franck's study the greatest benefits were seen when a degree of interaction with the model was available. Thus we would expect that the greatest benefits of 3D visualisation would be experienced in interactive UML systems rather than static displays. The prototype described was conceived with this in mind and was therefore designed as a tool for constructing and editing UML models as well as for simple visualisation.

4.1 The Forces

The first pair of forces we use are common to many Force Directed approaches[Battista et al., 1999]: a *Spring Force* between nodes connected by an edge which seeks to restore the edge to its natural length according to Hooke's law, and a *Repulsive Force* similar to an electrostatic force between each pair of nodes in the graph and proportional to the product of their mass. For the repulsive force we consider the nodes to have masses or charges and we set the mass of a cluster to be the sum of the masses of its constituent nodes.

We describe the force which makes our clustered layout method possible in more detail:

4.2 Origin Force

When laying out a generalised graph, the presence of a repulsive force will cause disconnected subgraphs to drive each other to infinity.

The use of a relatively weak but long range force towards an arbitrary origin is one elegant solution to this problem. This is similar to a force which [Frick et al., 1994] described as a "gravitational" force affecting the whole graph. We extend this force to apply to each cluster in a nested graph. This approach is also similar to a method suggested by [Huang and Eades, 1998], however they maintain cluster cohesion through the use of a 'virtual' node at each cluster centre and 'virtual springs' connecting this to each member node of the cluster. Another method suggested by [Wang and Miyamoto, 1995] used a divide and conquer approach to layout clusters but this is not easily animated.

In a nested context the origin force is essential so that forces acting on an entire cluster can be balanced against the forces acting on its member nodes, particularly members connected to nodes outside the cluster. An example showing the forces reaching a balanced state is shown in Figure 1.

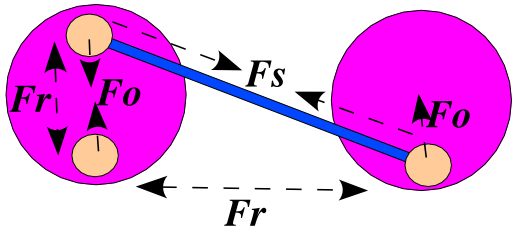
The force is also important in hierarchies of clusters because each cluster may contain its own locally disconnected subgraphs and also because the application of a relatively strong force between each of a cluster's members and the cluster's origin (or centroid) results in cluster cohesion making clusters appear as tighter groups within a larger graph.

The form of the origin force we applied is:

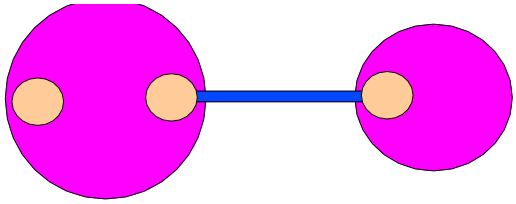
$$\vec{F_{origin}}(n) = \begin{cases} C_o \|\vec{V(n,o)}\| \vec{V(n,o)} & \text{if } \|\vec{V(n,o)}\| < 1 \\ 2C_o \vec{V(n,o)} & \text{otherwise} \end{cases} \quad (1)$$

where:

- C_o is the Origin Force constant, a scale factor affecting the strength of the force.



(a) Initial Layout



(b) Balanced Layout

Figure 1: Two figures showing all the forces reaching a balanced state, where *Fr* is Repulsion, *Fs* is the Spring force and *Fo* is the Origin force.

- $\vec{V(n,o)}$ is a vector from the position of node n to the origin of the cluster.
- $\|\vec{V(n,o)}\|$ is the Euclidean distance of n from the origin. This extra scale factor is applied only when it is less than 1 so that the force forms a smooth "Gravity Well" as n approaches the origin rather than a sharp "V" which occasionally causes unstable oscillations due to the iterative nature of the FDA.
- multiplying the second term by a factor of 2 results in a smooth force versus displacement gradient.

A great benefit of the FDA is that more forces can be added as required to fulfill other aesthetic criteria. The following force to align directed edges can be seen at work in the 3D UML Figures 4 and 5 and was inspired by [Sugiyama and Misue, 1995] but the calculation differs in 3D.

4.3 Directed Field Force

This is a force field across an entire cluster that causes directed edges to align themselves to minimise their angle of intersection with the field. This is analogous to the behaviour electric and magnetic dipoles in their respective physical fields.

Figure 2 shows the situation where:

- $\vec{B}$ is a vector representing the magnetic field direction and magnitude.
- $\vec{v}$ is a vector representing the edge joining n_1 and n_2
- $\vec{F}$ and $-\vec{F}$ are the resultant forces on n_1 and n_2.

In 2D it is easy to calculate a vector for this force by calculating the magnitude proportional to θ:

$$\|\vec{F_{2D}}\| = \|\vec{B}\| \arccos \frac{\vec{v} \cdot \vec{B}}{\|\vec{v}\|\|\vec{B}\|} \quad (2)$$

and the direction of the resultant force is simply the normal to the edge.

However in 3D there is no single normal to an edge. We obtain a vector normal to the plane in which B and v lie and then find a vector perpendicular to this normal and the edge itself (hence in the plane in which B and v lie), see Figure 2.

$$\vec{F_{3D}} = \frac{\vec{v} \times (\vec{v} \times \vec{B})}{\|\vec{v}\|^2} \tag{3}$$

The divisor $\|\vec{v}\|^2$ will make short edges (or compressed edges) equally susceptible to the force as long (or stretched) edges. This comes from the observation that the magnitude of the force (without the divisor) is (from the definition of magnitude of a cross product):

$$\|\vec{F_{3D}}\| = \|\vec{v}\|^2 \|\vec{B}\| \|\sin\theta\| \tag{4}$$

Note that in Equation 3 we leave F_{3D} proportional to $\sin\theta$ rather than θ as a matter of taste since it both exaggerates the effects of the force and saves on the performance cost of calculating $\arcsin\theta$[3].

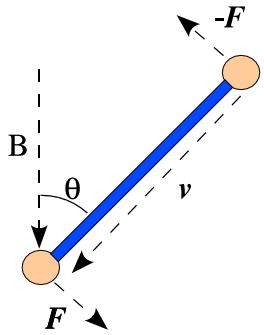


Figure 2: The forces exerted on two nodes connected by an edge due to the directed field force.

The tool constructed to test some of these ideas is called "Wilma". Constructed in Java using the Java3D toolkit it was carefully designed to be as extensible and maintainable as possible since the graph engine is suitable not only for visualising UML but for any type of conceptual modelling involving graphs. More information about this open source software tool is available from the Wilma website `http://www.wilmascope.org`. A sample class diagram showing the basic structure of the Wilma *graph* and *forcemodel* packages is depicted in figure 3.

4.4 3D Class Diagram Elements

Representing these elements in 3D allows for some creative use of the third dimension. In the Wilma tool it was also possible to add colour representations to the implementation so this was done adding yet another degree of freedom. We will now look at how this extra flexibility is utilised in our 3D UML interpretation. An example 3D interpretation of the class diagram given in Figure 3 is given in Figure 4. Note that some of the impact (and readability) of the visualisation is lost when printed out since the representation is really intended for interactive use in an animated, navigable, colour graphics environment.

Packages are represented by the clusters described above. This representation of packages is visible in Figure 4.

[3]Though the performance argument is not a very good one since using a very course grained look-up table for arcsin would be taking much less of a liberty with our "cartoon" physics.

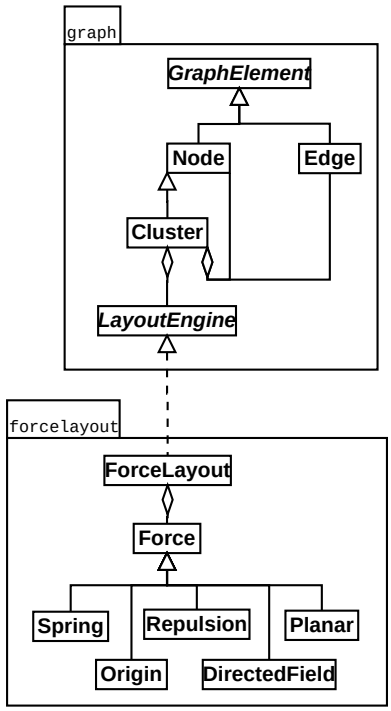


Figure 3: A Class diagram showing the essential classes implementing Wilma's graph and force model engines.

Classes are represented by a 3D cube with a label painted on the face using texture mapping. Legibility of the label text turned out to be a problem as the graphs need to be rotated and zoomed freely to find the optimal viewing angle. Therefore to improve readability one face of the cube is kept facing the viewer to maximise the visible area[4]. The class cubes are also scaled horizontally to best fit the text.

Plain Edges are are rendered as blue pipes. They are easily visible but blue is unobtrusive enough not to distract from the Class nodes.

Edge Labels are labelled by text inverse in colour to the background floating in space just above the edge. The labels are oriented to always face the viewer. Cardinality is shown by similar labels at the ends of the edge.

Directed Plain Edges are represented by the addition of a small blue cone next to the label, pointed so as to indicate direction.

Inheritance Edges are similar to plain edges but with the addition of a bright green cone replacing the triangle used in the traditional form. This has the advantage that the 2D projection of the edge will still look like the 2D form from most angles.

Aggregation Edges are like plain edges with the addition of small red cubes pierced through opposite corners by the edge cylinder. Thus providing a 2D projection once again similar to that of the traditional form.

[4]For the same reason it was decided not to support the display of attribute and method details on the cube surfaces for the prototype. Eventually this might be supported via a 'click to zoom' interface.

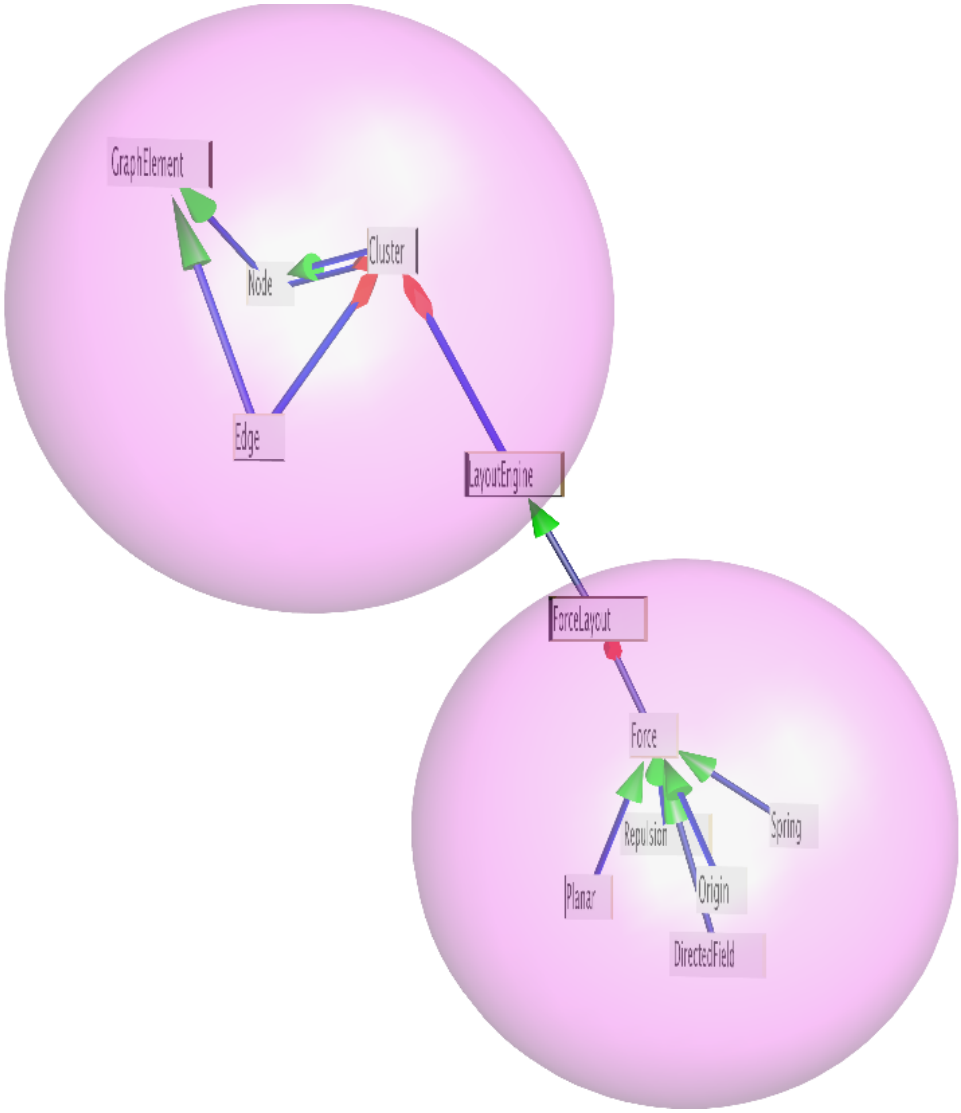


Figure 4: A 3D interpretation of the UML model in Figure 3 showing packages as clusters.

5 Evaluation of The System

An exploratory usability study was conducted on the system to determine the effectiveness of the Wilma tool and UML class models in 3D. The following is a very brief synopsis of the methodology and results of the study. More details are available in the Technical Report[Dwyer, 2000].

6 Methodology

We break the question of effectiveness into testing three hypotheses:

1. That the Wilma tool is useful in communicating UML Class models to users.
2. That the tool assists system architects when creating UML models.
3. That the tool helps to suggest meta structure in models to the users.

However, attempting to test the success of our paradigm for 3D UML according to these hypotheses with an empirical study would be inconclusive due to the number of unconstrained variables involved.

Therefore a heuristic evaluation[Nielson and Molich, 1990] methodology, a common form of software usability study, was chosen. The aim is to assess our hypotheses qualitatively, hopefully leading to further insights into the utility (or futility) of our paradigm. To achieve this a panel of experienced system architects was interviewed while using the system in a series of planned activities in a usability lab. Several heuristics were proposed to the subjects, on which to base their evaluation. These heuristics were based on those suggested by Nielson and Molich but are slightly modified to better suit the domain of an interactive graph drawing tool:

feedback does the tool provide satisfactory feedback?

prevent errors does the interface help to prevent errors?

mental model does the tool's presentation of the model correspond well with your own mental model?

navigation does the tool allow you to freely explore the model without getting lost?

In a style similar to that used by Storey[Storey, 1998, page 104], the questions posed to the subjects required them to perform either *Concrete tasks* which could be completed correctly or incorrectly in a measurable amount of

time, or *Abstract* tasks which required the subject to verbally appraise the tool.

There were several stages in each interview:

Preliminary Questions Subjects were asked about their experiences with system design, modelling, UML and software tools they had used for these tasks.

General 3D UML Interpretation Subjects were shown a very simple 3D UML model and tested on their interpretation of the individual diagram elements.

Basic Structure This set of questions aimed to test whether the participants could easily identify the most basic structural properties in a more complicated 3D UML model based on a real software system. An example of the type of system shown to the subjects is given in Figure 5.

Using a Directed Graph Again using a complicated model each user was asked questions testing the effectiveness of the Directed Field force in conveying hierarchy in a graph with directed edges.

Using a Clustered Graph A graph containing clusters was shown to the user and they were asked questions about the meta-structure of the model.

Simple Modelling Exercise The user was given a description of a simple system and asked to produce a model of the system in 3D.

6.1 Results

In general the subjects did very well in answering the concrete questions about the models shown to them in the sections on general 3D UML interpretation, basic structure and using a directed graph. There were some interesting comments relating to the various heuristics especially those relating to mental model:

> 'I think it was obvious when things were related to each other because there were clumps'
>
> 'You can easily see the symetry across several different axis'
>
> 'Being able to see the diagram from different angles allows you to emphasise different parts of the structure and really lets you see the tree in your head'

Both the anecdotal evidence and the observed performance of the subjects in the concrete tasks supports the viability of the 3D visualisation of UML models as described here and supports hypothesis 1. That a tool is 'useful' is obviously not the strongest claim that could be made, but there are still many people who question the utility of 3D user interfaces in almost any setting[Nielsen, 1998]. Therefore we see testing 'usefulness' as a necessary first step in evaluating the paradigm.

Our hypotheses get progressively more ambitious. Hypothesis 2 was tested in a simple modelling exercise. Subjects were given a textual description of a hypothetical banking network and were asked to model the system using the tool. A model produced by one of the subjects is shown in Figure 6. Once again the subjects mostly had little problem completing this exercise and did so in only a few minutes. However, as mentioned above there were subjects who said they would have preferred to have been able to define all the classes separately before determining the relationships between them. This is reminiscent of the classic CRC method for determining the roles of classes in a system [Wirfs-Brock et al., 1990]. Others had no problems with the top-down scheme and actually felt that it helped them, possibly because it forced them to work in a well organised top-down fashion. Perhaps both capabilities need to be supported. Certainly some systems may lend themselves better to top down design than others.

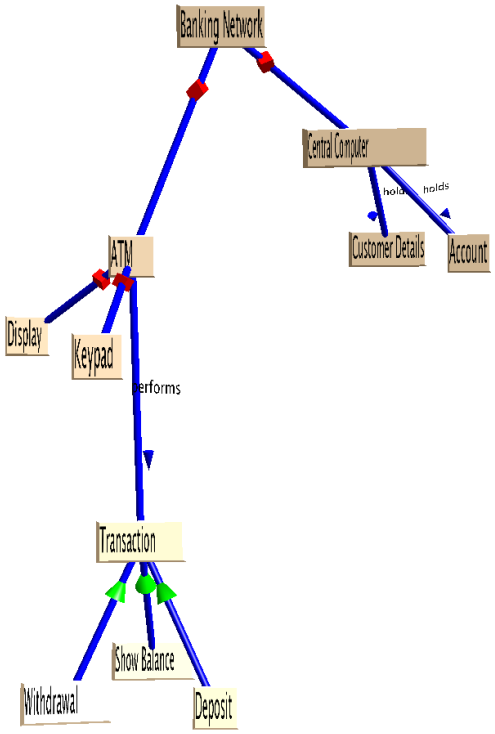


Figure 6: A banking network model produced by one of the subjects

The fact that the subject who had the most difficulty tried to do the exercise on paper before using the tool suggests that there may be a learning curve for some people in the paradigm shift from 2D to 3D. The idea that some degree of 'retraining' may be required for some users or that 3D representation may not be as suitable for some users is definitely worthy of attention in further studies. Notably, the three subjects who had the least difficulty navigating the 3D model all commented that they felt their experience with 3D "First Person Shooter" games was beneficial to this task.

Another test of the above hypothesis came when subjects were asked to find a logical error that was deliberately introduced. All but one test subject found the error. An interesting phenomenon that became evident at this stage was that amongst the subjects who found the error there was some variation in the time taken, from only a few seconds up to a minute. When asked to explain how they found the error it became evident that the slower group were examining each of the nodes in turn and trying to 'reason out' the problem. The quicker group generally made comments such as:

> 'It just stuck out awkwardly'.

Which suggests that they were using gestalt information about the model. That is, they were viewing the overall configuration of the model rather than trying to reason about each of its elements. This is beginning to give evidence for hypothesis 3.

This concept was also tested when we asked the users to identify patterns in the clustered model.

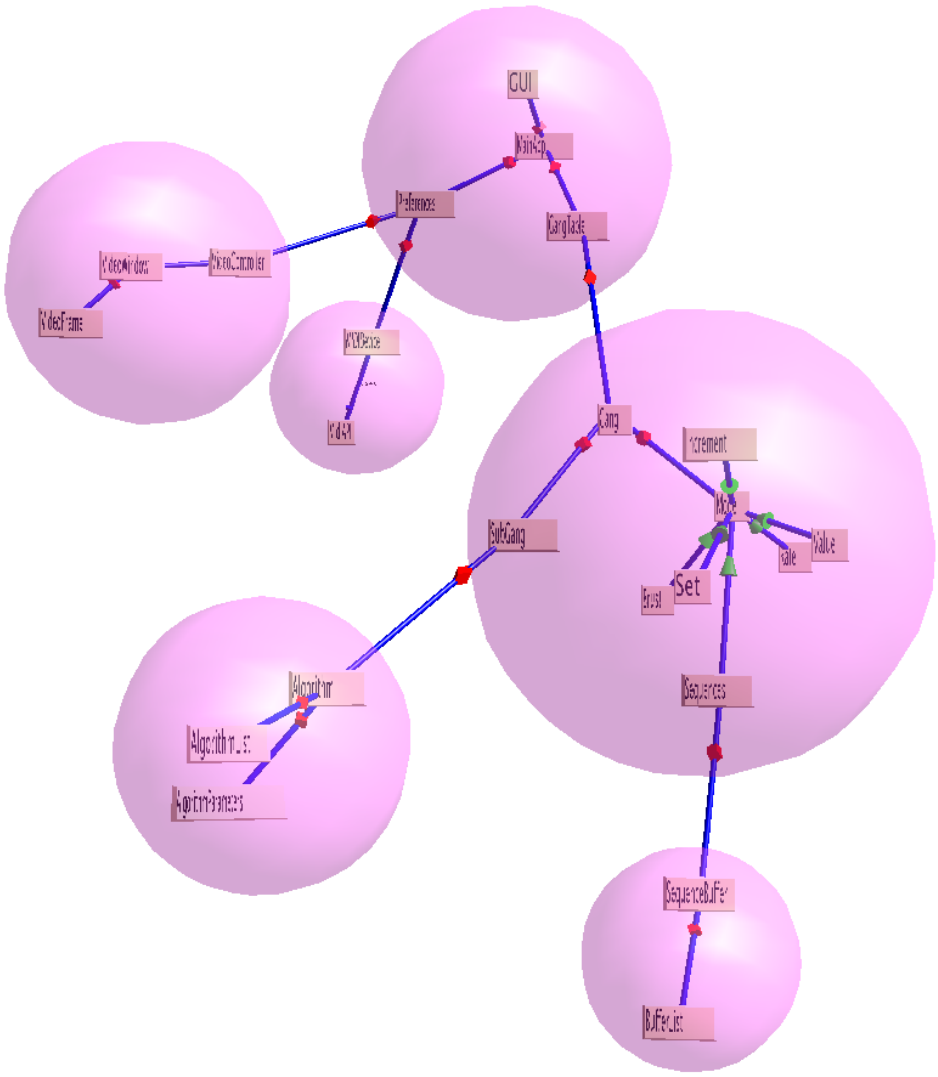


Figure 5: One of the models shown to the subjects.

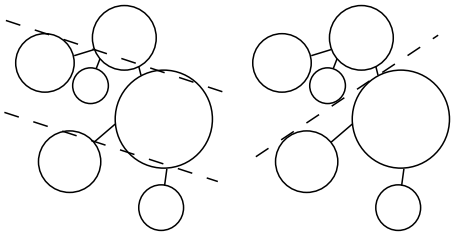
Most users claimed that they could see layers appearing however when asked to show the divisions between layers that they could see it became evident that there were 3 different interpretations. These are shown in Figure 7.

Note that on further investigation all three interpretations would be valid partitionings of the actual system described by the model. There are several possible explanations for the differences:

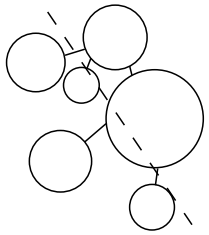
- The model shown was found to be similar to systems they had seen before. Therefore they may have had preconceptions about how it should be divided into layers. However, if this is the case, the fact that they were still easily able to identify such layers shows that the tool is assisting them in this task.
- When viewed from different orientations, different divisions may seem more obvious. If all the different interpretations are valid then a skilled user might deliberately rotate the model using our tool to find them.
- The choice of where to partition may depend on the goals of the partitioning. For example if one was trying to arrange the structure into layers dictated by depth in the tree hierarchy then Figure 7(a) may be most appropriate. If however one is dividing up a system so that it may be deployed as a distributed system over a network one would avoid dividing the system in such a way that many points of coupling would have to be mapped into a network transparent representation. If such was the goal of partitioning then Figure 7(b) may be most appropriate. The last possibility is to simply separate leaf nodes in the tree, or components which have no other components depending on them. This last view is given by Figure 7(c).

In any case all of these observations are symptomatic of different effects of coupling and cohesion between modules in the system, as were the many comments received from users about the visibility of 'dependencies'. A module that has many dependencies on another module can also be said to be 'strongly coupled' to that module. A module whose constituent classes are highly dependendant on one another can be said to have 'high cohesion'. In general software engineers tend to agree that high cohesion within modules and low coupling between modules are desirable properties of an architecture[Pressman, 1997, pages 357–361]. So being able to see where these relationships occur in a design is clearly a desirable property of a software visualisation system.

So in our study test subjects were able to identify meta structure in the models as presented to them by the tool which supports hypothesis 3.



(a) Layering (b) Decoupling



(c) Leaves from roots

Figure 7: Possibilities for partitioning the model from Figure 5.

7 Concluding Remarks and Further Work

We have shown an effective method of layout for 3D UML diagrams and argued that this paradigm improves a user's cognition of the architectural structure of complex system models. The evaluation described here was designed to be a pilot study and as such raised some interesting issues and suggested the potential of our system but it is hoped that future studies will provide more conclusive evidence for this claim. Work is also ongoing in improving the performance and functionality of the Force Directed Algorithm implementation used and testing its application to a number of different domains.

7.1 Acknowledgements

This paper is based on work done as part of an Honours thesis[Dwyer, 2000] at the University of Melbourne. The Author gratefully acknowledges the supervision received from Edmund Kazmierczak and Elizabeth Sonenberg throughout this work and would also like to thank the Department of Information Systems for access to their IDEA lab (`http://www.dis.unimelb.edu.au/staff/idgroup/IDEA-Lab`) for the usability study. The Author also collaborated with Peter Eckersley who investigated web page classification[Eckersley, 2000] and assisted with the implementation of clusters.

References

[Battista et al., 1999] Battista, G. D., Eades, P., Tamassia, R., and Tollis, I. (1999). *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice-Hall, Englewood Cliffs, NJ.

[Brandes et al., 2000] Brandes, U., Kääb, V., Löh, A., Wagner, D., and Willhalm, T. (2000). Dynamic www structures in 3d. *Journal of Graph Algorithms and Applications*, 4(3):183–191.

[Cohen et al., 1994] Cohen, R., Eades, P., Lin, T., and Ruskey, F. (1994). Three dimensional graph drawing. In *Proceedings of GD'94, LNCS*, volume 894, pages 1–11. Springer-Verlag, Berlin.

[Davidson and Harel, 1996] Davidson, R. and Harel, D. (1996). Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics*, 15(4):301–331.

[Dwyer, 2000] Dwyer, T. (2000). Three dimensional uml using force directed layout. Technical Report TR2001/25, Department of Computer Science, The University of Melbourne, Parkville, Australia.

[Eades, 1984] Eades, P. (1984). A heuristic for graph drawing. *Congress Numerantium*, 42:149–160.

[Eckersley, 2000] Eckersley, P. (2000). Classiscope: Using meta-structure analysis to build a better magnet for the web haystack. University of Melbourne. Honours Thesis.

[Frick et al., 1994] Frick, A., Ludwig, A., and Mehldau, H. (1994). A fast adaptive layout algorithm for undirected graphs. In *Proceedings of GD'94*, volume 894, pages 388–403. Springer-Verlag, Berlin.

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

[Gogolla et al., 1999] Gogolla, M., Radfelder, O., and Richters, M. (1999). Towards three-dimensional representation and animation of uml diagrams. *Proceedings of UML'99, Lecture Notes in Computer Science*, 1723:489–502.

[Huang and Eades, 1998] Huang, M. and Eades, P. (1998). A fully animated interactive system for clustering and navigating huge graphs. In *Proceedings of Graph Drawing 1998*, volume 1547, pages 374–383. Springer-Verlag, Berlin.

[Huang et al., 1998] Huang, M., Eades, P., and Wang., J. (1998). On-line animated visualization of huge graphs using a modified spring algorithm. *Journal of Visual Languages and Computing*, 9(v1980094):623–645.

[Kamada and Kawai, 1989] Kamada, T. and Kawai, S. (1989). An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7 – 15.

[Kumar and Fowler, 1994] Kumar, A. and Fowler, R. (1994). A spring modeling algorithm to position nodes of an undirected graph in three dimensions. Technical report, Department of Computer Science, University of Texas.

[Mellor, 1994] Mellor, P. (1994). Cad: Computer-aided disaster. *High Integrity Systems*, 1(2):101–156.

[Misue et al., 1995] Misue, K., Eades, P., Lai, W., and Sugiyama, K. (1995). Layout adjustment and the mental map. *Journal of Visual Languages and Computing*, 6(2):183–210.

[Nielsen, 1998] Nielsen, J. (1998). 2d is better than 3d. http://www.zdnet.com/devhead/alertbox/981115.html.

[Nielson and Molich, 1990] Nielson, J. and Molich, R. (1990). Heuristic evaluation of user interfaces. In *ACM Conference on Computer Human Interaction*, SIGCHI, pages 249–256.

[Pressman, 1997] Pressman, R. (1997). *Software Engineering a Practitioner's Approach*. McGraw Hill, 4th edition.

[Purchase, 1998] Purchase, H. (1998). Performance of layout algorithms: Comprehension not computation. *Journal of Visual Languages and Computing*, 9(v1980093):647–657.

[Reiss, 1994] Reiss, S. (1994). 3-d visualization of program information. In *Proceedings of GD'94*, volume 894, pages 12–24. Springer-Verlag, Berlin.

[Rumbaugh et al., 1999] Rumbaugh, J., Jacobson, I., and Booch, G. (1999). *The Unified Modeling Language User Guide*. Addison-Wesley.

[Storey, 1998] Storey, M.-A. (1998). *A Cognitive Framework For Describing and Evaluating Software Exploration Tools*. PhD thesis, Computing Science, Simon Fraser University, Canada.

[Sugiyama and Misue, 1995] Sugiyama, K. and Misue, K. (1995). Graph drawing by the magnetic spring model. *Journal of Visual Languages and Computing*, 6(3):217–231.

[Wang and Miyamoto, 1995] Wang, X. and Miyamoto, I. (1995). Generating customized layouts. In *Proceedings of GD'95*, volume 1027, pages 504–515. Springer-Verlag, New York Inc.

[Ware and Franck, 1994] Ware, C. and Franck, G. (1994). Viewing a graph in a virtual reality display is three times as good as a 2-d diagram. In *IEEE Conference on Visual Languages*, pages 182–183.

[Ware et al., 1997] Ware, C., Franck, G., Parkhi, M., and Dudley, T. (1997). Layout for visualizing large software structures in 3d. In *Proceedings of VISUAL'97*, pages 215–223.

[Wirfs-Brock et al., 1990] Wirfs-Brock, R., Wilkerson, B., and Wiener, L. (1990). *Designing Object-Oriented Software*. Prentice-Hall, Englewood Cliffs, NJ 07632.

[Wiss and Carr, 1999] Wiss, U. and Carr, D. (1999). An empirical study of task support in 3d information visualizations. In *Proceedings of IEEE InfoVis'99*, pages 392–399.