Computer Network Visualisation

Richard Zschech, Paul Coddington and Ken Hawick
{rz, paulc, khawick}@cs.adelaide.edu.au

Technical Report DHPC-099
27 October 2000

Department of Computer Science
Adelaide University,
South Australia, 5005

Abstract

Computer networks and especially the Internet are by their very nature tremendously complicated. This is because networks include many hosts and connections between them. Computer visualisation techniques offer the opportunity to display complicated sets of information in an easy to view and easy to understand manner. This thesis examines methods for mapping computer networks and visualising the results.

This project involved implementation of a generic three-dimensional graphing package. The package contains many different techniques for laying out the graphs in an easy to visualise pattern. Good layout methods are needed to facilitate people's understanding of the visualisation application. The resulting graphs were rendered in three-dimensions using Java3D. The user is able to view and interact with the graphs in real time.

One application of visualisation methods is the mapping of computer networks and the collection of statistics about them. This project used the Simple Network Management Protocol to query the required information from the network and used it to build the graphs.

Acknowledgements

I would like to thank the following people for their help and inspiration during this project. My supervisors Dr. Ken Hawick and Dr. Paul Coddington for all their support. Dr. Heath James for his help with miscellaneous problems in the project. Theodore Wyeld for his inspiration with the IP address based graph layout algorithm. My fellow honours students who helped me through the year. And finally my family and friends who were always there for me.

Table of Contents

Table of Figures

1 Project Introduction

The goal of computer visualisation is to help people view and better understand their data. This data can come from measurements, experiments or numerical simulations. Frequently the size and complexity of the data makes it difficult to understand by direct inspection. The data may be generated several times or change with time during an experiment or simulation and understanding how the data varies with time may be difficult.

Computer visualisation helps with these problems by representing the data in an easy to understand graphical manner. Using virtual reality the data can be viewed and manipulated naturally in a true three-dimensional environment. Viewing the data in this way can quickly draw the user's attention to interesting or abnormal portions of information.

This project developed a package for visualising arbitrary graphs in three dimensions. This included the development of an underlying graph data structure for storing the graph information and the three-dimensional rendering module for visualising the graph. I also developed some graph layout algorithms for laying out the graph in a neat and well-organised manner that was easy to visualise. The package could easily be extended to allow many filtering and overlaying methods in the package, for example showing only nodes two hops from a node or highlight the shortest path between two nodes.

This graphing package could have many applications from simple highway connections between cities to exploring state spaces in the Communicating Sequential Processes [Hoa84] language. A specific application using the package was developed, which explores and maps sections of local area networks. This application used network hosts for the nodes of the graph and the connections between hosts as the edges of the graph.

This thesis is organised as follow:

Chapter 2 covers some background work on visualisation in two and three-dimensions. It then describes previous work in visualisation of graphs and graph layout algorithms. Last it covers previous work in mapping computer networks.

Chapter 3 covers the graph visualisation package and how it was implemented. The describes how the package can be extended for the specific network visualising application

Chapter 4 gives a detailed description on how the Simple Network Management Protocol works which was used to map and gather statistics about the computer networks.

Chapter 5 describes the graph layout algorithms that were used in laying out the computer networks.

Chapter 6 gives an overview of the final network mapping and visualisation software.

Chapter 7 describes the file format used to save the network graphs to files.

Chapter 8 outlines future work that this project could lead on to.

Chapter 9 gives conclusions about the success of the project.

2 Background

2.1 Computer Visualisation

Computer visualisation has become a large field and "sub-fields" are beginning to emerge such as visualisation of graphs. Graph visualisation can be used in many different application areas. Most people have encountered a file hierarchy on a computer system. A file hierarchy can be represented as a tree, which is just a special type of graph. It is often necessary to navigate through the file hierarchy in order to find a particular file. Anyone who has done this has probably experienced a few of the problems involved in graph visualisation such as “Where am I?” and “Where is the file that I'm looking for?” These problems can be potentially overcome using computer visualisation because the user can get a higher-level view of the information.

Other familiar types of graphs include the hierarchy illustrated in an organisational chart, web site maps, as well as browsing history. In biology and chemistry, graphs are used as evolutionary trees, molecular maps, genetic maps, biochemical pathways, and protein functions. Other areas of application include object-oriented systems like class browsers, state-transition diagrams, Petri nets, data flow diagrams, subroutine-call graphs, entity relationship diagrams, semantic networks and knowledge-representation diagrams, project management PERT diagrams, logic programming, VLSI circuit schematics, and virtual reality scene graphs. Some of this information is not always in a purely hierarchical format which necessitates the use of general graphs than trees [HMM98].

The size of the graph to view is a critical issue in graph visualisation. Large graphs pose several difficult problems. If the number of elements is large good performance can be compromised and the size of the graph can reach the limits of the viewing platform. Even if it is possible to layout and display all the elements, the issue of viewability and usability arises. This is because it can become impossible to distinguish between nodes and edges. Usability is usually an issue before the problem of being able to distinguish is reached. Comprehension and analysis of data in graph structures is easiest when the size of the displayed graph is small. Displaying an entire large graph may give an indication of the overall structure but makes it difficult to comprehend.

The layout of the graph is of critical importance. A good layout can help people to understand the application, but a bad diagram can be misleading. One technique is to layout graphs in three-dimensions instead of two-dimension. The hope is that the extra

dimension would give more space, which would ease the problem of displaying large structures and simplify the layout.

The following figure shows an example of previous three-dimensional visualisation work. It is displaying J. Rekimoto's virtual desktop as an information cube. It uses a generalisation of the two-dimensional approach using nested boxes to layout the information [Rek93].

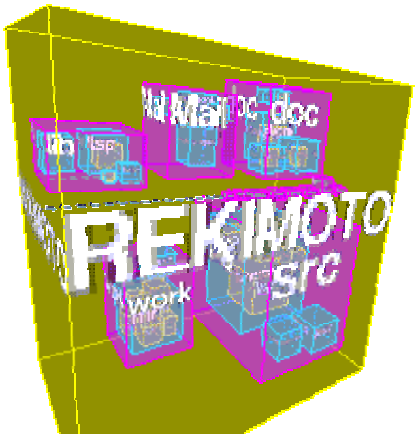


Figure 2.1 Information Cube

In addition to gaining more space another possible advantage of using three-dimensions is because the user is more familiar with the real three-dimensional environment. Three-dimensional graphics lends itself to the creation of real world metaphors. These metaphors should help in perceiving complex structures. The following figure 2.2 shows a file system navigator. This program displays a tree like system of roads between cities which represent the directories in the file system. The size of the cities and building inside the cities are dependent on the size of the files in the directories [FSN].

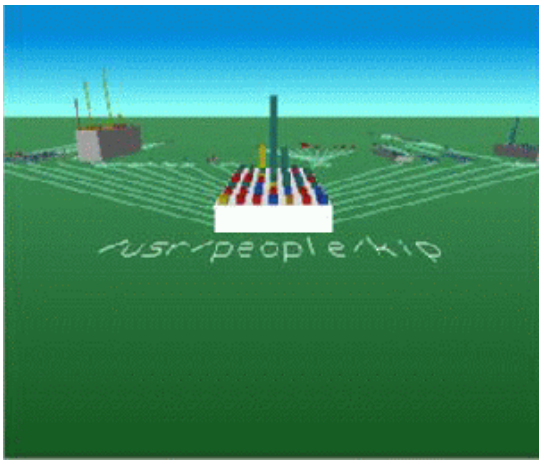


Figure 2.2 File System Navigator

There are many techniques for laying out more complicated graphs which may be more applicable to the mapping of computer networks. The following example figure 2.3 show graphs laid out using the radial technique in two and three dimensions [EW94]. These techniques clearly reflect the intrinsic hierarchy of the information graph.

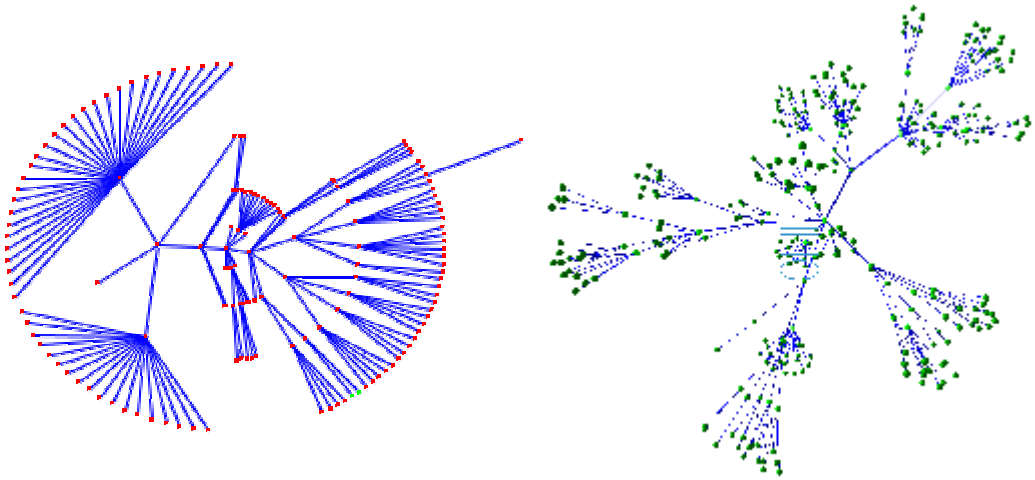


Figure 2.3 Radial Graph Layout in 2D and 3D

The following example shown in figure 2.4 shows a graph using the "H-Tree" layout technique [EW94]. This layout shows the graph in a less structured fashion because it is less clear where the centre of the graph is. This may lead the user to navigate the graph in a less hierarchical fashion and they may get a different and possibly better understanding.

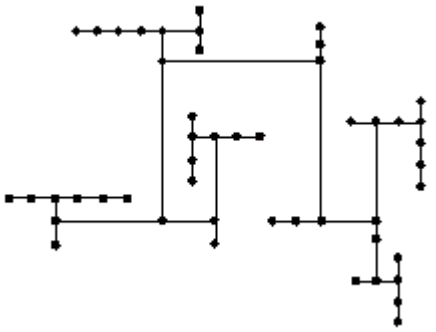


Figure 2.4 H-Tree Graph Layout

The next example shown in figure 2.5 shows a graph layout using the mechanical spring model. This model simulates the graph as set of rings, as the nodes of the graph, and springs connecting the rings, as edges of the graph. To layout the graph

the position of the nodes is calculated such that there is the minimum tension on the springs.

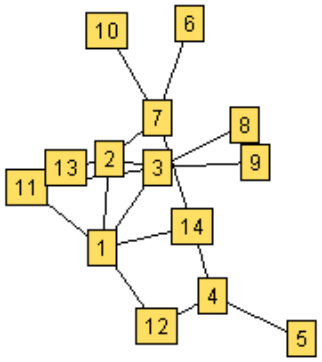


Figure 2.5 Example Spring Layout

2.3 Mapping Computer Networks

A small amount of work has been done in mapping computer networks. The first example of this is by was done by a company called “Matrix” in January 1998 which used an Internet domain survey from another company “Network Wizards”. This survey attempts to discover every host on the Internet by doing a complete search of the Domain Name System. The physical location of the domains were then obtained through other non-automated means and the first company then placed all the hosts in the survey and placed them on a map of the world. The final result is shown in figure 2.5 below [MNP].

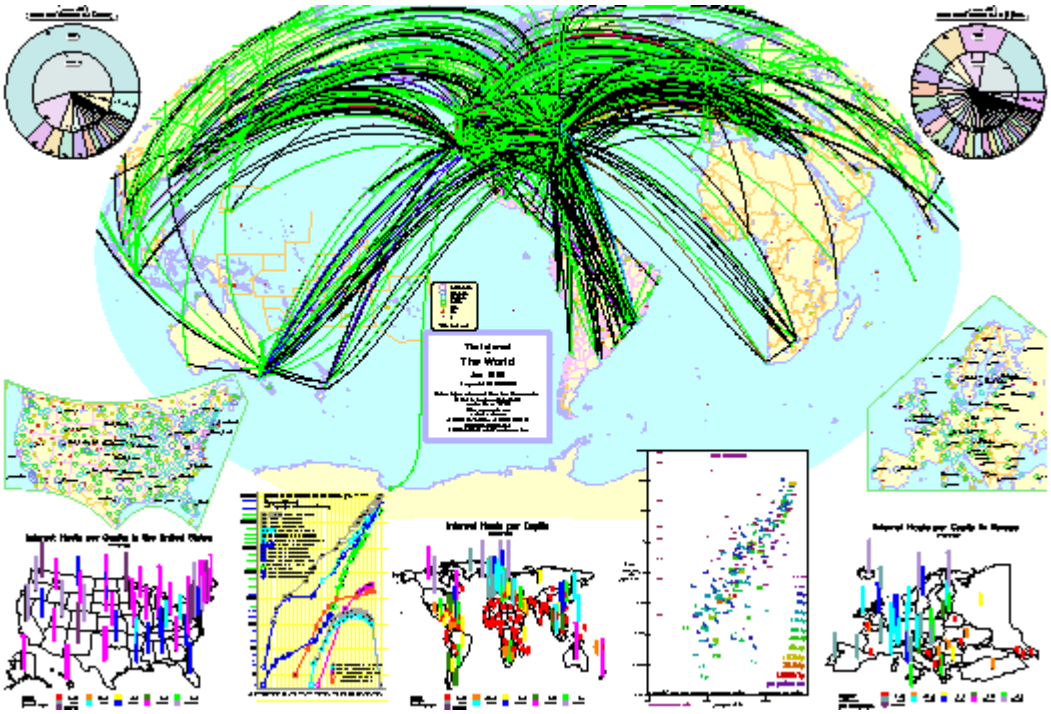


Figure 2.5 Internet Map using domain name survey

Another example of network mapping this is by was done by Hal Burch and Bill Cheswick in January 2000. This mapping calculated the shortest path from a computer in Murray Hill, New Jersey to every computer network registered in the authoritative global database kept by Merit Network Inc. The final result is shown in figure 2.6 below [CB00]. The different colours in the network are based on the different extensions of the computers domain names. For example “com”, “edu”, “net” are different colours.

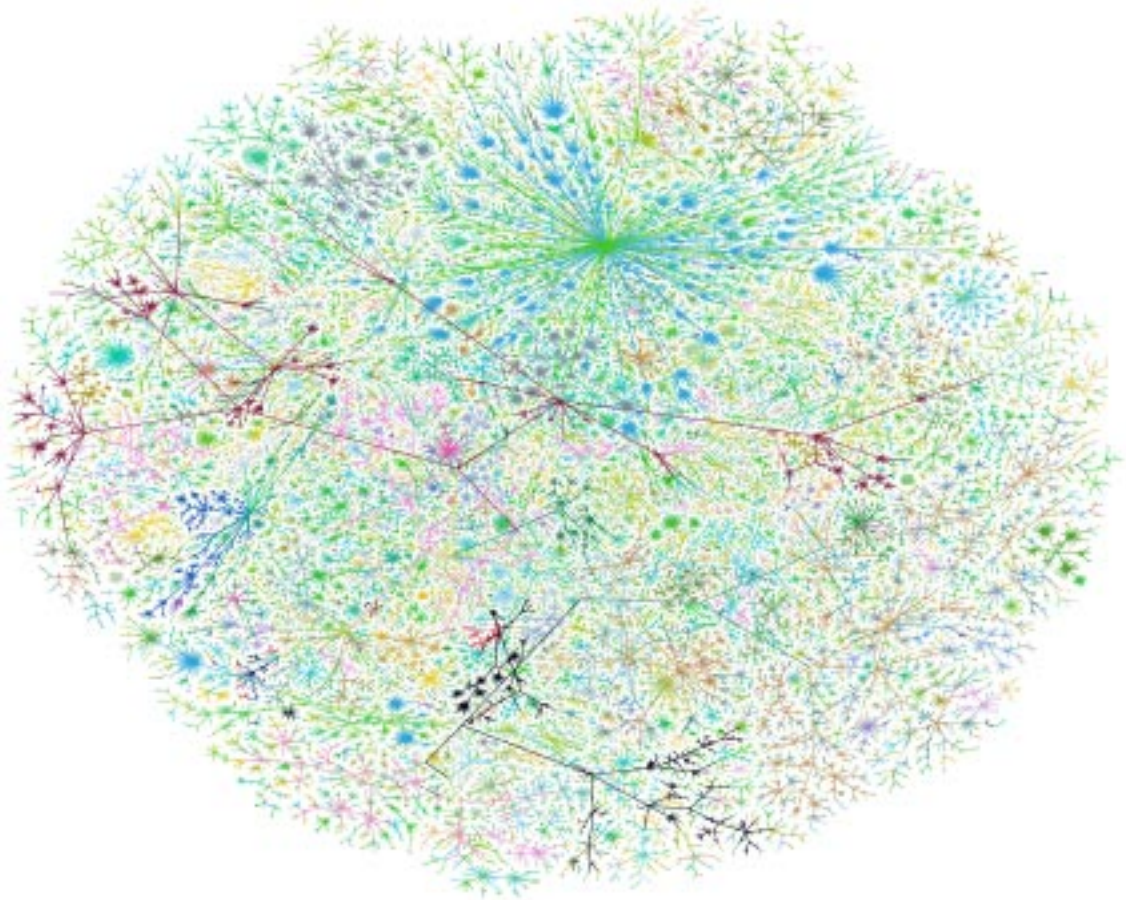


Figure 2.6 Internet Map using registered networks

3 Graph Visualisation and Java 3D

3.1 Graph Overview

Graphs are one of the most general data structures. They consist of a set of nodes and a set of edges, which connect the nodes. The nodes and edges can have information attached about the node or edge. The edges can be directional from one node to another and not back; they can be bi-directional joining two nodes in both directions and they can be undirected. In some specifications, a node of a graph can be a subgraph so that within the node there is another entire graph.

Graphs are very general because there can be an arbitrary number of nodes and edges between the nodes. They could be used to represent linked list or even tree data structures.

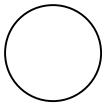
3.2 Java 3D Overview

Java 3D is an interface for writing programs which displays and interacts with three-dimensional objects. It provides a collection of high-level constructs for creating and manipulating 3D geometry and structures for rendering that geometry.

Java 3D defines a virtual universe which includes all the information about the geometry, lights and cameras that are used to do the three-dimensional rendering. Java 3D uses the common *scene graph* technique to store all this information in a hierarchy. The term *scene graph* is a bit misleading because almost all of the elements in the graph can only have one parent element which is more like a tree structure containing branches and leaf nodes [J3DAPI].

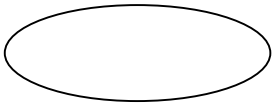
The following figures use the symbol conventions defined in the "Getting Started with Java 3D" reference manual [GSJ3D] which are as follows:

Virtual Universe: This is root node of the scene graph.

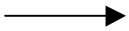


Group (Branch) Node: This scene graph node allows other nodes to branch from it. It may also contain other information used in the rendering process. For example a Transform Group node includes a three-dimensional transformation used for all its sub-branches.

Leaf Node: These screen graph nodes do not allow other nodes to branch from it. It contains information used in the rendering process. For example a Shape node or Light node.



Node Component: This scene graph node specifies attributes about a Leaf node. For example the geometry associated with a shape node.



Parent Child Link: This arrow indicates how the nodes are arranged with the children being the nodes branching off the parent node.

Attribute Reference: This arrow indicates that an attribute is attached to a node.

A standard simple scene graph contains two main branches. These branches are the *view branch* and the *geometry branch*. They are connected by the virtual universe node as shown in figure 3.1 below.

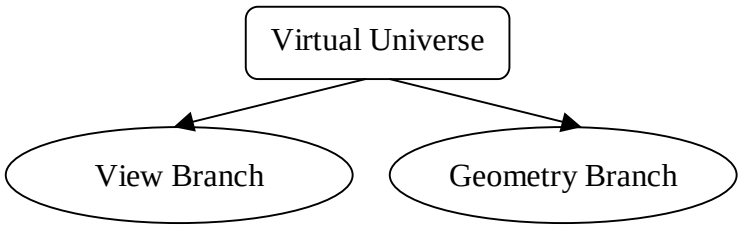


Figure 3.1 Scene Graph

The view branch contains information about the cameras in the virtual universe. The view branch contains a 3D transform, which represents the information about the position and orientation of the camera with respect to the virtual universe. This is shown as TG in figure 3.2 below. The view platform lead node, shown in figure 3.2 below, contains information about the camera used to view the scene. This includes information like the field of view, focal length, maximum and minimum range of the camera and the size of the image produced by the camera. Java 3D supports multiple view branches representing multiple cameras in the virtual universe. These can be added to the branch group node, shown as BG in figure 3.2 below, which is always the first node in the view branch.

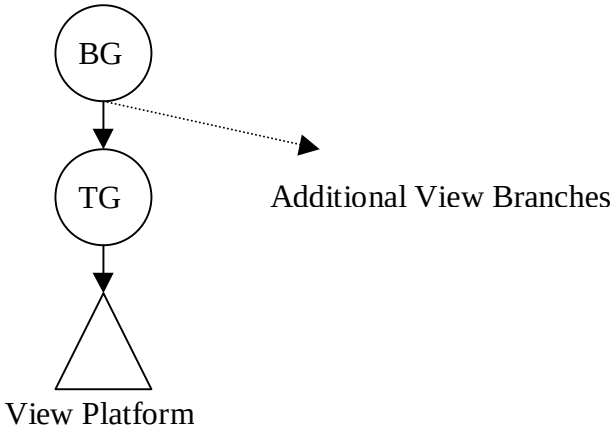


Figure 3.2 View Branch

The geometry branch contains information about all the three dimensional objects in the virtual universe. An example geometry branch is given in figure 3.3. This branch starts with a branch group node so that other nodes can be extended under it. The shape node is a leaf node which includes geometry information like vertices and triangles and also includes appearance information like the material of the shape which is used for lighting. The shapes can be added under a transform group which positions and orients the shape much like the transform for the view branch above.

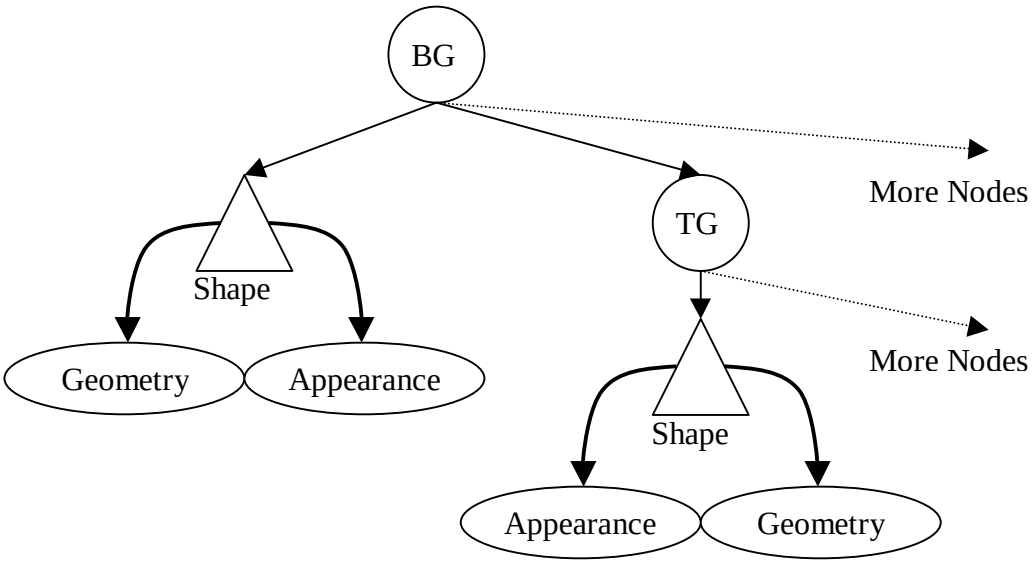


Figure 3.3 Geometry Branch

The branches for the objects can be arbitrarily complicated except that each node can only have one parent. The Java 3D renderer then follows the branches from the root

to the leaf nodes performing the transformations and renders the shapes along the way. Java 3D also includes nodes for lights, sounds and behaviours [J3DAPI].

3.3 Graph Package

3.3.1 Graph Package Classes

The graph package that was implemented for this project consists of three main Java classes as follows:

Node: this class represents the nodes in the graph. Each node is uniquely identified by an object, which cannot be changed after the node's creation. It also contains methods to get and set the object that stores the information attached to the node.

Edge: this class represents the edges in the graph. Each edge is also uniquely identified by an object, which cannot be changed after the edge's creation. The edge is also created with references to the source and destination nodes for the edge. These references cannot be changed after the creation of the edge. The edge also contains methods to get and set the object that stores the information attached to the node.

Graph: this class represents the graph. The graph class extends the node class so that entire graphs can be considered as nodes and can be added to a super graph as a node. This is shown in the class hierarchy diagram below. The graph class contains a list of all the nodes and edges contained in the graph, and methods to add and remove nodes and edges. It also contains methods to check if a node or edge is in the graph and to enumerate the edges and nodes in the graph.

The specification of the Node, Edge and Graph classes are contained in appendix 1.

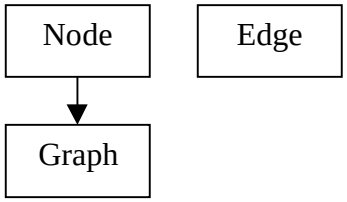


Figure 3.4 Class Hierarchy Diagram

This graph package was initially intended to be distinct from the three-dimensional graph package described below. This was to be done by defining the graph

package as above with out any three-dimensional information except that the Graph class did not extend the Node class. The three-dimensional information was then to be included by extending the graph classes as shown in the following figure 3.5 below where Node is extended by Node3D, Graph is extended by Graph3D, and Edge is extended by Edge3D.

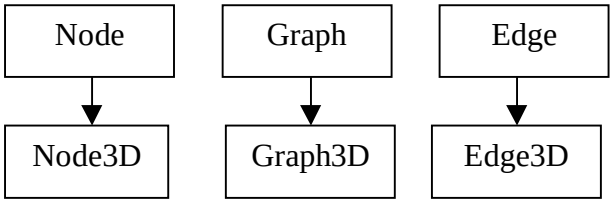


Figure 3.5 Proposed Class Hierarchy Diagram

During the implementation it was decided that it would be nice if the Graph class was extended from the Node class. This allowed graphs to easily be added to another graph a just like a node. This allows graphs to contain subgraphs, which is essential for the hierarchical graph layout algorithm described in section 5.6 below. This change required the class hierarchy shown in figure 3.6. This hierarch requires multiple inheritance which is not provided in Java.

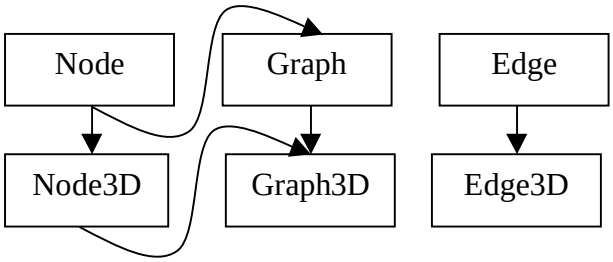


Figure 3.6 Required Class Hierarchy Diagram

It was the decided that using Java interfaces for nodes and graphs to overcome the multiple inheritance problems would be overly complicated. Instead the class hierarchy was collapsed to that shown in figure 3.6 where the base classes include all the required three-dimensional information.

The user of the graph package can still extend the classes for specific applications. This can be done by creating extending classes which specifying the type of the identifiers and attached information. For example the Node class was extended for the network visualisation application by defining a NetNode class. This example restricts the identifier for a node to be an Internet address and the information attached to a node to be an array of Simple Network Management results which are statistics that

are stored for each network node. The Node methods that are used for the extension are defined as follows:

```
public class Node extends Node {
   public Node(Object Id, Object Info, String Label, ...);
   public Object getId();
   public Object getInfo();
   public void setInfo(Object Info);
}
```

The constructor's parameters are abbreviated because they only complicate this example. The implementation of the NetNode class, which extends the Node class, is defined as follows:

```
public class NetNode extends Node {
   public NetNode(InetAddress Address, SnmpResult[] Info,
      ...) {
      super(Address, Info, Address.getHostName(), ...)
   }

   public InetAddress getAddress() {
      return (InetAddress)super.getId();
   }

   public SnmpResult[] getSnmpInfo() {
      return (SnmpResult[])super.getInfo();
   }

   public void setSnmpInfo(SnmpResult[] Info) {
      super.setInfo(Info);
   }
}
```

3.3.2 Graph Package Implementation

The implementation of the graph package is designed to be as general as possible, much like the other standard data structures in Java. The identifier and attached information, for the nodes and edges, are of type Object. This means the graph package can be used for other applications besides the network visualisation program. These uses could be from things like Petri Nets, Entity-Relationship Diagrams, Program Flow Graphs, Data Flow Diagrams, and PERT Diagrams.

The edges and nodes are stored in two Java standard Hashtable objects. This allows quick access of nodes and edges based on their identifiers. The Hashtable class also supplies enumerators over the objects in the hash table, the edges or the nodes, and enumerators over the keys for the objects in the hash table, the identifiers. The most complicated method to implement was an enumerator over the edges from or to a particular node.

3.3.3 Three Dimensional Graph Package

The three dimensional graphing package that was implemented overlays a Java 3D scene graph structure over the standard graph that was described earlier. It does this by adding scene graph nodes to the node and edge classes.

The scene branch for the nodes and edges used in the three dimensional graphing package is shown in figure 3.7 below. There is one of these scene branches for each node and edge in the standard graph.

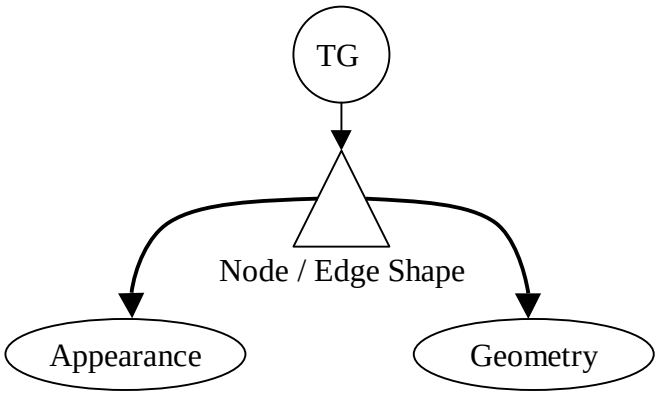


Figure 3.7 Nodes and Edges Branches

The scene branch for a standard graph includes a transform group scene node with the scene branches for the nodes and edges attached as shown in figure 3.8 below.

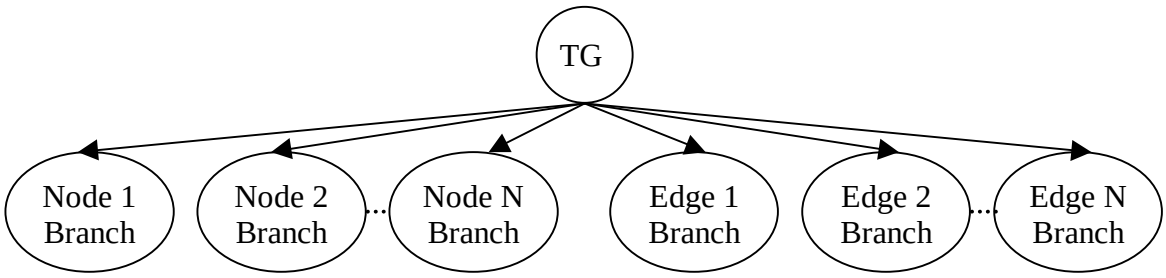


Figure 3.8 Standard Graph Branch

The transform groups at the top of the node and edge branches translate, orientate and scale the nodes and edges with respect to the graph in which they are contained. Similarly, the transform group at the top of the standard graph branch allows an entire graph to be translated oriented and scaled. This is used when a graph is added to a larger graph as a node. The hierarchal structure of the scene graph matches exactly the hierarchal structure of the standard graphs. Finally, to view the entire graph the transform group of the graph's scene branch is added to the geometry branch in the virtual universe shown in figure 3.1.

The translations or positions for the standard graphs nodes are set by the user of the three-dimensional graphing package. The transformations for the edges are calculated by the package so that the edge shape reaches from the source to the destination node as follows. The transformation assumes the edge shape is one unit long in the *z* direction. If an edge is bi-directional that is there is an edge from node A to node B and an edge from node B to node A then the transformations for the edges will be such that the shapes point in the opposite directions.

3.3.4 The Layout Interface

In addition to the three main classes in the graph package an interface Layout was defined. The interface is implemented to layout the nodes and edges in the graph for a specific application. The interface is like the LayoutManager interface defined in the Java AWT package. A graph is allocated a layout, like a window in AWT, and then when nodes and edges are added or removed to or from the graph the Layout interface is called. The Layout interface is defined as follows:

```
public interface Layout {
   void EdgeAdded(Edge e);
   void EdgeRemoved(Edge e);
   void NodeAdded(Node n);
   void NodeRemoved(Node n);
   void setGraph(Graph g);
}
```

A simple example implementation that lays out the nodes in a ring used in the network visualisation application is as follows.

```
public class RingLayout implements Layout {
  // The graph we are laying out
  Graph g;

  public void setGraph(Graph g) {
    this.g = g;
    // Calculate the ring
    InvalidateLayout();
  }

  // Recalculate the ring when a node is added or removed
  public void NodeAdded(Node Node) {
    InvalidateLayout();
  }
  public void NodeRemoved(Node Node) {
    InvalidateLayout();
  }

  // Don't need to recalculate the ring because this
  // layout is not dependant on edges
  public void EdgeAdded(Edge Edge) {}
  public void EdgeRemoved(Edge Edge) {}

  public void InvalidateLayout() {
    // Layout the nodes in the graph g
    // Based on the description in section 5.2
  }
}
```

3.4 Issues with Java 3D

During the implementation of this project, a few issues arose concerning the use of the Java 3D.

The biggest issue with Java 3D was that information could not be reliably added to the scene graph while Java 3D is rendering the scene. The cause of the problem is that there is no synchronisation of access to the objects in the scene graph between the thread rendering the scene and the thread changing the scene. This project requires the ability to add and change information in the scene graph at any time, whereas all the example programs that come with Java 3D build the scene graph before displaying it on the screen and before rendering starts. This cannot be easily overcome because the rendering thread's access to the objects in the scene graph must use synchronisation, which is impossible for the user of the Java 3D package to change.

As a result of this problem, sometimes during the construction of the three-dimensional graph some of the nodes and edges are not added to the scene graph correctly and are therefore not rendered. To overcome this problem the entire exploration of the network can be done before the rendering starts, which reliably builds the scene graph correctly. Ideally it would be nice to be able to add nodes and edges to the graph while the graph is being visualised. This would allow for example, the user to dynamically explore the network by choosing nodes from which to continue the search.

The second issue with Java 3D was the speed of rendering the virtual universe. With about 100 nodes and 100 edges in the scene the maximum frame-rate was reduced to about three frames per second on a Pentium III. This impacted on the usability and interactivity of the program. The slow frame-rate makes it difficult for the user to rotate, zoom in, and navigate the graph.

When complicated graphs are constructed with nodes distributed in three dimensions it can be difficult to determine which nodes are in front of others. It can also be hard to determine which nodes are small but close and which are large and distant. The best cue for the user to determine these things is to move through or around the graph. This movement is made difficult by the slow frame-rate and therefore makes the visualisation of the graph difficult.

The slow frame-rate can also disorient the user. This is because the users control over the environment is not responsive. The user tends to exaggerate or repeat movements of the controls because they are not sure whether or not the program is responding. This leads to the program making large movements through the scene

without rendering any of the frames along the path of movement so the user are not shown how they arrived at their new location or even what there new location is.

The third issue with Java 3D is the amount of memory required to store the scene graph. The example above with about 100 nodes and 100 edges required in excess of 64 MB or memory. This is mainly due to Java 3D not allowing information like the geometry of the objects to be shared between branches in the scene graph. This is because each node in the scene graph is only allowed one parent. This means that large networks cannot be visualised without large amounts of memory.

The efficiency issues of Java3D could be overcome by using “Immediate Mode” rendering. This is a lower level renderer which doesn’t use a scene graph. The speed of rendering the three-dimensional scene could be significantly improved by a factor of about two. Also because this renderer doesn’t require a scene graph shape information for nodes and for edges could be shared. The shape information had to be duplicated using a scene graph because each scene node could only have one parent.

4 Simple Network Management Protocol

4.1 Introduction

The Simple Network Management Protocol (SNMP) is the protocol that was implemented and used to gather information about the network. This information was then used by the network visualisation package. The International Organization for Standardization defined the protocol in 1990. It was primarily designed for remote administration of network devices such as servers, hubs, switches, and routers. These devices run a SNMP service that records the status of the device and responds to SNMP requests. These devices running the SNMP service are referred to as "SNMP agents" [RFC1155].

A package that uses the Simple Network Management Protocol to gather information about the network by communicating with SNMP agents was implemented for this project. This information was then used to generate the layout and generate the statistics for the network visualisation program. The following sections describe how the Simple Network Management Protocol works and how it was implemented for this project. A tool for exploring the Management Information Base for SNMP agents was also developed. This is described in section 4.8 below.

4.2 Management Information Base

The Management Information Base (MIB) is a database for storing managed object definitions. The Simple Network Management Protocol uses Managed Objects for describing the information that can be requested from a SNMP agent.

These managed object definitions are stored textually in a file that is parsed by a grammar that was implemented for this project. The implementation uses a lexical analyser called "JLex version 1.2.5" [JLEX] to read and convert the text file in to tokens. The implementation then uses a compiler constructor tool called "Java CUP version 0.1j" [JCUP] to produce the grammar parser. An example of the textual definition of a managed object is in section 4.2.3 below.

4.2.1 Object Identifiers

Object identifiers are a means for uniquely identifying a managed object. An object identifier is a sequence of integers and may be assigned a textual name for simplicity. The sequence of integers traverses a global tree of Managed Objects. The

tree consists of a root connected to a number of labelled nodes via edges. Each node may in turn, have children of its own which are also labelled. In this case, the node is termed a subtree. This process may continue to an arbitrary level of depth [RFC1155]. A tree of standard Object Identifier is included in the RFC1213 and is partially shown in figure 4.1. This figure is taken from the SNMP tool described in section 4.8 below.

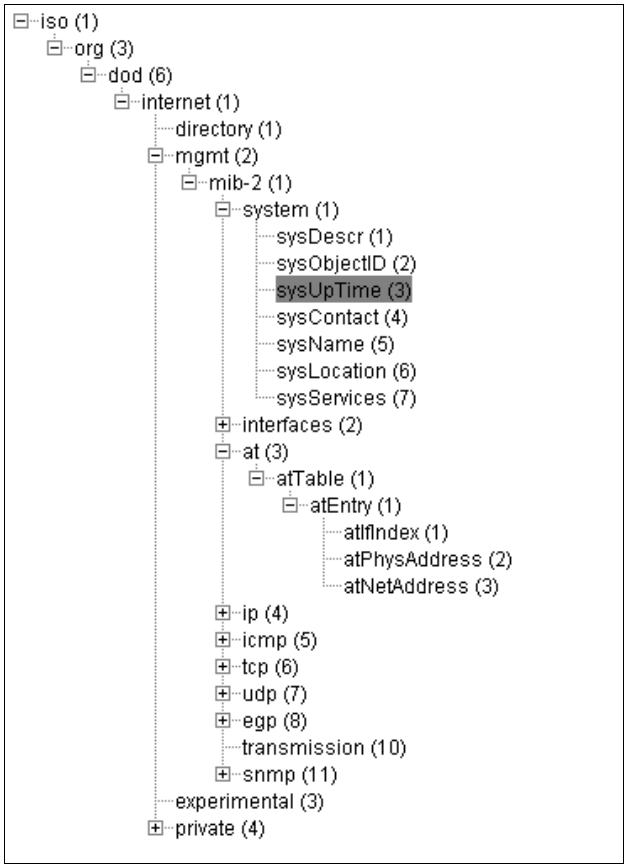


Figure 4.1 Object Identifier Tree

The root node of the tree is administered by the International Standards Organization (ISO) and is designated the sequence of integers “1” and the name “iso”, which is abbreviated as “iso(1)”. Under the iso(1) node the International Standards Organization has designated one subtree administered by international organisations org(3). Under the org(3) node the United States Department of Defence has been allocated dod(6). So following the nodes from the root node to the dod node gives the sequence of integers 1.3.6 which uniquely identifies the dod node.

As an example the highlighted node in the figure 4.1 is named “sysUpTime” and has 1.3.6.1.2.1.1.3 as its Object Identifier.

4.2.2 Managed Objects

A managed object defines the information that can be requested from a SNMP agent. Managed Objects may contain the following fields [RFC1155]: The examples for each field are for the Managed Object identified by “sysUpTime”.

Object Descriptor: A unique textual name for the managed object. From the example: “sysUpTime”

Object Identifier: A unique object identifier for the managed object. From the example: “1.3.6.1.2.1.1.3”.

Syntax: The abstract syntax or type for the object defined using the Abstract Syntax Notation as described below. From the example: “TimeTicks” which is an integer that counts the time in hundredths of a second since some epoch.

Description: A textual description of the information the managed object represents. From the example: “The time (in hundredths of a second) since the network management portion of the system was last re-initialized.”

Access: One of *read-only*, *read-write*, *write-only*, or *not-accessible* defining how SNMP agents can interact with the managed object. From the example: “read-only”

Status: One of *mandatory*, *optional*, or *obsolete* defining whether or not the managed object must be defined in an SNMP agent. From the example: “mandatory”

4.2.3 Management Information Base Definitions

Management Information Base definitions define the Managed Objects in the Management Information Base. The definitions are created and administered by the appropriate originations. The definitions are stored and distributed as text. The following is an example taken from RFC1213 for the Managed Object “sysUpTime”.

```
sysUpTime OBJECT-TYPE
  SYNTAX  TimeTicks
  ACCESS  read-only
  STATUS  mandatory
  DESCRIPTION
    "The time (in hundredths of a second) since the
    network management portion of the system was last
    re-initialized."
  ::= { system 3 }
```

The object identifier for “sysUpTime” is shown on the right hand side of the assignment statement “::=” as system.3. This can be reduced to a string of integers as follows:

sysUpTime = system.3 = mib-2.1.3 = mgmt.1.1.3 = … = 1.3.6.2.1.1.3

A Management Information Base should contain all the Managed Objects defined in RFC1213.

4.3 SNMP Commands

There are four main commands that can be sent to an SNMP agent. These commands are as follows:

Firstly, the “Get” command posts a request to an SNMP agent to get the value of a Managed Object. The request contains the Object Identifier that is being requested. The agent receives the request then looks up the value in its Management Information Base and responds with the value. This request should only be performed on Managed Objects with access equal to *read-only* or *read-write* otherwise the SNMP agent will respond with an error.

Secondly, the “GetNext” command is like the Get command except the SNMP agent responds with the value and Object Identifier for the next Managed Object in the Management Information Base after the Object Identifier in the request. For example, “sysObjectID” is defined as “system.2” which is next after “sysDescr” which is defined as “system.1”.

Thirdly, the “Set” command posts a request to an SNMP agent to set the value of a Managed Object in the Management Information Base. The agent then sets the value of the Managed Object and responds with a success or failure code. This request

should only be performed on Managed Objects with access equal to *read-write* or *write-only* otherwise the SNMP agent will respond with an error.

Lastly, the "Walk" command is used to explore all the Managed Objects in a subtree of the Management Information Base. The Walk requests can not be directly sent to a SNMP agent. Instead the request starts from a specified Object Identifier and posts a number of GetNext requests until the entire subtree under the Object Identifier has been explored. For example, a Walk request starting from "system" will result in "sysDescr", "sysObjectID", "sysUpTime", "sysContact", "sysName", "sysLocation", and "sysServices" which can be seen in figure 4.1 above. The results of this walk, using the University of Adelaide's Power Hub as the SNMP agent, and taken from the SNMP tool, are shown in figure 4.2 below.

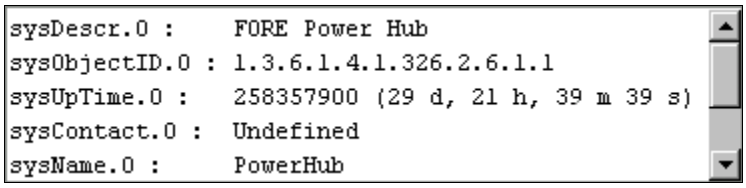


Figure 4.2 Walk Starting from "system" Object Identifier

The Walk command is also used for traversing tables in the Management Information Base. For example the Managed Object identified by "atTable" shown above in figure 4.1 is the address translation table for the SNMP agent. This is used to translate between physical addresses and Internet addresses. This table will contain one entry for each interface attached to the SNMP agent. The Managed Objects "atIfIndex", "atPhysicalAddress" and "atNetAddress" acts as records inside the table. Each of these three Managed Objects are effectively separate tables which are logically grouped into one table.

The entries in these tables typically have the index of the entry appended to their unique Object Identifier, for example "atIfIndex.1" would be element 1 in the "atIfIndex" table. This indexing convention is not always used and therefore accessing the tables by appending the index as described cannot always be done. For example Internet Protocol numbers are used to index and as data in the "atNetAddress" table. This is shown in figure 4.3 below, using the University of Adelaide's Power Hub as the SNMP agent, taken from the SNMP tool.

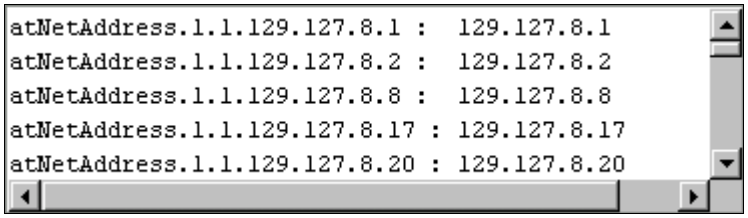


Figure 4.3 Walk Over the "atNetAddress" Table

This means that these entries can only be accessed using a GetNext request, or to access all the entries in the table, a Walk request because the indexing method for a table cannot be assumed. Also the fact that these Managed Objects contain tables is not defined by any SNMP definitions, it only stated in the textual description of the "atTable".

These SNMP requests also contain an octet string called "community name". This string is used to authenticate the use of the command. There may be different community names for Get and Set requests and different community names to access different Managed Objects. If the request contains an incorrect community name the SNMP agent will respond with an error. The community name for Get access to most SNMP agents is the string "public".

The requests and responses are encoded into a string of bytes, as defined in the following section, then put in a Internet datagram packet and sent using the standard Internet Protocol on port 161.

4.4 Data Types and Encoding

The data types and encoding used in the Simple Network Management Protocol are defined in the RFC1157. They use the Abstract Syntax Notation One (ASN.1) standard and the Basic Encoding Rules (BER) standard to serialise the SNMP requests and responses for transmission across networks.

4.4.1 Abstract Syntax Notation

The Abstract Syntax Notation One (ASN.1) is a standard that defines basic types and encoding rules for data objects. The Simple Network Management Protocol uses a subset of the ASN.1 standard to encode the messages passed between SNMP agents. The following machine independent basic types are defined by the ASN.1: Boolean, Integer, Bit String, Octet String, Null and Object Identifier. The standard also defines Sequence and Set which are used to group the basic types into structured types.

From these basic types, any record based structure can be made. The Simple Network Management Protocol restricts the Abstract Syntax Notation by types excluding the use of Boolean, Bit String, Sequence and Set for simplicity and to prevent aggregate types. SNMP also adds the following types [RFC1157]:

IpAddress: This type represents a 32-bit Internet address. It is represented by an Octet String of length 4.

Counter: This type is represented by a non-negative 32-bit integer which monotonically increases until it reaches a maximum value of 2^{32}-1, when it wraps around and starts increasing again from zero.

Gauge: This type is represented by a non-negative 32-bit integer which may increase or decrease, but which latches at a minium value of zero and latches at a maximum value of 2^{32}-1.

TimeTicks: This type is represented by a non-negative 32-bit integer which counts the time in hundredths of a second since some epoch. When object types are defined in the MIB which use this ASN.1 type, the description of the object type identifies the reference epoch.

4.4.2 Basic Encoding Rules

The Basic Encoding Rules (BER) are used to convert the ASN.1 types and their values into a stream of octets that can be transmitted over a network.

The first step in the encoding uses the concept of an ASN identifier that is uniquely assigned to every data type. The identifiers are predefined byte values, for example the type Boolean is represented by 0x01, Integer is represented by 0x02 and so on [BER]. This identifier is the first byte in the stream.

The second step is to calculate the length in bytes of the data associated with the type. The method of doing this is specific to the data type a few examples follow:

Null: the length of null is defined as zero.

Integer: the length of an integer is based on the number of significant bytes used by the value of the integer. The range 0..255 is of length 1, the range 256..2^{16}-1 is of length 2 and so on.

Octet String: the length of an octet string is the number of octets contained in the string.

These length values are represented differently depending on their size. If the length value is less then 127 than the byte with the length value is appended to the stream. Otherwise, the byte with the most significant bit set ored with the number of bytes that the integer representing the length will take. This byte and the bytes representing the length are then appended to the stream. For example:

Length of 28 is converted simply in to the byte 0x1C.

Length of 523 is converted into the byte with the most significant bit set 0x80 bit wise ored with 2 the number of bytes used to fit the integer number 523 which is 0x02 to get 0x82. This is then followed by the value of the length 0x020B (523).

The third step in the conversion is to convert the value of the data into a stream of bytes. Again, this is dependant on the type of the data. The following examples describe two conversions:

Integer: the value of an integer is converted by putting the most to the least significant byte of the integer, starting from the byte indexed by the length as described above, onto the stream.

Octet String: the value of an octet string is converted by putting each byte in the string onto the stream from first to last.

The types Counter, Gauge and TimeTicks, which were added by SNMP, are all converted like an Integer except that their identifier bytes are 0x41, 0x42 and 0x43 respectively. The IPAddress type is converted like an Octet String with an identifier of 0x40.

The SNMP standard also uses the BER defined a type “Sequence” for defining its protocol data unit as described below. The Sequence type cannot be used to define a managed object. The sequence type is used to describe compound structure based data types. The Sequence type is converted in to a stream as follows. The first byte is the ASN identifier 0x10. This is followed by the size in bytes of all the data elements in the sequence including their identifier and size bytes. This followed by the byte stream of the data elements in the sequence. Sequence types can be nested inside each other to make arbitral complicated data types.

4.4.3 Protocol Data Unit

The Protocol Data Unit (PDU) is used to transmit requests to and from SNMP agents. The PDU is defined in terms of, and converted into a stream of bytes using, the Basic Encoding Rules. These PDUs are sent using datagram packets using the Internet protocol on port 161. They are sent from the user to the SNMP agent for Get, GetNext and Set requests. The SNMP agent then sends a Response to back to the user.

The PDU is defined as the following structure [PDU]:

```
Sequence {
  Integer version
  Octet-String community-name
  PDU-Type | Sequence {
    Integer request-id
    Integer error-status
    Integer error-index
    Sequence {
      Sequence {
        ObjectID oid;
        Objecttype value;
      }
      ...
    }
  }
}
```

Where:

Version: is the version of SNMP for which this PDU was created. For the purposes of this project SNMP version 1 was used which is represented by the integer zero as defined using the BER.

Community Name: is the string used to authenticate the SNMP request as explained in section 4.3 SNMP Commands above.

PDU-Type: is the byte encoding for the type of SNMP request as follows: Get requests are 0x00, GetNext requests are 0x01, Responses are 0x02 and Set requests are 0x03.
This value is bit wise ored with the Sequence ASN identifier which is 0x10.

Request-Id: this is an integer containing a unique number to identify the PDU. This number is copied from the request PDU into the response PDU so that the user can match the request with the response.

Error-Status and Error-Index: these are used by the SNMP agent in the response PDUs to indicate that an error occurred. They should be set to zero for request PDUs.

Object Id and Value: this is a sequence of Object Ids and their respective values. These values are interpreted depending on the type of PDU. If it is a "Set" PDU then the Managed Objects identified by the Object Ids should be set to the values. If the PDU is a response then the values are what the Managed Objects currently contain. The values should be the Null type for Get and GetNext PDUs.

4.5 Extensions to SNMP Commands

In additional to the implementation of the above specification of the Simple Network Management Protocol, the following statistics-gathering commands based on the Walk command were added. These features were added for use in the network monitoring and visualisation program.

Sum: This command takes the result of a Walk command; sums all the Managed Objects of type integer received and returns the result.

Count: This command takes the result of a Walk command and returns the number of Managed Objects received.

Average: This command takes the result of a Walk command; sums all the Managed Objects of type integer received; divides this by the number of Managed Objects received and returns the result of the division.

Max: This command takes the result of a Walk command and returns the maximum value of all the Managed Objects of type integer received.

Min: This command takes the result of a Walk command and returns the minimum value of all the Managed Objects of type integer received.

These results can be used, for example, to calculate the total possible output from an SNMP agent. The Managed Object “ifSpeed” contains the nominal bandwidth in bits per second of one interface attached to the SNMP agent. There is one “ifSpeed” object for each interface. Posting a Walk command using the Managed Object will result in a list of the speeds of all the interfaces attached to the SNMP agent. The Sum command would then sum this list and return the total bandwidth for the SNMP agent.

4.6 Simple Network Management Protocol Issues

During the implementation of this project a few issues arose concerning the use of the Simple Network Management Protocol.

The first issue is that SNMP has poor type checking rules, even though the predefined Managed Objects have a defined type.

The first type-checking problem is caused by poor implementations of SNMP agent’s software. Many agents reply to requests with information of a different type to the definition of the associated managed object.

The second type-checking problem is because many of the Managed Objects in the agents Management Information Base are not predefined in the users Management Information Base. This is caused by different organizations and companies defining their own Managed Objects. The user can access these Managed Objects without knowing their definition using the GetNext and Walk commands. The type information for the object is included in the response but cannot be checked by the user because they do not know the definition of the managed object.

The second issue is that while the Simple Network Management Protocol is very useful for gathering fairly static information like the interconnections between agents, it doesn’t allow for much dynamic information such as the current input and output load on an agent. This makes SNMP useful for this project for mapping the networks but not very useful for network monitoring. A possible solution to this problem is proposed in section 4.7 below.

The last issue with the SNMP protocol is that most of the useful information required for this project is contained in tables of Managed Objects. The information in these tables can only be accessed by using GetNext and Walk commands. These commands require many requests and responses to be sent between the agent and the user. Over a large network these messages could have a high latency attached to them which could dramatically slow down the network monitoring software.

This problem could overcome by not defining the Walk command as a basic command like Get instead of defining it in terms of many GetNext commands. This could easily be done by slightly changing the SNMP standard to support the Walk command so that when it is sent to an SNMP agent the agent responds with all the information in the subtree under the requested Managed Object. The PDU that the SNMP agent responds with, defined in section 4.4.3 above, supports multiple Object Identifier and their associated values and could easily contain a whole table. This large PDU may cause other problems because the largest allowable PDU is defined as 2048 bytes in size. This size would also need to be increased for this solution to work.

4.7 Other Possible Extensions to SNMP Commands

To overcome the lack of dynamic information that can be accessed using the Simple Network Management Protocol an expression-based system could be added to the statical package. The user software could take predefined Managed Objects and compute the required dynamic information.

The following example shows how dynamic information about the total load on a SNMP agent can be obtained by using the predefined Managed Objects *sysUpTime*, *ifInOctets*, *ifOutOctets* and *ifSpeed*.

The current input and output for an interface attached to a SNMP agent can be defined as follows:

$$InputDifference = CURR(ifInOctets) - PREV(ifInOctets)$$
$$OutputDifference = CURR(ifOutOctets) - PREV(ifOutOctets)$$
$$TimeDifference = CURR(sysUpTime) - PREV(sysUpTime)/100$$
$$ifIOPerSec = \frac{InputDifference + OutputDifference}{TimeDifference}$$

Where:

ifIOPerSec is an estimation of amount of input and output the interface is currently doing in octets per second. The difference in the *sysUpTimes* is divided by 100 because to convert the time from hundredths of a second to seconds.

CURR is the current value of the managed object.

PREV is the value of the managed object the previous time it was requested or zero if the managed object has not been requested before

ifInOctets is the managed object that stores the number of octets the interface of the agent has received.

ifOutOctets is the managed object that stores the number of octets the interface of the agent has sent.
sysUpTime is the time since the SNMP was initialised in hundredths of a second.

The current load on an interface attached to a SNMP agent can be defined as follows:

$$ifLoad = \frac{ifIOPerSec}{ifSpeed / 8}$$

Where:
ifLoad is an estimation of the current load on an interface as a percentage. The *ifSpeed* is divided by 8 to convert it from bits to octets.
ifSpeed is the nominal bandwidth of an interface in bits per second.

The current total load on a SNMP agent can be defined as follows:

$$sysLoad = AVERAGE(ifLoad)$$

Where:
sysLoad is an estimation of the current load on the SNMP agent.

4.8 SNMP Tool

A Simple Network Management Protocol tool was also developed to explore the Management Information Base on a SNMP agent. The tool is a Java graphical user interface (GUI) based application which allows the user to post Get, GetNext, Set and Walk commands using a specified managed object and a specified SNMP agent. The following figure 4.4 shows a screen shot of the SNMP tool:

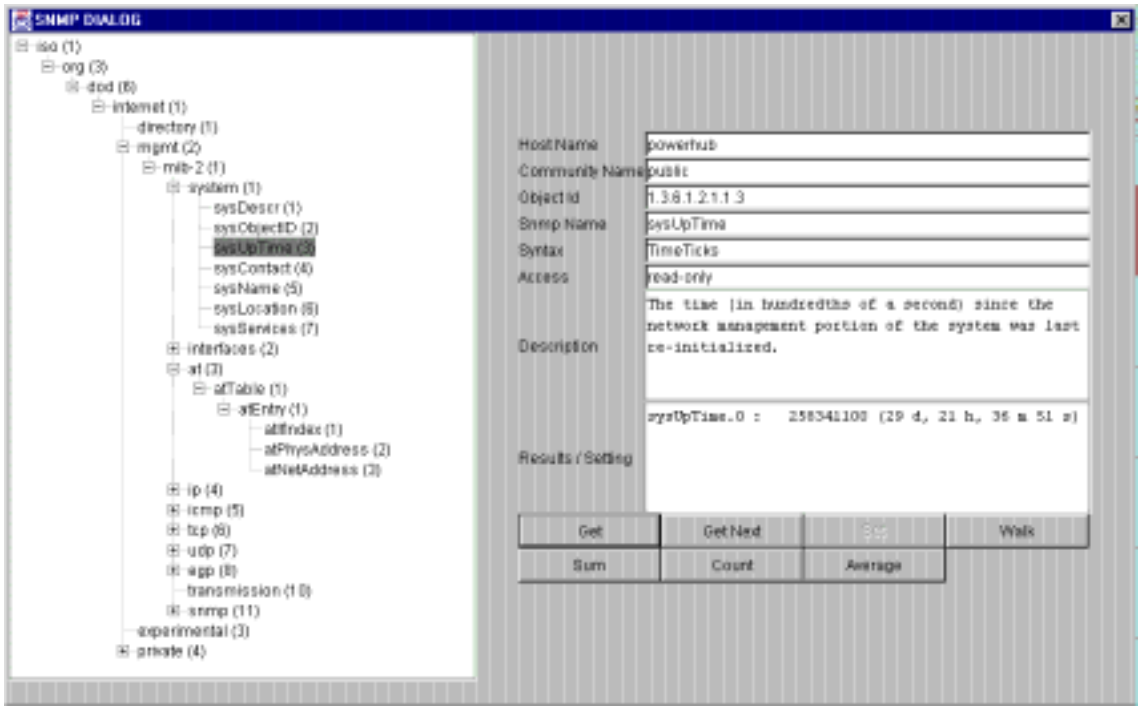


Figure 4.4 Simple Network Management Tool

The tool shows the Management Information Base organised in the tree hierarchy on the left side of the GUI. The tree shows the integer identifier and the name for the nodes. When the user selects a Managed Object in the tree, for example “sysUpTime” is selected in the figure, the right hand side of the GUI is filled in. The object identifier, name syntax, access and description text boxes are filled in based on the Managed Object definition. The user can also type in an object identifier or name into the text boxes and the appropriate Managed Object in the tree will be selected. The user then enters the host name or Internet address of the SNMP agent. The user can also enter the community name if required. The user can then click on a button for the specified commands and the results of the command will be shown in the results text box. The results in the figure above show a Get command using the Managed Object “sysUpTime”. This shows the integer representing the time and the integer converted into days, hours, minutes, and seconds for the Power Hub SNMP agent.

5 Graph Layout Algorithms

5.1 Introduction

Graph layout algorithms were used to layout the graphs and nodes into an easy to visualise pattern. A good layout can help people to understand the application, but a bad diagram can be misleading [HMM98]. The following sections describe and evaluate the quality of some of the algorithms that were tried and implemented. The figures of the laid out graphs use the same network information. This information was taken from the Computer Science Department at University of Adelaide's network. The blue nodes in the figures are Internet hosts that did not respond to SNMP requests so no statistics could be gathered from them. Their names have been removed to simplify the images.

5.2 Ring Layout

The first and simplest layout algorithm implemented was the "ring" algorithm. This algorithm simply takes the nodes in a graph and distributes them in a ring. The steps involved in layout process are as follows:

The first step is to calculate the radius of the ring as the following figure 5.1 shows.

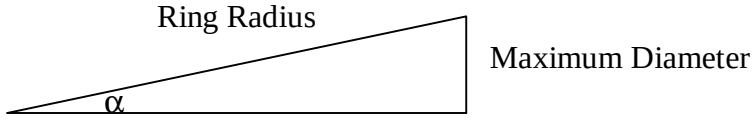


Figure 5.1 Ring Radius Calculation

Where:

Maximum Diameter is the diameter of the largest node in the graph.

α is $2\Pi \Big/ \text{The number of nodes in the graph}$

Ring Radius is $(\text{Spacing Ratio} + 1) \times \text{Maximum Diameter} \Big/ \sin(\alpha)$

Spacing Ratio is the amount of space between the nodes as a proportion of the size of the largest node. For example, 0 would be no space between the nodes and 2 would be twice as much space as the radius of the largest node between the nodes.

The second step in the layout is to position the nodes. This is simply done by distributing each node around a circle using the ring radius calculated above.

The results of a ring layout are shown in figure 5.2 below. This figure shows a problem with this simple layout method which is there are many crossing edges between the nodes. This is because the method does not account for which nodes are connected to each other.

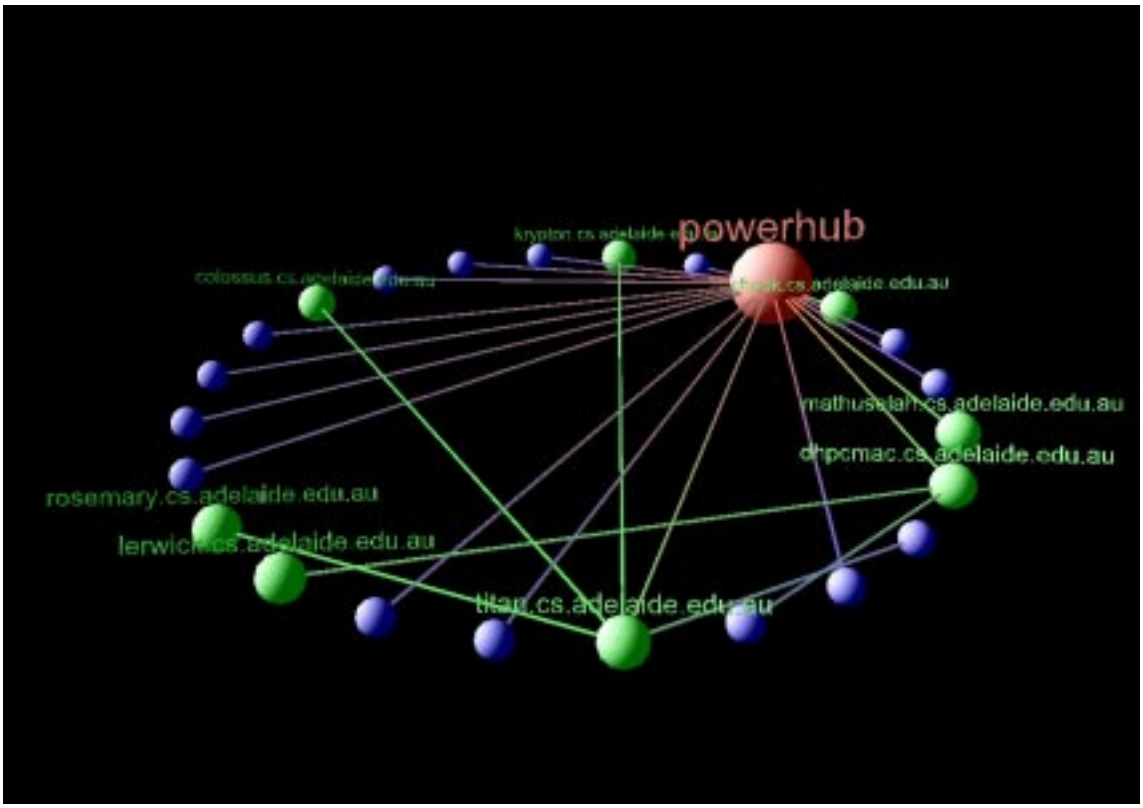


Figure 5.2 Ring Layout

Sorting the nodes based on their connections and then using the ring layout method can overcome this problem. This sorting can be done by building a list of nodes in a recursive fashion starting from the most connected node as follows:

For all the nodes connected to the current node add the connected nodes to the list on either side of the current node. Then for each connected node repeat the step recursively.

For example using the SNMP nodes shown in figure 5.2 and starting from the "powerhub" node the following steps are taken.

powerhub

For each node connected to the "powerhub" add the nodes to either side. That is "chook" to the left, then "mathuselah" to the right, then "dhpcmac" to the left, and "titan" to the right.

dhpcmac	chook	powerhub	mathuselah	titan

Then for each of those nodes repeat the process recursively. For example "chook" and "mathuselah" don't have any attached nodes so the recursion stops. The "dhpcmac" has "lerwick" connected to it so "lerwick" is added to the left. The "titan" node has "krypton", "colossus", and "rosemary" connected to it so they are added to the left and right of "titan". The result after the recursion is the list as follows:

lerwick	dhpcmac	chook	powerhub	mathuselah	rosemary	krypton	titan	colossus

The nodes in this list can be laid out using the ring algorithm above. The sorting results in fewer crossed edges in the graph as can be seen in figure 5.3 below. The only problem with this layout is that the edges to the "powerhub" node are bunched together which makes them hard to follow. The next layout algorithm solves this problem.

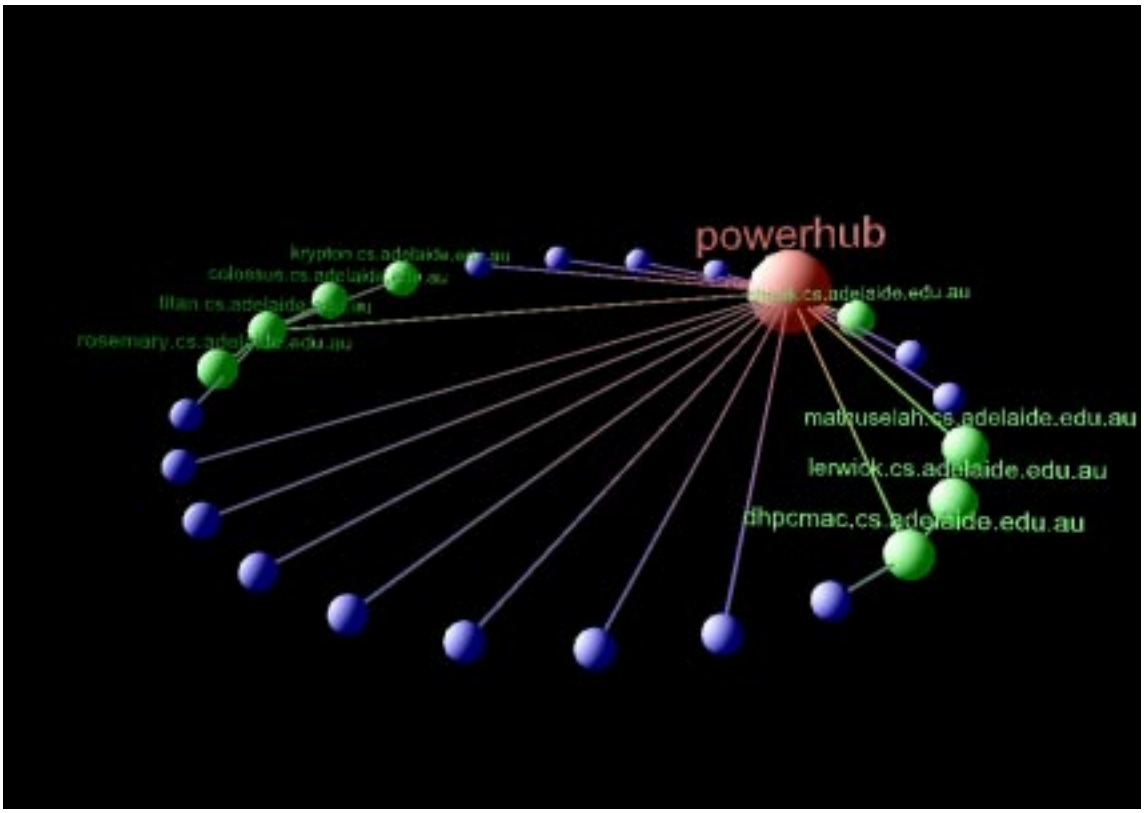


Figure 5.3 Sorted Ring Layout

5.3 Star Layout

The next layout algorithm that was implemented is the “star” algorithm. This is a slight variant on the ring algorithm.

It starts by working out the most connected node, that is the node which has the most connections to the other nodes. It then forms a ring using the remaining nodes as described in section 5.2 above and then places the most connected node in the middle. This can be seen in figure 5.4a below.

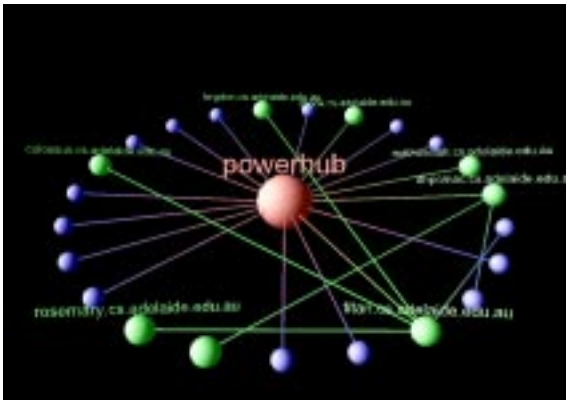


Figure 5.4a Star Layout

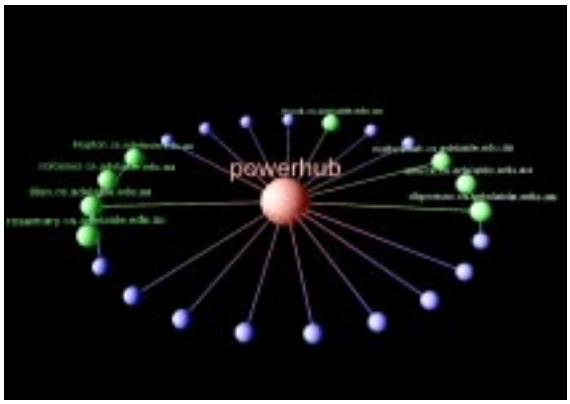


Figure 5.4b Sorted Star Layout

The star algorithm can also benefit from the same sorting technique described in section 5.2 above. This is shown in figure 5.4b with fewer crossing edges. This layout algorithm provides a simpler view of this network because the edges to the “powerhub” node are more widely spaced and are easier to follow.

The ring and star algorithms work well with a small number of nodes and edges. When the graph becomes larger the layout of the graph also becomes larger to accommodate the increase in nodes. With more edges in the graph the likelihood that sorting methods will prevent crossed edges decreases. These two problems can lead difficulties for the user in understanding the graph.

5.4 Sphere Layout

Another technique is to layout graphs in 3D instead of 2D. The extra dimension gives more space that would ease the problem of displaying large graphs. Displaying graphs in 3D can also introduce new problems because node can occlude one another, and it is also difficult to choose the best view in space [EHM98].

The next layout algorithm that was implemented, called the “sphere” algorithm, uses this technique. This algorithm evenly lays out the nodes on the surface of a sphere using a latitude and longitude style coordinate system as follows:

Firstly the number of latitude and longitude segments is calculated. This is done by starting with the number of nodes in the graph then subtracting two, for the north and south poles. This result is divided by three to get the number of latitude rings, and this result divided by three and multiplied by two to get the number of longitude lines. This gives a ratio of one latitude rings to every two latitude lines which gives a nicely spaced out layout.

The number of latitude rings is then used to calculate the radius of each latitude ring. The nodes are then divided up into the latitude rings and distributed based on the different radius for each ring as before.

The results of this sphere layout algorithm are shown below without sorting in figure 5.5 and with sorting in figure 5.6.

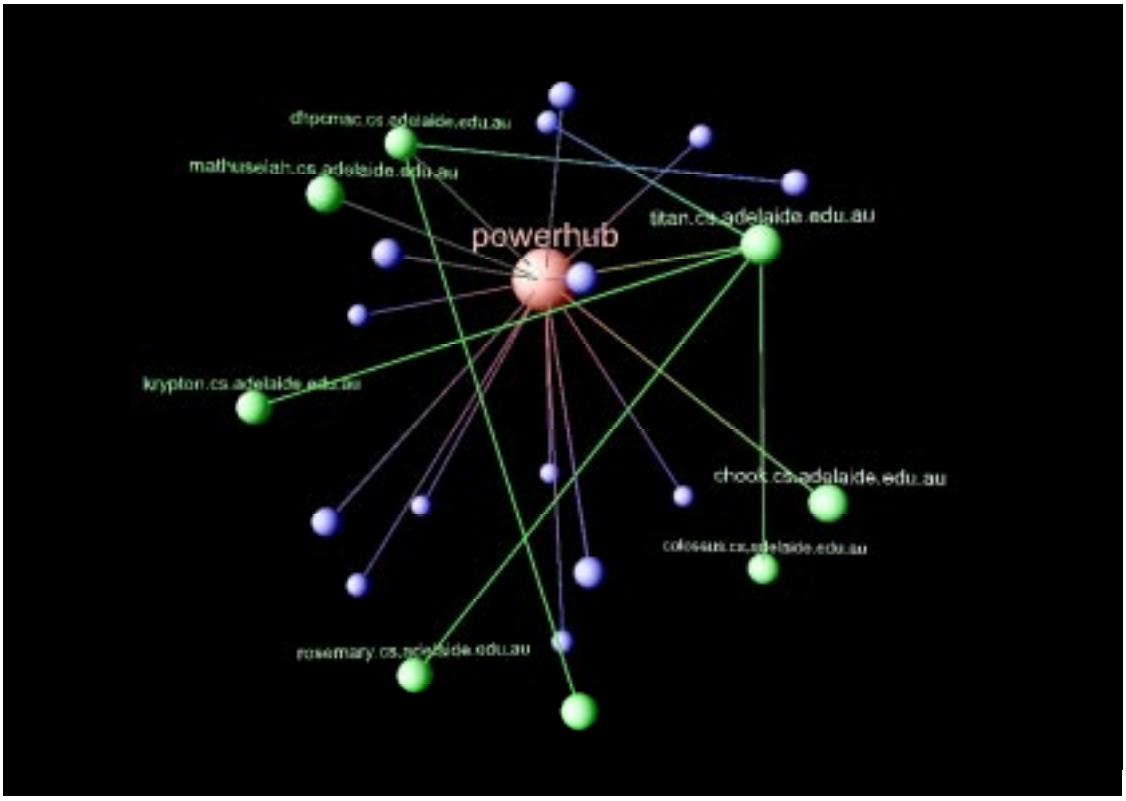


Figure 5.5 Sphere Layout

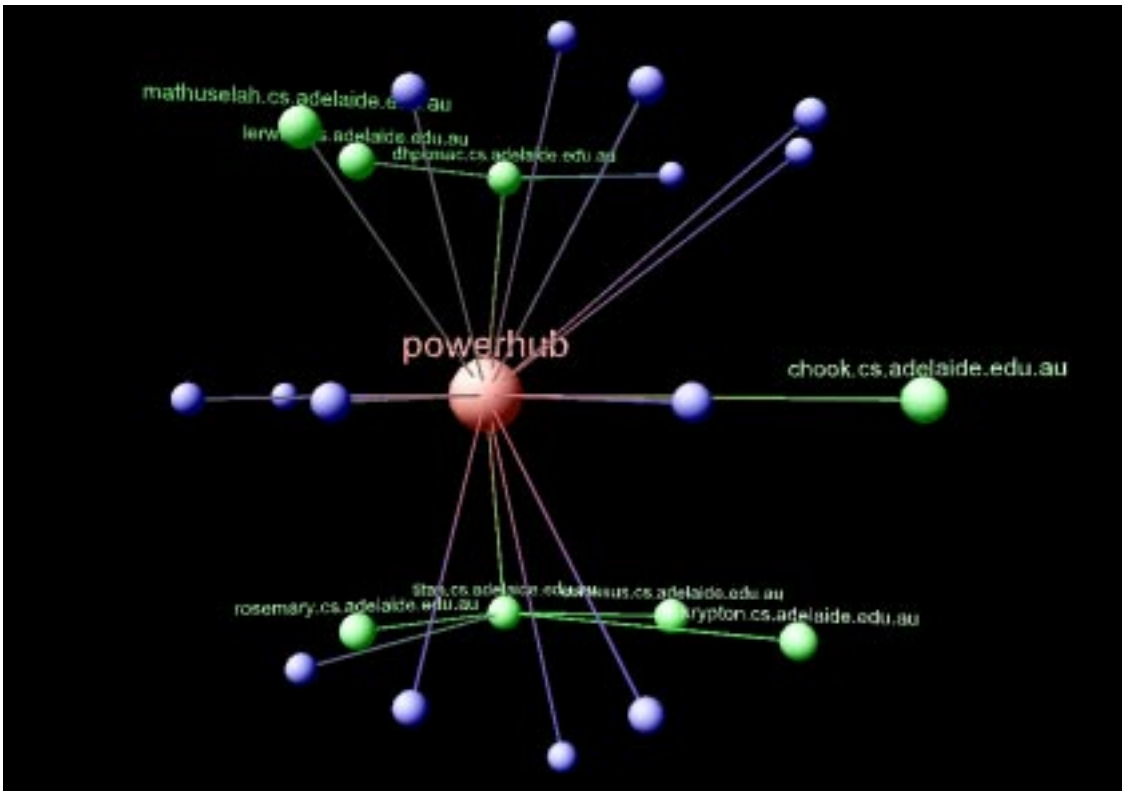


Figure 5.6 Sorted Sphere Layout

The sphere layout algorithm can also benefit by placing the most connected node in the centre much like the ring algorithm. The results of this are shown below without sorting in figure 5.7 and with sorting in figure 5.8.

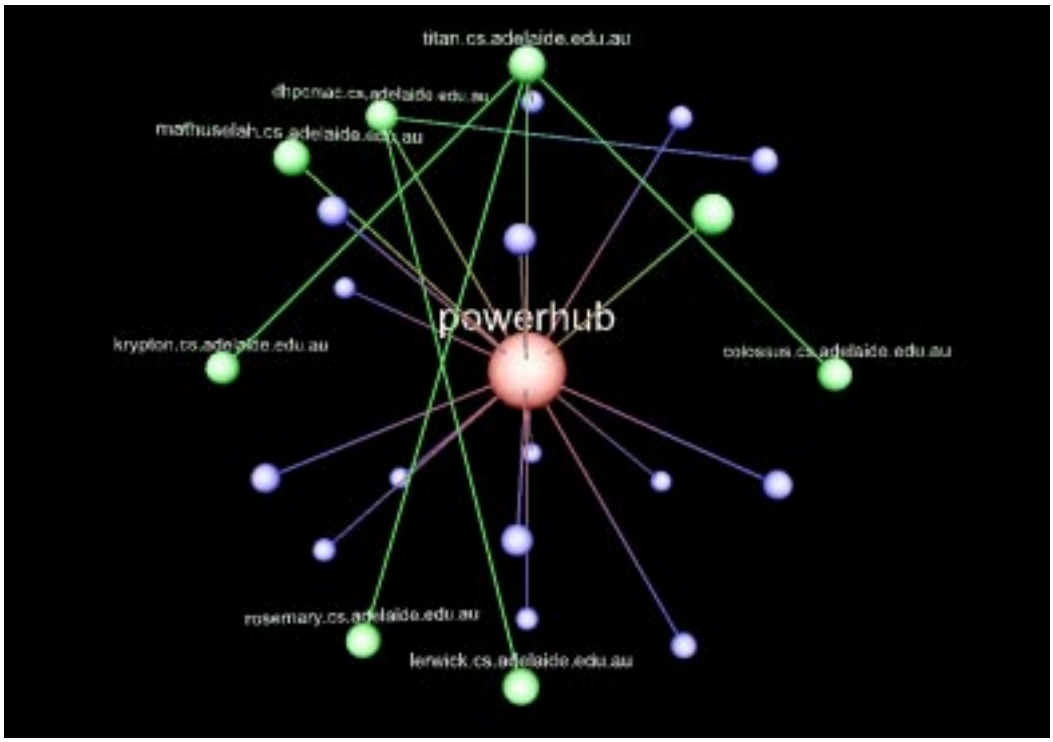


Figure 5.7 Central Node Sphere

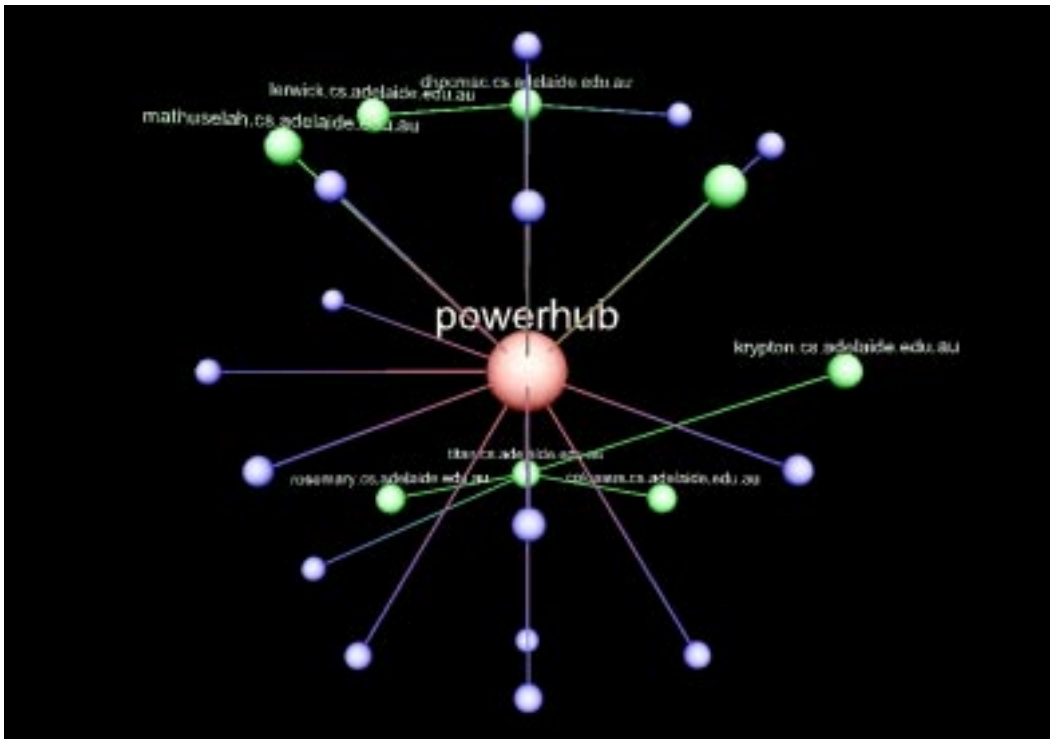


Figure 5.8 Sorted Central Node Sphere

5.5 IP Number Layout

Many other layout algorithms can be done based on the information the graph contains rather then just the nodes and edges. For the example of network visualisation information like the Internet Protocol (IP) number can be used to layout the graph.

This can be done by taking an IP number to be the centre of the layout. Then for each node in the graph assign its position based on the difference between the components of its IP number and the components of the central IP number.

For example for the Computer Science Department at the University of Adelaide an appropriate IP number would be "129.127.8.128". Then for each of the node, for example the "powerhub" with an IP number of "129.127.8.99", take the central IP number from its IP number, for example "129.127.8.99" – "129.127.8.128" = "0.0.0.–29". The last three digits of the resulting IP number can then be used as the x, y, and z coordinates of the node.

This will result in closely related nodes being laid out in lines. This is because they will have all but the last component of their IP numbers the same. The overall result will be a large three-dimensional grid of nodes. This layout has the problem that two nodes, for example "129.127.8.99" and "234.127.8.99", would be placed in the same position. This can easily be over come by using the first component of the IP number as a position index into a larger grid of networks.

5.6 Hierarchical Layout

The above layout algorithms work well for a small number of nodes and edges. For larger graphs the number of nodes increase the sizes of the rings and spheres. Larger graphs also have a large number of edges which will inevitably cross each other. This makes following the crossed edges across the large graphs difficult.

To over this problem the graphing package allows graphs to contain subgraphs as well as node. This allows the nodes in the graph to be organised into a hierarchical structure. The main graph and the subgraphs can all be laid out independently and can even use different layout algorithms. The following figure 5.9 shows a hierarchical layout of over eighty nodes. The main graph and each subgraph are arranged into stars. The main graph contains some of the SNMP agents connected to the powerhub. The subgraphs show the non-SNMP hosts attached to the SNMP agents.

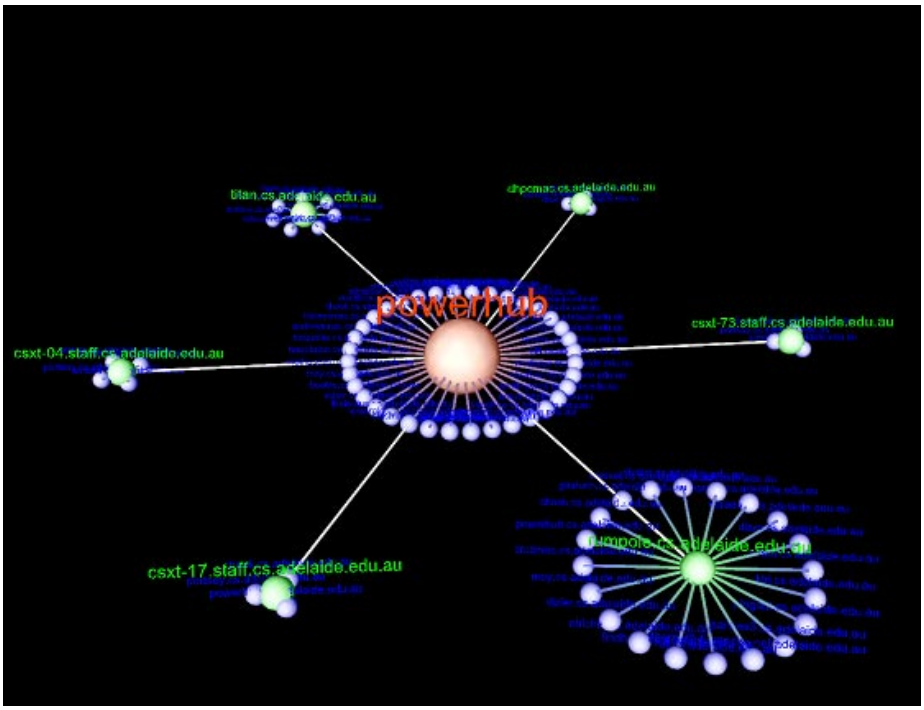


Figure 5.9 Hierarchical Layout

5.7 Layout Issues

Different layout algorithms are applicable to different applications. The layouts described above are suitable for mapping local area networks with a few somewhat centralised hubs.

On a larger scale decentralised network, for example all of Australia, different layout algorithms may be more appropriate. This is kind of network could use the mechanical spring technique which lays out a graph based on the length of the edges between the nodes [KF96]. The edges act as springs joining the nodes and the layout is determined by the positions of the nodes that has the minimum tension on the springs. The following 5.8 figure shows an example of the spring layout distributed with the standard Java Development Kit [JDK]. For the purposes of network mapping the length of the edges could be given by the ping time between the different nodes in the graph.

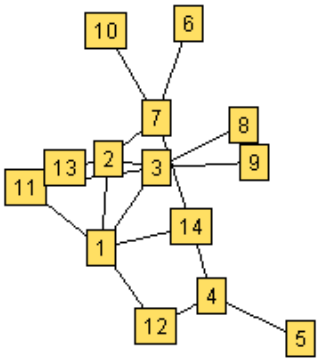


Figure 5.8 Example Spring Layout

Another interesting layout for large-scale networks would be to use the nodes actual geographic location and put them on a sphere of the earth. The geographic location information is not available using the Simple Network Management Protocol but could easily be added be defining another standard managed object.

On large-scale networks user may only be interested in the larger nodes rather then all of the small nodes. This could be done by collapsing all the small nodes in a subgraph and only display it as one node in the super graph. The user could then expand the subgraphs to display the detailed information about the small nodes inside if they wish. The network mapping software could also simply ignore nodes that are smaller than a specified size to reduce the number of nodes in the graph.

Other layout algorithms would be needed for different application domains. For example it would not be suitable for laying out chemical elements, as a graph of atoms joined by bonds, in a star. The layout required would account for the specific rules used in modeling the elements.

6 Computer Network Visualisation Software

6.1 Introduction

The Computer Network Visualisation software uses the packages described above to create a network mapping and three-dimensional visualisation program. The program also provides a simple user interface so that the user can interact with the mapped network. The following figure 6.1 shows the final product, which has mapped some of the computers in the Computer Science Department at the University of Adelaide.

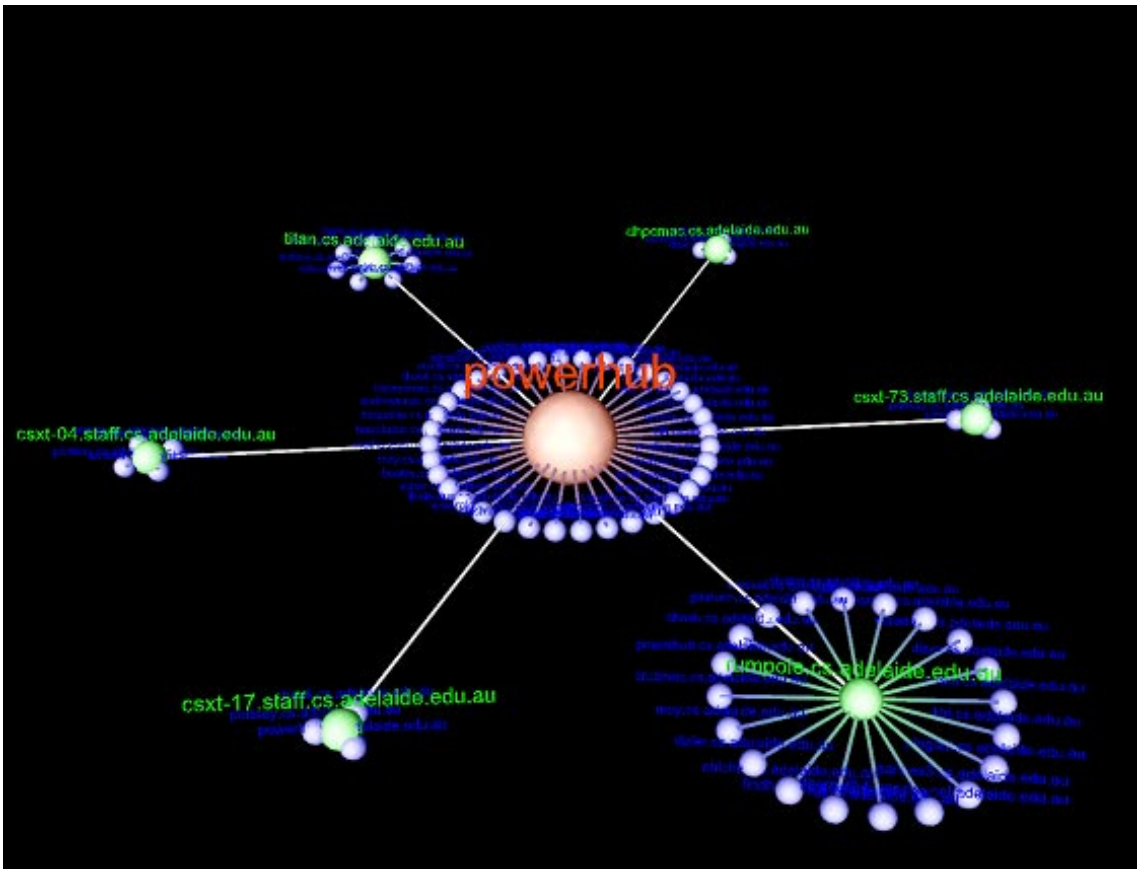


Figure 6.1 Computer Network Visualisation User Interface

6.1 Network Mapping

The Computer Network Visualisation software uses the Simple Network Protocol, described in section 4 above, to map and gather information about the computer network. The mapping works by initiating a depth first search starting from a

specified SNMP agent. It starts the depth first search by posting a Walk using the SNMP Managed Object “atNetAddress”. This Managed Object contains a table of the Internet address of all the computers attached to a SNMP agent. It then recursively starts the search again from each of attached addresses to a specified depth. Many of the attached addresses are not SNMP agents themselves so the search cannot continue through these addresses.

This above example shown in figure 6.1 shows a two level depth search starting from the “powerhub” SNMP agent. The search was confined to the first twenty nodes from each SNMP agent. This is because a complete two level depth search from the “powerhub” SNMP agent discovers hundreds and hundreds of nodes and requires more than 500MB of memory, which was more than available. This search took approximately 15 minutes including the time taken to gather the statics about the nodes. The nodes and addresses for SNMP agents are shown as red through to green depending on the number of interfaces the agents have. The blue nodes are hosts that did not respond to SNMP requests.

6.2 Usage and Ease of Use

The simple user interface that was developed allows the user to interact with the Computer Network Visualisation software. The user interface displays the graph in three dimensions using Java 3D as described above in section 3 above. An example three-dimensional graph is shown above in figure 6.1.

The user is also able to customise the information that is gathered about the computers in the network and how it is displayed. The user can interact with the graph in real time. The user can also use the interface to load and save the three-dimensional graph to graph files as described in section 7 below.

The user can interact with the graph in many ways by simply using the mouse. The user can left click on nodes of the three-dimensional graph to centre the view on the selected node. The user can left click on the and drag the mouse left, right, up and down to rotate the graph in the corresponding directions. The user can also right click and drag the mouse up and down to zoom in and out on the currently selected graph node. The user can right click on a node to display a popup menu containing the information that the user requires as shown in figure 6.2 below.

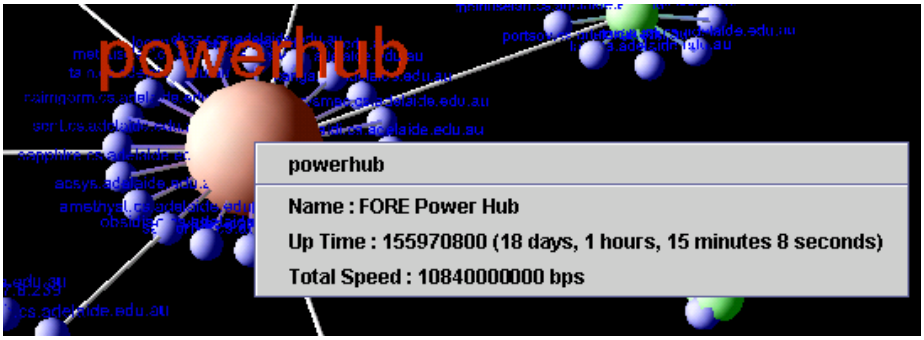


Figure 6.2 User Information

6.3 User Defined Information

The user can specify the information that they require and how it should be displayed by creating a settings file. This file is loaded when the Computer Network Visualisation software starts up. The software then gathers the information as it performs the depth first search and then stores the information for all the SNMP agents in the network.

The file contains three sections. The first defines the how the colour of the nodes are calculated. The following shows an example of this section of the settings file.

```
-- Node Color settings from green to red
COLOR_OBJECT_ID = "ifNumber"
COLOR_MAX_VALUE = 100
COLOR_MIN_VALUE = 0
COLOR_MAX =       255, 0, 0
COLOR_MIN =       0, 255, 0
COLOR_NON_SNMP =  0, 0, 255
```

The first line is a comment describing the colour settings. The second line specifies the object identifier on which the colour will be based. The next two lines `COLOR_MAX_VALUE` and `COLOR_MIN_VALUE` define the maximum and minimum values for specified SNMP identifier. The next lines define the colour used at the when the Managed Object identified above takes the maximum value. The next line `COLOR_MAX` defines the colour used when the Managed Object identified above takes the maximum value as a red, green and blue tuple. Similarly the `COLOR_MIN` defines the colour used when the Managed Object identified above takes the minimum value as a red, green and blue tupple. The final line `COLOR_NON_SNMP` defines the colour of a non-SNMP host.

The colour values are linearly interpolated using the maximum and minimum colours and values as follows:

$$Scaling = \frac{Get(COLOR\_OBJECT\_ID) - COLOR\_MIN\_VALUE}{COLOR\_MAX\_VALUE - COLOR\_MIN\_VALUE}$$

$$Color = Scaling \times (COLOR\_MAX - COLOR\_MIN) + COLOR\_MIN$$

Where:

Get(COLOR_OBJECT_ID) is the current value for the Managed Object identified by *COLOR_OBJECT_ID*.

Scaling is a scale between zero and one identifying where the current value is in the range *COLOR_MIN_VALUE* and *COLOR_MAX_VALUE*.

Color is the final colour of the node in the graph. This line uses three-vector math to calculate the colours.

The second section in the settings file defines the how the scale of the nodes are calculated. The following shows an example of this section of the settings file.

```
-- Node Scale settings
SCALE_OBJECT_ID = "ifNumber"
SCALE_MAX_VALUE = 100
SCALE_MIN_VALUE = 0
SCALE_MAX =       10
SCALE_MIN =       3
SCALE_NON_SNMP =  2
```

The meanings of the lines in this section are virtually identical to those in the first section except that the colours have been substituted for scale. The calculation of the scale for the nodes is also similar to the formula given above for the colours.

The third section in the settings file defines the information that the user wants to gather from the network. The following shows an example of this section of the settings file.

```
-- Statistics to gather
-- The name shown in the popup menu
-- and the associated Statistic / ObjectId
NODE_INFO
{
   "Name", "sysDescr";
   "Up Time", "sysUpTime";
   "Total Speed", SUM("ifSpeed");
}
```

This section defines a list of all the information the user requires marked by the `NODE_INFO` and enclosed by the braces. Each line in this list contains a string which is a user-friendly name for the Managed Object Identifier which follows. This information is gathered during the search of the network and is displayed in a popup menu as shown in figure 6.2 above.

The Managed Object Identifiers defined in each line of the `NODE_INFO` list and for `COLOR_OBJECT_ID` and `SCALE_OBJECT_ID` can also include the *SUM*, *COUNT*, *AVERAGE*, *MAX* and *MIN* statical commands which are defined in section 4.5 above. An example of this is shown above as `SUM("ifSpeed")`. This is also where mathematical expressions using Managed Objects would be written if the proposed extensions to SNMP described in section 4.7 were implemented.

7 Graph Files

7.1 Introduction

The graphing package implemented for this project also includes support for loading and saving the graphs to and from files. The graph package uses the Extended Mark-up Language (XML) to describe the information as text in the files. The XML file format is preferable to Java object serialisation. This is because the XML files are text files that are human readable and editable. Another advantage is that the XML files are independent of the Java object versions. The XML files will be compatible with different versions of the graph package where as Java object serialisation will not.

7.2 Extended Mark-up Language

The Extended Mark-up Language (XML) is a "meta" language that can be used to describe a board range of hierarchical structured data in a plain text editable file. The Extended Mark-up Language was defined by the World Wide Web Consortium (WC3) in February 1998 and is based on the Standard Generalized Mark-up Language (SGML). Extended Mark-up Language's primary purpose is to provide a cross platform cross program data format and is becoming a standard mechanism for exchange of structured data [XML].

The Extended Mark-up Language makes use of tags and attributes much like HTML. Tags are words bracketed by the "<" and ">" characters and attributes are strings of the form *name="value"* where *name* is the name of the attribute and *value* is the string value of the attribute. An element is classified as a tag and all of its attached attributes, for example <*element name="value"*> defines an element called *element* with one attached attribute *name* as described above. The element can then contain text or can recursively contain other elements much like HTML. The element is then ended by a tag with a slash and the tag name, for example <*/element*>. Unlike HTML an element can be ended with out any containing text or elements by ending the first tag with a slash, for example <*element name="value"/*>.

The Extended Mark-up Language unlike HTML does not define what the different tags and attributes mean, it only defines the structure of the document. The meaning and interpretation of the data is left to the application that uses it.

This project uses the Java API for XML Parsing *Release: 1.0.1* [JavaXML] to create and parse the XML files.

7.3 Graph File Format

The implemented graph file format defines some XML elements, tags and attributes specific to graphs and how they are composed. The following describes the element's, tags and attributes used:

Graph: This is the element that defines a graph. The elements only attribute is *id* which uniquely identifies the graph. The graph element contains other elements such as *Node*, *Edge*, *Position* and recursively other *Graph* elements. An example of the *Graph* element is as follows:

```
<graph id="foo"> ... </graph>
```

Node: This is the element that defines a node in a graph. The elements attributes are *id*, which uniquely identifies the node, and *scale*, which is a decimal number identifying the size of the node. The node element contains one other *Position* element defined below. An example of the *Node* element is as follows:

```
<node id="bar" scale="1.5"> ... </node>
```

Edge: This is the element that defines an edge in a graph. The elements attributes are *id*, which uniquely identifies the edge, *source* and *dest*, which identify the nodes or graphs at which edge starts and finishes. The edge element does not contain any other elements. An example of the *Edge* element is as follows:

```
<edge id="nay" source="foo" dest="baz"/>
```

Position: This is the element that defines a position in three-dimensional space. The elements attributes are *x*, *y* and *z* which are decimal number which identifies x, y and z coordinates in three-dimensional space. The position element does not contain any other elements. An example of the *Edge* element is as follows:

```
<position x="1.0" y="2.0" z="3.0"/>
```

This graph file format allows for graphs, nodes, edges and subgraphs. For simplicity the subgraphs and nodes of the graph are defined before the edges connecting them are defined.

7.4 Graph File Example

The following is an example of a graph file taken from the Computer Science Department at University of Adelaide's network. It shows how graphs can contain subgraphs, nodes and edges.

```
<graph id="ROOTGRAPH">
  <position x="0.0" y="0.0" z="0.0"/>
  <graph id="chook.cs.adelaide.edu.au/129.127.8.8">
    <position x="0.0" y="1.0" z="104.4"/>
  </graph>
  <graph id="powerhub/129.127.8.99">
    <position x="0.0" y="0.0" z="0.0"/>
    <node id="dizzy.cs.adelaide.edu.au/129.127.8.21"
        scale="2.0">
      <position x="0.0" y="0.0" z="31.2"/>
    </node>
    ... More nodes or subgraphs for powerhub subgraph
    <edge id="powerhub/129.127.8.99 to
         dizzy.cs.adelaide.edu.au/129.127.8.21"
         source="powerhub/129.127.8.99"
         dest="dizzy.cs.adelaide.edu.au/129.127.8.21"/>
    ... More edges for powerhub subgraph
  </graph>
  ... More nodes or subgraphs for ROOTGRAPH
</graph>
```

8 Future Work

While the goals of the project were implemented the performance of the system was disappointing. This could be overcome with more work on the following sections.

The first issue that could be overcome is the efficiency of Java3D. The speed of rendering the three-dimensional scene could be significantly improved by using "Immediate Mode" rendering. This is a lower level renderer which doesn't require a scene graph. This has the added benefit that the shape information doesn't have to be duplicated for each node and for each edge. The shape information had to be duplicated using a scene graph because each scene node could only have one parent.

The second performance issue that could be overcome is the speed of mapping the network. This could be improved by adopting the change to the Simple Network Mapping Protocol proposed in section 4.6. This change would add Walk as basic command like Get to reduce the number of requests and responses required to obtain all the information from a table in a SNMP agent. Another network mapping performance improvement could be achieved by creating multiple threads to map different parts of the network. This would allow some threads to continue working while some are blocked waiting for a response from the SNMP agents.

The last improvement is detailed in section 4.7 which proposes the implementation of an expression-based system to obtain more dynamic information from an SNMP agent based on already available information.

9 Conclusion

In conclusion this project has successfully implemented a generic three-dimensional graph visualisation package. This package was then used to visualise computer networks in three-dimensions. The projects successfully implemented different layout algorithms which suited the specific computer network application. Lastly project successfully implemented a network mapping and monitoring package which was used to gather information for visualisation.

The visualisation package could be used for many other applications, for example Petri Nets, Entity-Relationship Diagrams, Program Flow Graphs, Data Flow Diagrams, and PERT Diagrams.

This paper discusses the quality of different layout methods graphs based on the computer networks. The best quality layouts used hierarchical methods with which generally matches the layout of the physical networks. The layout methods are usually application dependant.

Lastly good three-dimensional visualisation requires fast rendering and an environment that is easy to interact with. This is required so that the user can understand the information that is been displayed and get a good feel for the third dimension. Advanced virtual reality systems such as a workbench or a CAVE could also be used to improve the users understanding of the environment.

Appendices

Appendix 1 Graph Package

Class Node

The Node class is used to represent and store information about the nodes in a graph.

Constructor Summary:

```
Node(Object Id, Object Info, String Label,
      javax.media.j3d.Shape3D Shape, javax.media.j3d.Material Color)
```
Constructs a new Node with Position=(0,0,0) and Scale=1.

```
Node(Object Id, Object Info, String Label,
      javax.media.j3d.Shape3D Shape, javax.media.j3d.Material Color,
      javax.vecmath.Vector3d Position, double Scale)
```
Constructs a new Node.

Method Summary:

```
Object getId()
```
Returns the Nodes Id.

```
Object getInfo()
```
Returns the Nodes Info.

```
String getLabel()
```
Returns the Nodes Label.

```
javax.media.j3d.Material getMaterial()
```
Gets the Material for this Node.

```
void getPosition(javax.vecmath.Vector3d Position)
```
Gets the Position for this Node.

```
void getPosition(javax.vecmath.Vector3f Position)
```
Gets the Position for this Node.

```
double getScale()
```
Gets the Scale for this Node.

```
javax.media.j3d.Group getSceneGraphRoot()
```
Gets the "scene graph root node" for this Node.

```
boolean getSelected()
```
Gets if the node is selected.

```
javax.media.j3d.Shape3D getShape()
```
Gets the Shape for this Node.

```
void setInfo(Object Info)
```
Sets the Nodes Info.

```
void setLabel(String Label)
```
Sets the Nodes Label.

```
void setMaterial(javax.media.j3d.Material Color)
```
Sets the Shape for this Node.

```
void setPosition(javax.vecmath.Vector3d Position)
```
Sets the Position for this Node.

```
void setPosition(javax.vecmath.Vector3f Position)
```
Sets the Position for this Node.

```
void setScale(double Scale)
```
Sets the Scale for this Node.

```
void setSelected(boolean Selected)
```
Sets if the node is selected.

```
void setShape(javax.media.j3d.Shape3D Shape)
```
Sets the Shape for this Node.

Class Graph Extends Edge

The Graph class represents the main graph data structure

Constructor Summary:

```
Graph(Object Id, Object Info, String Label,
      javax.media.j3d.Shape3D Shape, javax.media.j3d.Material Color)
```
Constructs a new empty Graph with Position=(0,0,0) and Scale=1

```
Graph(Object Id, Object Info, String Label,
      javax.media.j3d.Shape3D Shape, javax.media.j3d.Material Color,
      javax.vecmath.Vector3d Position, double Scale)
```
Constructs a new empty Graph.

Method Summary:

```
void addEdge(Edge e)
```
Adds an Edge to this Graph.

`void addNode(Node n)`
Adds a Node to this Graph.

`boolean containsEdge(Edge e)`
Returns true if this Graph conains the Edge.

`boolean containsEdge(Node Source, Node Dest)`
Returns true if this Graph conains the Edge.

`boolean containsEdgeById(Object Id)`
Returns true if this Graph conains the Edge with the Id.

`boolean containsNode(Node n)`
Returns true if this Graph conains the Node.

`boolean containsNodeById(Object Id)`
Returns true if this Graph conains the Node with the Id.

`int edgeCount()`
Returns the number of Edges in this Graph.

`java.util.Enumeration edgeIds()`
Returns an enumeration of the Edge Ids in this Graph.

`java.util.Enumeration edges()`
Returns an enumeration of the Edges in this Graph.

`java.util.Enumeration edgesFrom(Node n)`
Returns an enumeration of the Edges from the Node n in this Graph.

`java.util.Enumeration edgesTo(Node n)`
Returns an enumeration of the Edges to the Node n in this Graph.

`Edge getEdge(Node Source, Node Dest)`
Returns the specified Edge.

`Edge getEdge(Object Id)`
Returns the specified Edge.

`int getInDegree(Node n)`
Returns the number of Edges to Node n in this Graph.

`Layout getLayout()`
Gets the Layout for this Graph.

`Node getNode(Object Id)`
Returns the specified Node.

`int getOutDegree(Node n)`
Returns the number of Edges from Node n in this Graph.

`int nodeCount()`
Returns the number of Nodes in this Graph.

`java.util.Enumeration nodeIds()`
Returns an enumeration of the Node Ids in this Graph.

`java.util.Enumeration nodes()`
Returns an enumeration of the Nodes in this Graph.

`void removeEdge(Edge e)`
Removes the Edge from this Graph.

`void removeEdgeById(Object Id)`
Removes the Edge associated with the Id from this Graph.

`void removeNode(Node n)`
Removes the subgraph from this Graph.

`void removeNodeById(Object Id)`
Removes the Node associated with the Id from this Graph.

`void setLayout(Layout layout)`
Sets the Layout for this Graph.

`void setScale(double Scale)`
Sets the Scale for this Node.

Class Edge

The Edge class is used to represent and store information about the edges in a graph.

Constructor Summary:

```
Edge(Object Id, Node Source, Node Dest, Object Info,
      javax.media.j3d.Shape3D Shape, javax.media.j3d.Material Color)
```
Constructs a new Edge

Method Summary:

`Node getDestNode()`
Returns the Edges destination Node.

`Object getId()`
Returns the Edges Id.

`Object getInfo()`
Returns the Edges Info.

`javax.media.j3d.Group getSceneGraphRoot()`
Gets the "scene graph root node" for this Edge.

`javax.media.j3d.Shape3D getShape()`
Gets the Shape for this Edge.

`Node getSourceNode()`
Returns the Edges source Node.

`void setInfo(Object Info)`
Sets the Edges info.

`void setShape(javax.media.j3d.Shape3D Shape)`
Sets the Shape for this Edge.

Interface Layout

The Layout interface is implemented to define a layout manager for a graph

Method Summary:

`void EdgeAdded(Edge e)`
Notifies the layout that a edge has been added

`void EdgeRemoved(Edge e)`
Notifies the layout that a edge has been removed

`void GraphAdded(Graph g)`
Notifies the layout that a subgraph has been added

`void GraphRemoved(Graph g)`
Notifies the layout that a subgraph has been removed

`void InvalidateLayout()`
Instructs the layout to re-layout everything

`void NodeAdded(Node n)`
Notifies the layout that a node has been added

`void NodeRemoved(Node n)`
Notifies the layout that a node has been removed

`void setGraph(Graph g)`
Notifies the layout of the graph it is laying out

Class GraphException

GraphException raised by the graph package.

Constructor Summary:

GraphException()
Constructs a GraphException with no detail message.

GraphException(String s)
Constructs a GraphException with the specified detail message

References

[Hoa84] "Communicating Sequential Processes"
C. A. R. Hoare
Prentice-Hall
1984

[HMM98] "Graph Visualisation and Navigation in Information Visualisation"
I. Herman, G. Melançon, M. S. Marshall
Centre for Mathematical and Computer Sciences, Amsterdam, Netherlands
1998

[Rek93] "The Information Cube: Using Transparency in 3D Information Visualization"
J. Rekimoto, Sony Computer Science Laboratory, Inc., Japan
Proceedings of the Third Annual Workshop on Information Technologies and Systems, pp. 125-132
1993

[FSN] "File System Navigator"
Joel Tesler and Steve Strasnick.
Silicon Graphics Incorporated

[EW94] "Drawing Graphs in Two Layers"
P. Eades and S.H. Whitesides
Theoretical Computer Science, 131(2), pp. 361-374
1994

[KF96] "A Spring Modeling Algorithm to Position Nodes of an Undirected Graph in Three Dimensions"
Aruna Kumar and Richard H. Fowler
Department of Computer Science University of Texas
October 1996

[MNP] Matrix.net wall size poster of the Matrix and the Internet
Matrix Incorporated
http://www.mids.org/mapsale/poster/index.html

[CB00] "Internet Mapping Project"
Bill Cheswick Bell Labs
Hal Burch Computer Science Carnegie Mellon University
January 2000
http://www.cs.bell-labs.com/who/ches/map/index.html

[J3DAPI] "The Java 3D API Specification"
Henry Sowizral, Kevin Rushforth and Michael Deering
Sun Microsystems, Inc
Version 1.2
April 2000

[GSJ3D] "Getting Started with Java 3D"
Sun Microsystems, Inc
http://java.sun.com/products/java-media/3D/index.html
1999

[JLEX] "A Lexical Analyser Generator for Java"
Elliot Berk
Computer Science Department Princeton University
http://www.cs.princeton.edu/~appel/modern/java/JLex/
February 2000

[JCUP] "A Compiler Construction Tool for Java"
Scott E. Hudson
Georgia Institute of Technology
http://www.cc.gatech.edu/fce/domisilica/docs/maunual.html
July 1999

[RFC1155] "Management Information Base for Network Management of TCP/IP-based Internets"
Network Working Group: Request for Comments: 1155
M. Rose Performance Systems International
K. McCloghrie Hughes LAN Systems, Inc.
May 1990

[RFC1157] "The Simple Network Management Protocol"
Network Working Group: Request for Comments: 1157
J. Case
University of Tennessee at Knoxville
M. Fedor and M. Schoffstall
Performance Systems International
J. Davin
MIT Laboratory for Computer Science
May 1990

[RFC1213] "Network Working Group: Request for Comments: 1213"
K. McCloghrie
Hughes LAN Systems, Inc.
M. Rose
Performance Systems International
March 1991

[ASN] "Specification of Abstract Syntax Notation One (ASN.1)"
Information processing systems - Open Systems Interconnection
International Organization for Standardization, International Standard 8824
December 1987

[BER] "Specification of Basic Encoding Rules for Abstract Notation One (ASN.1)"
Information processing systems - Open Systems Interconnection,
International Organization for Standardization, International Standard 8825
December 1987

[JXML] "Java API for XML Parsing Release: 1.0.1"
Sun Microsystems, Inc
http://java.sun.com/xml/
2000

[JDK] "Java Development Kit"
Sun Microsystems, Inc
http://java.sun.com/j2se/1.3/
"Graph Layout Applet"
http://java.sun.com/applets/jdk/1.0/demo/GraphLayout/example1.html

[EHM98] "Finding the Best Viewpoints for Three-Dimensional Graph Drawings"
P. Eades, M. E. Houle, and R. Webber
Proceedings of Symposium on Graph Drawing
1998

[XML] "Extensible Mark-up Language"
World Wide Web Consortium
http://www.w3.org/XML
XML Specification
http://www.w3.org/TR/1998/REC-xml-19980210