ACKNOWLEDGEMENT

I would like to thank Dr. Peter Au for supervising my project. His help enabled me to transform his suggestions into implementations, and his criticisms into a complete project.

I would like to thank Dr. Moustafa M Ghanem for the initial proposal of this project. I would like to thank him for the long hours he spent trying to improve the functionality of the tool.

I would like to thank Mr. Christian Brenninkmeijer for the use of his Piping Tool. The tool enabled me to concentrate my efforts on the visual aspects and implementation of my project.
Clickstream is a generic term to describe visitors’ path through one or more Web sites. Clickstream data is derived from raw page requests and their associated information recorded in the Web server log files. Analysing the clickstreams can show how a Web site is navigated and used by the visitors.

The project idea originated from the weaknesses discovered in a previous MSc project and aims to resolve those problems identified. In addition, this project will propose and implement innovative additions for the visualisation and interpretation of the clickstream data.

The visualisation tool will represent the clickstream data in a circular representation, with a hierarchical representation depicting the directory structure of the Web server. In addition, this system provides facilities for zooming, filtering, color coding and dynamic querying.
# LIST OF FIGURES

2.1 WebVis WVVisualiser  
2.2 WebVis Example – Lack of Clarity  
2.3 Two-Dimensional Cone-Tree *(taken from Information Visualization)*  
2.4 Three-Dimensional Cone-Tree *(inXight)*  
2.5 H-Tree *(taken from Information Visualization)*  
2.6 Radial Tree *(NicheWorks)*  
2.7 Hyperbolic Tree *(inXight)*  
2.8 Tree Map *(taken from Information Visualization)*  
2.9 Representation of Network Graph Example  
3.1 Context Diagram for JVisC  
4.1 Proposed Representation  
4.2 JVisC Draft Desktop GUI  
4.3 Data Pre-processing  
4.4 Allocation – Directory Structure  
4.5 Allocation – Method 1  
4.6 Allocation – Method 2  
4.7 Adjacency Matrix used in JvisC  
4.8 Representing the Weights  
4.9 Representing the In and Out Traffic  
4.10 Adjacency Matrix when grouping files  
4.11 Uncluttering of Selected Node  
4.12 Sorting of the Nodes  
5.1 Flow Diagram for JViscProcessor  
5.2 Class Diagram for JVisc  
5.3 Class Diagram for JViscVisualiser  
5.4 Display the Directory Structure (Screenshot)  
5.5 Allocation of Nodes (Method 1)  
5.6 Allocation of Nodes (Method 2)  
5.7 Trace to Root and Children Nodes  
5.8 Displaying the Session Information (Screenshot)  
5.9 Highlighting the Paths  
5.10 Grouping of Nodes and Collapsing of Folders  
5.11 Single Circle View  
5.12 Concentric Circle View  
5.13 Inner Graph and Settings Box  
5.14 Single Selection Session Filter Box  
5.15 Multiple Selection Filter Box  
6.1 Case Study 1 (Example 1)  
6.2 Case Study 1 (Example 2)  
6.3 Case Study 1 (Example 3)  
6.4 Case Study 1 (Example 4)  
6.5 Case Study 2 (Example 1)  
6.6 Case Study 2 (Example 2)  
6.7 Case Study 2 (Example 3)  
6.8 Case Study 2 (Example 4)  
6.9 Case Study 2 (Example 5)  
6.10 Case Study 2 (Example 6)  
6.11 JVisC in Kensington (Screenshot)  
6.12 Performance Assessment (chart)  
7.1 JVisC in Sequence Analysis
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGEMENT:</td>
<td>i</td>
</tr>
<tr>
<td>ABSTRACT:</td>
<td>ii</td>
</tr>
<tr>
<td>TABLE OF FIGURES:</td>
<td>iii</td>
</tr>
<tr>
<td>CONTENTS:</td>
<td>iv</td>
</tr>
</tbody>
</table>

### CHAPTER 1 – INTRODUCTION:

1.1 OVERVIEW: 1
1.2 MOTIVATION: 1
1.3 PROJECT GOALS: 2
1.4 REPORT LAYOUT: 2

### CHAPTER 2 – BACKGROUND STUDY:

2.1 OVERVIEW: 3
2.2 WebVis: 3
2.3 Hierarchical Graphs: 5
2.4 Network Graphs: 8
2.5 Logfile Analysis: 10

### CHAPTER 3 – SPECIFICATION:

3.1 OVERVIEW: 11
3.2 SPECIFICATION:
   3.2.1 INTRODUCTION: 11
   3.2.2 BASIC SPECIFICATION: 11
   3.2.3 EXTENDED SPECIFICATION: 12
3.3 REQUIREMENTS ENGINEERING: 12
3.4 FUNCTIONAL REQUIREMENTS: 14

### CHAPTER 4 – CONCEPTUAL DESIGN:

4.1 OVERVIEW: 16
4.2 DESIGN DECISIONS:
   4.2.1 REPRESENTATION OF DIRECTORY STRUCTURE: 16
   4.2.2 REPRESENTATION OF SESSION INFORMATION: 17
4.3 DRAFT GRAPHIC USER INTERFACE (GUI) SKETCHES: 17
4.4 DESIGN COGS:
   4.4.1 DATA PRE-PROCESSING: 18
   4.4.2 REPRESENTATION OF DIRECTORY STRUCTURE: 19
   4.4.3 REPRESENTATION OF SESSION INFORMATION: 22
   4.4.4 REPRESENTING THE WEIGHTS: 22
   4.4.5 COLLAPSING OF FOLDERS AND GROUPING OF NODES: 23
   4.4.6 SESSION FILTERS: 24
   4.4.7 UNCLUTTERING: 26
   4.4.8 SORTING: 27
   4.4.9 JVisc Sequence Query Language: 27

### CHAPTER 5 – IMPLEMENTATION:

5.1 OVERVIEW: 31
5.2 MACHINE SPECIFICATION: 31
5.3 DEVELOPMENT TOOLS: 31
5.4 IMPLEMENTATION - JViscProcessor: 31
5.5 IMPLEMENTATION - JViscVisualiser: 32
   5.5.1 CLASS DIAGRAM: 33
   5.5.2 JViscVisualiser: 34
INTRODUCTION

This report details about the Individual Project, which is part of the requirements for the Bachelor of Engineering degree in Computing at the Department of Computing, Imperial College.

1.1 Overview

The overall project involves the design and implementation of a visualisation tool for clickstream data. Clickstreams are visitors' paths through a Web site. Clickstream data in a Web site is a collection of sessions in the site, which were derived from the raw file requests and their associated information from the Web server log files. In an e-commerce environment, clickstreams will provide information in understanding the effectiveness of marketing and merchandising efforts. In an ordinary Web site, clickstreams will provide information in finding out the type of pages that are well received and also the type of pages that are the 'killer' pages, i.e. where the visitor leaves. The owner of the Web site can then make decisions on how to change the site accordingly.

The project will represent the files in a hierarchical representation corresponding to the directory structure of the Web server. The source for the clickstream data will be from a web server’s access log file. The project will also aim to provide an interactive visual environment for the user, rather than the summary pages provided by current log analysis software available commercially.

As different users would have different concerns for the same data, the project will also allow users to set different levels of abstraction of the data to be viewed.

The remaining sections of this chapter will discuss the motivation for the project, the project goals and the structure for the rest of the report.

1.2 Motivation

The idea for this project originated from a MSc project “WebVis: Visual Network and Path Analysis” by Adrian R. Barrett in September 2000. The project detailed on different techniques for visualising the log file from a web server, and the visualisation is based on a circular display (as shown in Figure 2.1).

There were two main drawbacks in the final application for the MSc project, which are briefly discussed below:

- Lack of hierarchical structure
- Limited interactivity for the user

Lack of hierarchical structure

A hierarchical structure can either present the Web site layout, or the directory structure of the Web server. Though the first method would provide the hierarchical structure of the request of the links in the page, it
is not re-constructible from the log file. The second method of representing the directory structure will be easily comprehensible as most computer users are familiar with the file system of folders and files.

*Limited interactivity for the user*

An interactive visualisation tool has the advantage to cater for different cognitive patterns of different users. The application will then be able to cater for a larger target audience.

1.3 Project Goals

The main goal of the project is to provide a visualisation tool for analysing of log file data as a network structure with interactive manipulation of the view. From the initial log file, the tool will:

- Re-construct the directory structure of the Web server.
- Display the directory structure of the Web server.
- Display the session information from the log files.

The user of the tool should be able to group and ungroup different files to provide a collective view of the session information through the group of files.

The secondary goal is to allow the user to query the log files for specific sessions and to integrate the tool into the Kensington system to take advantage of the data mining features.

1.4 Report Layout

The rest of the report covers the study for the visualisation tool to be produced, its design, and evaluation of how the software meets the goals of this project:

**Chapter 2** discusses the background theory behind the project, highlighting related work and systems.

**Chapter 3** documents the design specification for the project.

**Chapter 4** describes the implementation of the software.

**Chapter 5** evaluates how well the system meets the requirements and goals of the project when tested with two case studies.

**Chapter 6** concludes the report, discussing the integration into the Kensington system, and potential future work.
This chapter presents the background study for the project. The theories and models used in the design of the application and related systems are also summarised within this chapter.

2.1 Overview

The background study done in relation to the development for the project can be categorised into four branches: WebVis – which analyses and highlights the pros and cons of the MSc project; Hierarchical Graphs – which investigates the different techniques for modelling hierarchical data; Representation of Network Graphs – will consider the methods that can be used to represent the transitions between nodes; Logfile Analysis – which studies the kind of information that can be obtained from the log files.

The remaining sections of this chapter will categorically examine the four branches mentioned above.

2.2 WebVis

"WebVis: Visual Network and Path Analysis" is a MSc project by Adrian R. Barrett.

WebVis consists of two main parts, WVFileConstructor and WVVisualiser. WVFileConstructor is a pre-processing application that can take raw text files and convert them into the .wvt format that the WVVisualiser uses. WVVisualiser (shown in Figure 2.1) is an application designed to visualise network flow and accompanying path related data.

The WVVisualiser was the beginning of the idea for the project, and the strength and weaknesses of the WVVisualiser have been investigated.
**Strengths**

**Single Circle Representation.**

The Single Circle Representation adopted by WebVis allows a comprehensible view of the session information between the individual nodes. The representation will also cater for a large number of nodes. However, there is little or no correlation between adjacent nodes in the representation.

**Weaknesses**

**Lack of hierarchical structure.**

A hierarchical structure can either present the website layout, or the directory structure of the Web server. The inclusion of hierarchy within the visualisation tool will allow users to relate the information with the storage location.

**Limited interactivity for the user.**

An interactive visualisation tool has the advantage to cater for different cognitive patterns of the different user. Several interactive options include displaying on paths with a minimal number, selecting different modes of representing the same information and performing dynamic grouping or collapsing of folders.

**Lack of clarity.**

WebVis allows the user to represent the in/out transitions between two nodes in a variety of ways. As the program tries to extend this representation throughout the representation, it failed to clearly represent the in/out traffic of individual nodes. For example, if red colour represents the in traffic and pick represents the out traffic. Viewing the node from either node /View or node /Search will give quite contradicting interpretations.

![Figure 2.2 WebVis Example – Lack of Clarity](image)

---

Figure 2.2 WebVis Example – Lack of Clarity
2.3 Hierarchical Graphs

This section looks at the various methods for representing hierarchical data and also looks at the advantages and disadvantages of each of the representation:

- Cone-Tree Layout

![Figure 2.3 Two-Dimensional Cone-Tree (taken from Information Visualization)](image)

The most traditional kind of representation for hierarchical data is the cone-tree. The representation positions the child nodes close to their common ancestor, either using top-down or the left-right approach with respect to Figures 2.3 and 2.4. The implementation of the representation can either by in two-dimension or three-dimension, depicted in Figures 2.3 and 2.4 accordingly.

![Figure 2.4 Three-Dimensional Cone-Tree (inXight)](image)

The advantages of the cone-tree representation include the speed of the algorithm and the ease of understanding the hierarchical information.

The disadvantages of the representation include the lack of scalability and unsuitability for web site data representation. The unsuitability for web site data representation is that although it can perform a map of the web site, links between pages are often difficult to represent.

- H-Tree Layout

The H-tree in Figure 2.5 is a variation of the simple hierarchical tree where the parent node has a line joining the first child node. The rest of the children nodes are placed along a line perpendicular to the first line.

The advantage of this representation is that related files can be found close to one another, i.e. the children of a parent are close together,
which will provide a useful representation for related page links from a page.

The disadvantages included difficulty to see the root of the tree and also the difficulty to comprehend the hierarchy structure. In addition, the approach is not particularly scalable.

• Radial Tree Layout

The representation shown in Figure 2.6 provides a radial view where the nodes are placed in concentric circles according to the depth in the tree. An example implementation is NicheWorks, which is a visualisation tool for the investigation of very large graphs.

The advantages for the radial representation include the suitability for viewing Website data as the space usage is much better than the others and allows for greater scalability. In addition, the representation will have less cluttering of nodes making the hierarchical structure easily understandable.

The disadvantage of the representation is the difficulty in the optimising the use of space, which lie in the algorithm. Besides considerations on the distribution of nodes to prevent overlapping, considerations for a fast algorithm may arise due to the interactive operations of navigation and zooming.
• Hyperbolic Layout

The hyperbolic tree representation shown in Figure 2.7. The initiative for this form of layout is the usage for graph visualisation, and view can be implemented in 2D or 3D. The example implementation above is from InXight Software, a spin-off company from Xerox.

The advantages of this representation include the provision of a bird’s-eye of the entire hierarchy. This technology is also able to translate dense thicket of Website data in easy-to-navigate visual maps.

The disadvantage is in the complexity of the mathematical operations required by the browser.

• Tree Map

The tree map representation shown in Figure 2.8 is a novel way of visualising hierarchically structured information, and demonstrated by an implementation in
TreeViz. The diagram shows the construction of a tree map from a cone-tree. The development of this method of visualisation was in response to the common problem of a shared hard disk, and to determine which users consume the largest share of disk space.

The advantage would be the ease of deciding which particular area of the Website has the most traffic, or the largest size.

The disadvantage is that large hierarchies are unsuitable for this representation as it only allows a small degree of nesting. An alternative shown by TreeViz is to have a control for the level of depth visible. It was found that experienced users could cope with depths to 7 levels, whereas inexperienced users could only manage up to 3 levels.

### 2.4 Representation of Network Graphs

While the earlier section dealt with the different methods for representing the directory structure obtained from the web servers log file. This section will look at the methods of representing that of the session information – mapping the transitions between nodes onto the graph.

A graph is represented as two sets – a set of nodes V and a set of edges E. Every edge is associated with two nodes in V. A graph is denoted as G(V, E). There are several ways to represent graphs. The most commonly used are the Adjacency Matrix and the Adjacency List. Incidence Matrix is another way to store a graph. Consider an example, with each of the edges having a weight of 1:

![Figure 2.9 Representation of Network Graph Example](image)

**Adjacency Matrix**

The *Adjacency Matrix* stores a graph in a square table with rows and columns corresponding to the nodes in the graph. The entry of the matrix is the weight of the edge connecting the nodes corresponding to the entry's row and column. If there is no edge, the weight is 0. For a graph with no weight function defined, assume that each edge has a weight of 1. For the example, the *Adjacency Matrix* for Figure 2.9 will follow on the next page.
As we can see, for a not directed graph, the matrix is symmetrical. Also, there is lots of zeros in the matrix, which is the consequence of the graph being sparse. A sparse graph is the graph for which the number of edges is much less than the number of entries in the adjacency matrix, basically $|E| << |V|^2$. Since most of the graphs are sparse, the Adjacency Matrix representation is not the most efficient.

**Adjacency List**

The Adjacency List representation stores a graph as a collection of linked lists. Each list starts at some node of the graph and contains all the nodes adjacent to this node. The adjacency list for the example follows:

- A → d
- B → j
- C → i
- D → a → h
- E → h → k
- F → g
- G → f
- H → d → e → k
- I → c → j
- J → b → i
- K → e → h

**Incidence Matrix**

Incidence Matrix representation is a table with rows corresponding to vertices and columns corresponding to the edges of the graph. The Incidence Matrix for the example:

\[
\begin{array}{cccccccccc}
  & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
a & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
b & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
c & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
d & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
e & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
f & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
h & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
i & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
j & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
k & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
\end{array}
\]
2.5 Logfile Analysis

When a person visits a website, there is a connection with a specific web server on the Internet that serves the files (HTML files, image files, etc) that the web browser requests. The web server creates a log of each of the files that were requested, when they were requested and other related information. The logs are separated into access logs, referrer logs, etc. The access-log file usually comes in the form of a text file containing the following information:

```
200.222.183.50 -- [19/Nov/2000:22:30:15 +0000] "GET /~ih/doc/lcd/pc_example/p1.html HTTP/1.0" 304 -
```


elite.doc.ic.ac.uk -- [19/Nov/2000:22:30:19 +0000] "GET /index.html HTTP/1.0" 200 14495

The above is the first 4 lines of an access-log file which is about 250,000 lines, equivalent to about a day’s worth of traffic.

Briefly explaining what details are within each line of the log file, taking the first line as an example. The first item 200.222.183.50 will refer to the host, which can either be a ISP number or the name. The next two -- refer to ident and auth user, which are usually ignored. The item [19/Nov/2000:22:30:15 +0000] is the timestamp, which gives the date and time the request was made. The next item "GET /~ih/doc/lcd/pc_example/p1.html HTTP/1.0" will refer to the type of request, the file requested and the HTTP form used. The next item 304 will refer to the status, where 200 means OK and the rest are usually errors. The last item will refer to the size of the file transmitted, which is - for the above example.

In raw form as shown above, the log files are just another text file and essentially useless. Logfile Analysis deals with the conversion of this raw log data into meaningful information about the web site traffic. Christian Brenninkmeijer’s Piping Tool was used to pre-process the raw log file into sessions, as well as cleaning up the data. Further details of how the Piping Tool was use will be documented in Chapter 5.
This chapter will document the design specification for the visualisation tool.

3.1 Overview

The specification for the visualisation tool will explore the specification and requirements engineering performed. The rest of the chapter will tackle the specification and requirements engineering of the project. The chapter will then conclude by looking at the functional requirements of the system.

3.2 Specification

3.2.1 Introduction

JVisC is a visualisation tool for clickstream data. This section aims to describe the basic and extended specification of the visualisation tool.

3.2.2 Basic Specification

Data

The JVisC shall obtain the required information from a pre-processed log file. The pre-processing will include the arrangement of the data into sessions. In addition, filtering of log file entries and removal of erroneous log file entries can also be performed.

The final data source for JVisC will include the list of entries with the 3 basic components:

- Session Identifier
- Host Identifier/Name
- File Name (including the path)

Representation

JVisC shall be able to display and manipulate the nodes - representing the individual files within the data. JVisc shall be able to re-construct the directory structure from the log files. JVisc shall allow operations like grouping of nodes and collapsing of the folder nodes to provide an aggregated representation.

JVisC shall maintain the hierarchical information possessed by the file, i.e. \docs\mytext.txt represents a file mytext.txt in a folder docs.

JVisc shall be able to display and manipulate the session information.
3.2.3 Extended Specification

Querying

JVisc shall implement a querying language for querying the session information and to represent the responses visually using the tool.

Kensington

JVIsC shall be integrated into the Kensington system as a visualiser, and possibly to make use of the data mining features available within the system.

3.3 Requirements Engineering

Requirements Engineering is the process of establishing a set of requirements for software development. The goals of which is to:

- Identify the external interfaces for the system
- Identify, categorise, and prioritise the system requirements

Due to the nature of this project, several steps have been omitted. The remainder of this section will perform the steps described in the Requirements Engineering phase Use Cases (pages 17-32)

Identify External Interfaces

The purpose of this step is to identify and document the set of external interfaces for the system: hardware, software and data store. The benefits of producing a context diagram is well documented, and thus shown below to depict JVIsC:

![Context Diagram for JVIsC](image)

Figure 3.1 Context Diagram for JVIsC

The context diagram of the project is depicted in the illustration in Figure 3.1. The system will read in the session data from the database and communicate with the user in two channels: from JVIsC, is the visual display to the user; to JVIsC, is the settings made by the user.

Excluded from the diagram are the separate processes of pre-processing and future integration. Pre-processing is concerned with the processing of the raw
log file is into sessions and encompassing certain filters on the data. The future integration with Kensington is also excluded from the diagram.

3.5 Functional Requirements

The project can be considered in three areas of operations:

1. Display and manipulation of the directory structure.
2. Display and manipulation of the session information.
3. Extended functionality for the tool.

1. Display and manipulation of the directory structure

### Use Case 1: Manipulation of directory structure

**Overview:**
This use case enables the Operator to perform operations on the directory structure that will alter the display, which is shown to the user.

**Preconditions:**
1. Only the display of the directory structure is active.

**Scenario:**

<table>
<thead>
<tr>
<th>Action</th>
<th>Software Reaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Operator clicks on a node.</td>
<td>The node is selected and the display updated, information of the selected node is also presented to the Operator.</td>
</tr>
<tr>
<td>2. Operator clicks on the zoom in/out button.</td>
<td>The display is updated with a view that is enlarged/reduced.</td>
</tr>
<tr>
<td>3. Operator clicks on the zoom in/out (with a node selected).</td>
<td>The display is updated with a view that is enlarged/reduced, the view is centred on the selected node</td>
</tr>
<tr>
<td>4. Operator right-clicks on a folder node.</td>
<td>A pop-up menu will allow the user to expand/collapse the folder, depending on the current state of the folder.</td>
</tr>
<tr>
<td>5. Operator selects collapse on a folder node (after the right-click (4)).</td>
<td>The display is updated with the children of the folder hidden, i.e. collapsed.</td>
</tr>
<tr>
<td>6. Operator selects expand on a folder node (after the right-click (4)).</td>
<td>The display is updated with the children of the folder shown, i.e. expanded.</td>
</tr>
</tbody>
</table>

**Scenario Notes:**
It is to be noted that the manipulation of this view is without the session information. The inclusion of the session information would require additional operations to be performed on the data.
2. Display and manipulation of the session information

Use Case 2: Manipulation of session information

Overview:
This use case enables the Operator to perform operations on the session information that will alter the display, which is shown to the user.

Preconditions:
1. The display of the directory structure MUST be active.

Scenario:

<table>
<thead>
<tr>
<th>Action</th>
<th>Software Reaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Operator chooses the .sess file</td>
<td>The file is read into the visualiser and the transitions between the individual files are visually represented to the user.</td>
</tr>
<tr>
<td>2. Operator sets a minimum limit on the value of the paths to be shown</td>
<td>The display is updated with the transitions between individual files, shown only when the number of transitions is greater than the minimum limit set.</td>
</tr>
</tbody>
</table>

Use Case 3: Manipulation of directory structure (with session information)

Overview:
This use case enables the Operator to perform operations on the directory structure that will alter the display. For the case when a folder is collapse, the aggregated value of the children will be shown to the user.

Preconditions:
1. The display of the directory structure is active.
2. The session information is also represented.

Scenario:

<table>
<thead>
<tr>
<th>Action</th>
<th>Software Reaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Operator clicks on a node.</td>
<td>The node is selected and the display updated, information of the selected node is also presented to the Operator. The number of transitions with other nodes is also represented visually.</td>
</tr>
<tr>
<td>2. Operator right-clicks on a folder node.</td>
<td>A pop-up menu will allow the user to expand/collapse the folder, depending on the current state of the folder.</td>
</tr>
<tr>
<td>3. Operator selects collapse on a folder node (after the right-click (4)).</td>
<td>The display is updated with the children of the folder hidden, i.e. collapse. The folder node will now hold the aggregated transitions with the other nodes.</td>
</tr>
<tr>
<td>4. Operator selects expand on a folder node (after the right-click (4)).</td>
<td>The display is updated with the children of the folder shown, i.e. expanded. The children node will now show their individual transitions with other nodes.</td>
</tr>
<tr>
<td>5. Operator groups a couple of nodes.</td>
<td>The display is updated with the creation of a new node (i.e. the group), the nodes grouped will be hidden. The new node will now show the aggregated transitions with other nodes.</td>
</tr>
<tr>
<td>6. Operator ungroups a group of nodes.</td>
<td>The display is updated with the deletion of the group node. The nodes represented by the original node will all be shown, and their individual transitions with other nodes represented.</td>
</tr>
</tbody>
</table>

Scenario Notes:
The additional functionality of grouping/ungrouping should be executable in Use Case 1, but have not been presented in that Use Case to reduce the amount of overlapping between different Use Cases.
3. **Extended functionality for the tool**

### Use Case 4: Session Filter

**Overview:**
This use case enables the Operator to set filters when reading in the session information.

**Preconditions:**
1. The display of the directory structure is active.
2. The file holding the session information has been defined.

**Scenario:**

<table>
<thead>
<tr>
<th>Action</th>
<th>Software Reaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Operator selects a node and clicks on the menu item.</td>
<td>A pop-up menu will allow the user to choose the options: Start – where the selected node must be the starting node of the session. End – where the selected node must be the ending node of the session. Any – where the selected node must be in the session.</td>
</tr>
<tr>
<td>2. Operator selects <strong>Start</strong>, after (1).</td>
<td>The representation will only show the sessions that have the selected node (from (1)) as the starting node.</td>
</tr>
<tr>
<td>3. Operator selects <strong>End</strong>, after (1).</td>
<td>The representation will only show the sessions that have the selected node (from (1)) as the ending node.</td>
</tr>
<tr>
<td>4. Operator selects <strong>Any</strong>, after (1).</td>
<td>The representation will only show the sessions that have the selected node (from (1)) as part of the session.</td>
</tr>
<tr>
<td>5. Operator selects multiple nodes and clicks on the menu item.</td>
<td>A pop-up menu will allow the user to choose the options: Random – where the selected nodes must be the session. Ordered – where the selected nodes must be the session, with the added restriction of the ordering of the selected nodes corresponding to the occurrence of the selected node within the session.</td>
</tr>
<tr>
<td>6. Operator selects <strong>Random</strong>, after (5).</td>
<td>The representation will only show the sessions that contain the selected nodes.</td>
</tr>
<tr>
<td>7. Operator selects <strong>Ordered</strong>, after (6).</td>
<td>The representation will only show the sessions that contain the selected nodes in the same order that the nodes were selected.</td>
</tr>
</tbody>
</table>

**Scenario Notes:**
For (5), the session filter will only consider nodes that are file nodes as setting a filter over the folder nodes are not yet considered for the current implementation.
This chapter will document the conceptual design for the visualisation tool.

4.1 Overview

The conceptual design for the visualisation tool will investigate the general design principles for the project. The chapter will begin with discussing the design decisions taken. The chapter will also include the draft GUI sketches and design cogs, which will look at the design of the individual components that arose during the implementation of the tool.

4.2 Design Decisions

Design decision will account for the decisions made during this stage that will influence the rest of the design process. This section describes the major decision made after the background study and the initial perception of the final implemented project.

4.2.1 Representation of Directory Structure

The radial tree that was shown in Figure 2.6 - Radial Tree (NicheWorks) will be the rough idea of how the directory structure will be represented. The adapted version that the visualisation tool will use is shown in Figure 4.1 (left), which includes a circular grid.

For example, looking at the prototype view of the directory structure from Figure 4.1 (left). The representation will correspond to that of a file system, with files and folders. A folder will be symbolised by a square node (which is blue in colour) and a file will be symbolised by a circle node (which is red in colour). The concentric circles (i.e. the circular grid) will then describe the depth of the individual file node or folder node from the root directory (symbolised by the R).

The choice of using the radial tree pattern can be attributed to a three factors. The first factor is the scalability of the representation, as this display can handle a lot more nodes than several other representations described in chapter 2. The second factor is that the whole...
representation can be easily represented within a single display. The third factor is the convertibility between the representation and the WebVis representation (shown in Figure 2.1).

4.2.2 Representation of Session Information

Based on WebVis, the session information will be drawn on top of the directory structure representation that is shown in Figure 4.1 (right). To represent the session information, the red line will correspond to the traffic to/from between the two nodes. A point to note is that the folder nodes at present will not have any traffic through it. If the folder node is collapsed, the folder node will represent the aggregated value of the children nodes.

Therefore under the two above methods of representation, we have the complement image of the final implementation. While being able to view the directory structure to able to decide the level of abstraction. The user can also view the session information to aid him in the decision.

4.3 Draft Graphic User Interface (GUI) Sketches

This section will provide a draft of the GUI for the system, as envisioned during the development of the scenarios in Section 3.4. A rough sketch of the GUI for the main visualiser is shown in Figure 4.2.

The development will make use of Swing components, which are part of the Java™ Foundation Classes (JFC). Other sketches of the user interface have been omitted as they can be easily implemented using the JDialog, JFileChooser, etc. Further details can be found from:

- The Java website at http://java.sun.com
- Java Swing Tutorial at http://java.sun.com/docs/books/tutorial/
4.4 Design Cogs

This section will look at the design of the individual cogs, which are the components that make up the system.

4.4.1 Data Pre-processing

The data taken in by the visualisation tool will require the two stages of pre-processing the data (as shown in Figure 4.3).

- Stage 1: Ordering the raw log file into sessions.
- Stage 2: Converting to a format suitable for the visualisation tool.

Stage 1: Ordering the raw log file into sessions

A raw log file consists of information that will not be required for the visualisation tool. When a file request is made to the Web server, an entry is placed into the log file which contains information of the host, the file requested, the timestamp, etc. The initial ordering of the entries will therefore be based on the time when the file request is made to the Web server.

For a Web server with many users, there is a need to order the log file entries into sessions. A session is the collection of entries for a particular user from when he enters the Web server till he leaves. This will be achieved using Christian Brenninkmeijer’s Piping Tool. At the minimal form, each line of the new file should include the host id of the user that made the request, the file that was requested and the session id to identify which session that the entry involves.

Figure 4.3 Data Pre-processing
Stage 2: Converting to a format suitable for the visualisation tool

The second stage is concerned with converting the file from the first stage to that which is understandable by the visualisation tool. The data will consist of the three files `.files`, `.sess` and `.users`.

- `.files` will consist of all the files from the session data. The purpose of this file is to create the initial representation of the web site’s directory structure.

- `.sess` will consist of all sessions from the session data. Each line in the file will represent a single session, and will include the session identifier and the paths between files that the user took. The path will have index of the files visited with reference to the file list in the `.files` file.

- `.users` will consist of all users from the session data. Each line in the file will represent a single user, and also all the session ids attributed to the user.

The separation of the session data to the three files has two advantages:

- Saving of space.
- Logical sequencing of the development phases.

 Saving of space

The saving of space is evident as the file and host name are only stored once. For example, an initial file of 11 MBytes will be reduced to 2,278 kBytes after stage 1. Stage 2 will then create the `.files` (538 kBytes), `.sess` (272 kBytes) and `.users` (140 kBytes). The three files approach will only require about 40% of the original file size.

 Logical sequencing of the development phases

The logical sequencing of development phases is that the directory structure was developed first, which requires the `.files` file. The representation of the session information will then require the `.sess` file. It is more effective to separate the files in the same manner as the data was separated.

4.4.2 Representation of Directory Structure

The directory structure representation will be re-constructed from the `.files` file described in section 4.4.1. From each line in the file, we can obtain a file node and possibly several folder nodes, i.e. `/man/text/file.html` will result in folder nodes `man` and `text` and a file node `file.html`.

Each of the nodes will be allocated a location on the concentric circles, and each will be given a level and an angle. The level represents the depth of the file or folder from the root node. The angle gives a unique position to each file node.
There are two methods of allocating the initial angles for the nodes: the first is based on the total number of child nodes that accessible from the node; the second is based on the number of child nodes at the immediate level.

**METHOD 1**

The basic algorithm for the allocation would be as follow, starting with the root node as the target node:

1. get the number of children (immediate) for the target node.
2. get a list which of the number of file nodes accessible from each child.
3. allocate an angle range for each child
4. FOR EACH CHILD:
   a. If file, then allocate angle for file node, i.e. midpoint of range.
   b. If folder, allocate angle for folder node, then repeat algorithm with node as target node.

An example with the directory structure, as shown in Figure 4.4 will be as follows:

Target Node: ROOT NODE (0°, 360°)
Number of children: 6
Level: 1

<table>
<thead>
<tr>
<th>File/Folder</th>
<th>Children</th>
<th>Angle Range (°)</th>
<th>Angle (°)</th>
<th>Folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>man</td>
<td>3</td>
<td>0, 90</td>
<td>45</td>
<td>✓</td>
</tr>
<tr>
<td>test0.txt</td>
<td>1</td>
<td>90, 120</td>
<td>105</td>
<td>✓</td>
</tr>
<tr>
<td>temp</td>
<td>1</td>
<td>120, 150</td>
<td>135</td>
<td>✓</td>
</tr>
<tr>
<td>src</td>
<td>2</td>
<td>150, 210</td>
<td>180</td>
<td>✓</td>
</tr>
<tr>
<td>doc</td>
<td>2</td>
<td>210, 270</td>
<td>240</td>
<td>✓</td>
</tr>
<tr>
<td>last</td>
<td>3</td>
<td>270, 360</td>
<td>315</td>
<td>✓</td>
</tr>
</tbody>
</table>

Target Node: man(0°, 90°)
Number of children: 3
Level: 2

<table>
<thead>
<tr>
<th>File/Folder</th>
<th>Children</th>
<th>Angle Range (°)</th>
<th>Angle (°)</th>
<th>Folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>file00.txt</td>
<td>1</td>
<td>0, 30</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>file01.txt</td>
<td>1</td>
<td>30, 60</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>file02.txt</td>
<td>1</td>
<td>60, 90</td>
<td>75</td>
<td></td>
</tr>
</tbody>
</table>

This process will continue until all nodes are allocated an angle.
METHOD 2

The basic algorithm for the allocation would be as follow, starting with the root node as the target node:

1. get number of children (immediate) for the target node.
2. allocate angle range for each child
3. FOR EACH CHILD:
   a. If file, then allocate angle for file node, i.e. midpoint of range.
   b. If folder, allocate angle for folder node, then repeat algorithm with node as target node.

An example with the directory structure, as shown in the Figure 3.5 will be as follows:

Target Node: ROOT NODE (0°, 360°)
Number of children: 6
Level: 1

<table>
<thead>
<tr>
<th>File/Folder</th>
<th>Angle Range (°)</th>
<th>Allocated Angle (°)</th>
<th>Folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>man</td>
<td>0, 60</td>
<td>30</td>
<td>✓</td>
</tr>
<tr>
<td>test0.txt</td>
<td>60, 120</td>
<td>90</td>
<td>✓</td>
</tr>
<tr>
<td>temp</td>
<td>120, 180</td>
<td>150</td>
<td>✓</td>
</tr>
<tr>
<td>src</td>
<td>180, 240</td>
<td>210</td>
<td>✓</td>
</tr>
<tr>
<td>doc</td>
<td>240, 300</td>
<td>270</td>
<td>✓</td>
</tr>
<tr>
<td>last</td>
<td>300, 360</td>
<td>330</td>
<td>✓</td>
</tr>
</tbody>
</table>

Target Node: man(0°, 60°) (3b)
Number of children: 3
Level: 2

<table>
<thead>
<tr>
<th>File/Folder</th>
<th>Angle Range (°)</th>
<th>Allocated Angle (°)</th>
<th>Folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>file00.txt</td>
<td>0, 20</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>file01.txt</td>
<td>20, 40</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>file02.txt</td>
<td>40, 60</td>
<td>150</td>
<td></td>
</tr>
</tbody>
</table>

This process will continue until all nodes are allocated an angle.

The advantage of using Method 1 is that there is a similar angle between two adjacent nodes for all the nodes in the representation. The advantage of employing Method 2 is that it allows the identification of mass, i.e. given four folders representing the different user directory, this method will allow identification of the users with the most number of files. An example implementation of the two different methods is shown in Figure 5.2.
4.4.3 Representation of Session Information

The adjacency matrix described in Section 2.4 will be used to store the session information when read in from the .sess file. Although the adjacency list method provides a much more effective way of storing the data, the adjacency matrix was used as it was more suitable if the grouping of nodes were to be performed interactively.

An example of the adjacency matrix that will be used in JVisc:

![Adjacency Matrix Diagram]

Figure 4.7 Adjacency Matrix used in JVisc

To demonstrate how an row from the .sess file will be calculated into the adjacency matrix, consider the follow line from the .sess file:

```
2 1 0 3 3 1 3 2 1 3 1 2
```

The first item is the session identifier, i.e. 2. The rest of the line would represent the path taken in the session, i.e. from START → 1(1), 1 → 0(3), 0 → 3(3), 3 → 3(4), 3→1(5), 1→3(6), 3→2(7), 2→1(8),1→3(9), 3→1(10),1→2(11), 2→END(12)

The adjacency matrix above is the result of reading the example session line.

The main disadvantage of using the adjacency matrix for storing the traffic between nodes is the use of memory that is of the order of $n^2$. A future improvement will be to convert the data structure to that of the adjacency list, which will require much less memory storage.

4.4.4 Representing the Weights

To represent the matrix, we should look at the figure below:

```
a = [0, 3] + [3, 0] = 1
b = [0, 1] + [1, 0] = 1
c = [1, 2] + [2, 1] = 2
d = [1, 3] + [3, 1] = 4
e = [2, 3] + [3, 2] = 1
```

Figure 4.8 Representing the Weights
We see that the figure will map out the values from the adjacency matrix discussed in Section 4.4.4 given above to the values on the right. Each line will represent both the traffic from and to the node. Consider the nodes m and n, the traffic from m to n is the value at [n, m] and the traffic from n to m is the value at [m, n].

Considering between using thickness and colour to distinguish between the different weights, the decision was to use colour. However, it is also important to differentiate the to/ from traffic and to represent the traffic within the node, i.e. [3, 3] = 1.

To differentiate the to/from traffic, it is felt that the general application of the methods used in WebVis failed to provide a consistent interpretation as highlighted in section 2.2. Rather than using the halves and bars methods, a highlighting method will be used.

Consider the Figure 4.9 where node 1 is selected, we can see that the pink highlight will represent the to traffic, i.e. the whole line b will represent the traffic from 1 → 0 and half of line c will represent the traffic from 1 → 2. The blue highlight will represent the from traffic, i.e. half of line c represent will represent the traffic from 2 → 1.

4.4.5 Collapsing of Folders and Grouping of Nodes

This section will deal with the collapsing of a folder or the grouping of several files to provide an aggregated value. The operation of grouping several files and collapsing a folder will require performing of mathematical operations on the adjacency matrix, which represents the traffic between nodes.

For the grouping of files, the affected nodes will be the files that have been chosen to group. For the collapsing of a folder, the affected nodes will be the descendent nodes of the folder, i.e. the files can be the child of the folder, the grand-child of the folder, the great-grand-child of the folder and so on.

Figure 4.10 will detail the mathematical operations that will be done to the rows and the columns of the affected file. Consider the grouping of nodes 0 and 1 with the creation of a new Group G, there are 3 operations from the initial adjacency matrix to obtain the adjacency matrix which includes the group. They are as follow:

(1) Sum the values of columns 0 and 1 for each row to a new column G.

(2) Sum the values of rows 0 and 1 for each column to a new row G.
(3) The third operation will exclude the columns and rows of 0 and 1 when plotting the session information. The sub-matrix, which is bounded in red, will be used to display the relevant information.

4.4.6 Session Filters

Occasionally, there is a need to view the sessions that only include a certain node or certain nodes. This can be achieved by including a filter that will only allow the sessions that fit the conditions to be read into the adjacency matrix when reading in the session information. The filter that will provide this functionality can be categorised into two different modes, namely single node selection and multiple nodes selection. The single node selection mode is when there is only one node selected. The multiple nodes selection mode is when there are more than one node selected.

**Single Node Selection Mode**

The single node selection mode can be split into the 3 sub-parts start, end and any. Start will only allow sessions where the selected node is
the start of the session. **End** will only allow sessions where the selected node is the end of the session. **Any** will only allow sessions where the selected node is part of the session.

<table>
<thead>
<tr>
<th>Session 1:</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session 2:</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>Session 3:</td>
<td>c</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Session 4:</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For example, taking the representation of the individual files as letters a, b, c and considering the list of sessions above as the `.sess` file. If node a was selected, selection of the different modes will only read in the sessions that meet the filter criteria.

- **Any** was selected, all the above 4 sessions will be included as each contains the file a.
- **Start** was selected, sessions 1 and 4 will be included as both of them have file a as the first file in the session.
- **End** was selected, only session 3 will be included as it is the only session that has file a as the last file in the session.

### Multiple Nodes Selection Mode

The *multiple nodes selection* mode can also be split into 2 sub-parts, which are the [random](#) and [ordered](#). Random will only allow sessions where the selected nodes are part of the session. **Ordered** will have the added restriction that the selected nodes occur in the order that they were click. For example, the selected nodes are in the order n₁ and n₂. The sessions that are allowed will have the path n₁ → ... → n₂ (where ... can be either a path or nothing).

For example, using the same sessions and files above. If nodes a and c were selected in order, the choice of the different modes will read in the sessions that meet the filter criteria.

- **Random** was selected, all 4 sessions will be included as all of them have the files a and c within the session.
- **Ordered** was selected, sessions 1, 2, 4 will be included as all of them have the files a and c occurring in order within the session. Session 3 will be included because file c occurs before file a.

For a user, when he either selects a single node or multiple nodes. He will be presented with the different choice of options for the filters as mentioned above. The visualisation tool will then read in the sessions with the filters and present the user with the representation of the sessions that satisfy the filter conditions.
Uncluttering is a practical operation that can be used when there are many nodes are clustered together. However, the technique that was used in WebVis would lead to some overlapping of nodes at the boundaries of change. The design for the new technique would push the nodes either side of the selected node away from the selected node, thus creating space in the proximity of the selected node. The method used involves compressing the 180° sector into a 175° sector (shown in Figure 3.10). Mathematically, this is achieved in two steps:

1. The first step involves setting the selected node at the arbitrary 0°, and then to calculate the angle between the other nodes and the selected node, shown in the pseudocode below.

   procedure unclutter(Node n) {
     for all other nodes (_n) excluding n{
       get the difference in angle with n
       i.e. _angle = _n.angle - n.angle;
       call uncluttering(_n, _angle);
     }
   }

2. The second step will involve all other nodes, and the difference in angle between the node and the selected node. The procedure will identify on which side of the selected node the node is and to perform the compression which first compressed the 180° into 175°, and then moving it away from the selected node (by either +5 or –5, dependent on which side the node is.) The last part of the procedure will set the node to the new angle. The pseudocode below shows the online of the procedure mentioned.

   procedure uncluttering(Node n, double angle) {
     if angle difference greater than 0 {
       newAngle = angle*((double) 175/180)+5;
     } else if angle difference is less than 0 {
       newAngle = angle*((double) 175/180)-5;
     }
     set new angle for n
   }
4.4.8 Sorting

Sorting will provide the user with a method of locating a file, with the prior knowledge that the nodes are in an ordered manner. Figure 4.12 shows the design of how the sorting method will be. The display will sort the nodes in the first level (left diagram), and for folders sort the children nodes of the folder (right diagram).

![Sorting in Level 1](image1)

Figure 4.12 Sorting of the Nodes

4.4.9 JVisC Sequence Query Language

The JVisC Sequence Query Language (SqQL) was only designed in the later parts of this project. The query language serves as an extension to the sessions filters discussed above.

The visualisation tool is used to visualise clickstreams or sessions, which consists of a sequence of file requests. Sequences are extremely common and are widely used, i.e. a document is a sequence of words or a word is a sequence of characters. Traditional query languages are often used for specific kinds of data. The rest of the section will aim to design a new query language SqQL, which will be applicable to all types of sequences.

<table>
<thead>
<tr>
<th>SqQL LANGUAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHOW</td>
</tr>
<tr>
<td>ALL</td>
</tr>
<tr>
<td>LENGTH &lt;= &gt;</td>
</tr>
<tr>
<td>WHERE</td>
</tr>
<tr>
<td>START</td>
</tr>
<tr>
<td>END</td>
</tr>
<tr>
<td>ANY</td>
</tr>
<tr>
<td>RANDOM</td>
</tr>
<tr>
<td>ORDER</td>
</tr>
<tr>
<td>SEQUENCE</td>
</tr>
<tr>
<td>PROJECT</td>
</tr>
<tr>
<td>FROM</td>
</tr>
<tr>
<td>TO</td>
</tr>
<tr>
<td>OPERATORS</td>
</tr>
<tr>
<td>AND</td>
</tr>
<tr>
<td>OR</td>
</tr>
</tbody>
</table>
Consider the three sequences:

A:  1  2  3  1  3
B:  2  1  2
C:  3  2  3  3  1

The following are a list of possible queries that would include the query in natural language and using SqQL, and the sequences that match the query.

(Q1) Show all the sequences.

SHOW ALL

A [1,2,3,1,3]  B [2,1,2]  C [3,2,3,3,1]

(Q2) Show all the sequences of length less than 5.

SHOW LENGTH<5

B [2,1,2]

(Q3) Show all the sequences of length equal to 5.

SHOW LENGTH=5

A [1,2,3,1,3]  C [3,2,3,3,1]

Simply, SHOW ALL can also be represented by the query SHOW LENGTH>0.

For a simple query involving a single node, we have the following:

(Q4) Show the sequences that start with node 1.

SHOW ALL WHERE START=1

A [1,2,3,1,3]

(Q5) Show the sequences that end with node 3.

SHOW ALL WHERE END=3

A [1,2,3,1,3]

(Q6) Show the sequences that consist of node 1.

SHOW ALL WHERE ANY=1

A [1,2,3,1,3]  B [2,1,2]  C [3,2,3,3,1]

In addition, we can make use of the operators to enforce a double constraint on the sequences that match the query.

(Q7) Show the sequences that start with node 2 and end with node 2.

SHOW ALL WHERE START=2 AND END=2

B [2,1,2]
(Q8)  Show the sequences that start with node 3 or end with node 3.

    SHOW ALL WHERE START=3 OR END=3

    A [1,2,3,1,3]   C [3,2,3,3,1]

Occasionally, we would only require that part of the sequences beginning from or terminating from a single node be displayed. The following queries are examples.

(Q9)  Show all sequences that contain node 2 from node 2.

    SHOW ALL WHERE ANY=2 PROJECT FROM 2


(Q10) Show all sequences that contain node 1 to node 1.

    SHOW ALL WHERE ANY=1 PROJECT TO 1


For (Q10), the PROJECT TO will show the sequence to the last occurrence of the specified node. The purpose of the next query is to check for cycles, i.e. 1 → 2 → … → 1.

(Q11) Show all sequences that contain nodes 1 from node 1 to node 1.

    SHOW ALL WHERE ANY=1 PROJECT FROM 1 TO 1


The following queries will involve multiple nodes.

(Q12) Show all sequences that start with node 1 and end with node 3.

    SHOW ALL WHERE START=1 AND END=3

    A [1,2,3,1,3]

(Q13) Show all sequences that start with node 1 or start with node 2.

    SHOW ALL WHERE START=1 OR START=2

    A [1,2,3,1,3]   B [2,1,2]

(Q14) Show all sequences that contain node 1 and node 2.

    SHOW ALL WHERE RANDOM=[1,2]

    A [1,2,3,1,3]   B [2,1,2]   C [3,2,3,3,1]
(Q15) Show all sequences that contain node 1 and node 2 (in order).

SHOW ALL WHERE ORDER=[1,2]

A [1,2,3,1,3]  B [2,1,2]

(Q16) Show all sequences that contain node 2 and node 1 (in sequence).

SHOW ALL WHERE SEQUENCE=[2,1]

B [2,1,2]

RANDOM will allow sequences that include the nodes specified. ORDER will allow sequences that include the nodes that are in order, i.e. \ldots \rightarrow n_1 \rightarrow \ldots \rightarrow n_2 \rightarrow \ldots$. SEQUENCE will only allow the sequences that have the nodes in sequence, i.e. $\ldots \rightarrow n_1 \rightarrow n_2 \rightarrow \ldots$

(Q17) Show all sequences that contain nodes 3 and 1 from nodes 3 to 3.

SHOW ALL WHERE ORDER=[3,1] PROJECT FROM 3 TO 1


The design of this query language was the result of realising that most of the time the user will only be interested on a small subset of the sessions. Therefore, it would be useful to provide the SqQL or a part of the language to partition the session information.
This chapter will document the implementation of the visualisation tool.

5.1 Overview

After designing the major areas of the visualisation tool, the next step will involve the implementation of the tool. Areas that this chapter will cover include the specification of the machines and the development tools that were used for the implementation. The rest of the tool will look at the implementation of the major fragments of the visualisation tool.

5.2 Machine Specification

The machines that were used for the development were the computers in the Imperial College Computing Labs and the home computer. The following will be considered as the standard specification:

- Processor: Intel Pentium II 333 MHz
- Memory: 64 MB RAM
- Display: 800 × 600 pixels
- Input Device: Mouse and Keyboard

5.3 Development Tools

Java Development Kit v 1.3

The Java™ 2 Platform, Standard Edition (J2SE™) has revolutionized computing with the introduction of a stable, secure and feature-complete development and deployment environment designed from the ground up for the Web. It provides cross-platform compatibility, safe network delivery, and smartcard to supercomputer scalability. It provides software developers with a platform for rapid application development, making it possible to deliver products to market in Internet time. It defies traditional software development and deployment models by delivering on the promise of cross-platform compatibility.

(reproduced from http://java.sun.com)

KAWA v 4.10a

The Kawa Java IDE was used for the development of the visualisation tool.

5.4 Implementation – JViscProcessor

As mentioned earlier in Chapter 2, Christian Brenninkmeijer’s Piping Tool was used in stage 1 of the pre-processing, which involved converting the raw log file into a file with the same information with the inclusion of the session identifier. To concentrate more on the visualisation aspects of representing the session paths, details that are considered are only the session identifier, host and file requested. A copy of the UML file that was used with the Piping Tool to obtain
the data files that the JViscProcessor can take located through the page with the URL: http://www.doc.ic.ac.uk/~keja98/index.html.

After obtaining the file from the Piping Tool, JViscProcessor is the pre-processing application that will take in the file and convert them to the three files .files, .sess and .users, which is the stage 2 as described in Chapter 4.4.

JViscProcessor main task is to convert the processed log file into the format that JViscVisualiser will take. The application can either run as a stand-alone application or within the JVisC setup. JViscProcessor will be to obtain the input file, obtain the user settings and obtain the output filename from the user. The application will then construct the output files. A flow diagram for JViscProcessor is Figure 5.1.

The detailed screen shots for JViscProcessor can be found in User Guide (Appendix A, pages A.2-A.3). The user guide will also explain to the user how to convert a file into the three files which can be used by JViscVisualiser.

5.5 Implementation – JViscVisualiser

JViscVisualiser is the visualisation application that will take in the files created from JViscProcessor, and create a graphic representation of the information. JViscVisualiser is able to take in .files file generated by JViscProcessor and construct the directory structure representation. It is also able to take in the corresponding .sess file and represent the session information relating to the files from the .files file.
5.5.1 Class Diagram

The class diagrams for JVisc and JViscVisualiser can be seen in Figure 5.2 and 5.3.

There are additional classes like ScrollablePicture (used for Snapshot implementation) and HtmlFrame (used for help file and as a browser). Further details of the classes can be found in Appendix C.
5.5.2 JViscVisualiser

JViscVisualiser provides any interactive and user-friendly environment to view the information present within the web server log files. It allows an aggregate view of nodes by both grouping and collapsing of folders.

JViscVisualiser operates in two distinct modes: displaying the directory structure and displaying the session information.

Displaying the Directory Structure

From the above diagram, we can see the first mode of operation of JViscVisualiser: displaying the directory structure in action. Constructed from a .files file, each file node, or entry in the .files file is represented by a red circle. The blue squares are the folders, and are deduced from the filename. An additional node is the green and black squares, which represent groups. Grouping will be further demonstrated later in this section.

Panel Operations

From the initial screen, the user can have several interactions with JViscVisualiser from the circular display panel on the left. He can:

- **Left-Click** on a node to select the node. The corresponding information of the node will be displayed on the information panel on the right.
- **Ctrl-Left-Click** on several nodes to select them. The information panel will display the names of the nodes selected.
- **Click-and-Drag** to form a selection box, which will select several nodes together.
• *Click-and-Drag* on a node or a selection box to move the node/s.

• *Right-Click on a file node* will allow the user to hide the node, unclutter the node and to snap the node to the grid.

• *Right-Click on a folder node* will allow the user to hide the node, unclutter the node, snap the node to grid and to expand/collapse the folder nodes.

• *Right-Click on multiple selected nodes* to hide the nodes and to group the nodes.

• *Right-Click on a selection box* to group the nodes.

**Tool Bar Operations**

From the tool bar to the right of the circular display panel, there are several other interactions with JViscVisualiser. The user can:

• *Click* on the buttons on the left to zoom in/out. If a node is selected, the zoom will be centred on the selected node. If not, the zoom will be pegged on the left-top corner of the display. Alternatively, the user can make use of the zoom slider above the information panel.

• *Click* on the buttons on the left to toggle between the *Single Circle View* and the *Concentric Circles View*. Both the views will be explained later in the section.

• *Click* on the button on the left to search the nodes. The search will highlight the nodes that match the search string.

• *Click* on the button on the left to change the level settings. There are two modes of operations: The first allows the user to toggle each level on/off individually. The second allows the user to set the value as the abstract level, such that all nodes beyond specified level will be collapsed.

• *Click* on the button on the left to view a menu showing the customisable features of the visualiser. The customisable features include changing the allocation method (as shown in Figure 5.5 and 5.6).

Figure 5.5 Allocation of Nodes (Method 1)
The user can also set the trace on, as shown in Figure 5.7. This operation will highlight the selected node and show the path to the ancestors (in a dark dashed line) and also the paths to the descendent nodes (in a grey line). Additional features include changing the font size and colour, changing the node size and colour and changing the size of the circular grids.

- **Click** on the button on the left and it allows the user to take a snap shot of the current view. With this snap shot, the user can either compare it with other snap shots and to make different decisions about the information present.

- **Click** on the button on the left and the nodes in the visualiser will be sort in alphabetical order from the 3 o’clock in an anti-clockwise direction.

- **Click** on the button when a file node is selected and the file will be shown using a HTML Viewer.
Displaying the Session Information

The user can then view the session information concerning the files displayed earlier, i.e. from the `.sess` file. The screenshot that depicts how the visualiser will appear to the user is shown on Figure 5.8.

In addition to the original display depicting the directory structure of the Web server, there is now the red line that depicts the traffic between the two connecting nodes $n_1$ and $n_2$. What this red line symbolises is that there is a session path where there is a page request for $n_2$ after a page request for $n_1$, or vice versa.

Panel Operations

From the display panel on the left, the user can then:

- **Left-Click** on a node to select the node. In addition to the information of the node being displayed on the information panel, the paths with other nodes will also be highlighted.
From Figure 5.9, we can see that all the traffic between `test1.txt` and `/src/file10.txt` will be `test1.txt → /src/file10.txt`. The traffic between `/src/file10.txt` and `file11.txt` will be `/src/file10.txt → file11.txt`. And lastly, we can see that the traffic between `/src/file10.txt` and `file02.txt` will be a mixture between `/src/file10.txt → file11.txt` and `file11.txt → /src/file10.txt`.

- **Grouping** the file nodes `file11.txt`, `file20.txt` and `file21.txt`. We can view the aggregated traffic as a single group node, as illustrated in Figure 4.9. Similarly collapsing the folder `doc` that is the parent of the files `file20.txt` and `file21.txt`, we can view the aggregated traffic as a single folder node.

![Figure 5.10 Grouping of Nodes and Collapsing of Folders](image)

**Tool Bar Operations**

From the tool bar to the right of the circular display panel, there are several other interactions with JViscVisualiser. The user can:

- **Click** on the buttons on the left to toggle between the **Single Circle View** and the **Concentric Circles View**. The **Single Circle View** (shown in Figure 5.11) will represent the file nodes on a single circle. If a folder is collapsed, the folder will be shown instead of the file. The **Concentric Circle View** (shown in Figure 5.12) represents the file nodes in a concentric circle, which represents the depth of the file/folder from the root.

![Figure 5.11 Single Circle View](image)
With the **Single Circle View**, the user can then view the traffic between the file nodes and/or folder nodes that have been collapsed. The **Single Circle View** will allow the user to easily comprehend the traffic between nodes without the possibility of obscuring nodes. However, the user will then have reduced ability to collapse the folder nodes.

With the **Concentric Circles View**, the user can then view the traffic between the nodes and collapse and expand folder nodes when he deems necessary. With an increase in the number of nodes, this view will produce lines that may be obscured by other nodes or their labels.

Both display work jointly to provide the user with the functionality of grouping and collapsing/expanding folders nodes, and also to have a clear and unobstructed view of the nodes and their paths.

- **Click** on the buttons on the left to perform the following operations to the display of the session information:
  - Toggle on/off the path trace from a selected node. This function will result in the differentiation of the traffic to the other nodes, which is shown in Figure 5.9.
  - Set a limit on the minimum amount of traffic between nodes before the line joining the two nodes are displayed. Only the lines that have traffic more than the setting will be displayed.
  - Perform session filters on the session information to view the session with various characteristics. (The session filter function will be further elaborated later)
5.6 Implementation – Additional Features

5.6.1 Inner Graph

In addition to the above, there is an additional display that allows the user to visualise the nodes that are the starting or the ending nodes. Being the start node refers to the node being the first node in the session. And the end node would refer to the node being the last. Termed the *Inner Graph*, the graph will allow the user to know which node/ collection of nodes that are often the starting or the ending nodes in a session. From Figure 5.13, the Inner Graph method can show that icon2.jpg is the most frequent start node for the sessions.

![Figure 5.13 Inner Graph and Settings Box](image)

The settings box on the left will allow the user to select whether the Inner Graph is displayed, and also vary between showing the starting or the ending nodes.

5.6.2 JVisC Sequence Query Language (SqQL)

Initially design as a session filter to be able to partition the vast amount of different sessions. The implementation identified addition ideas that led to the design of JVisC SqQL in Section 4.4.9. Referring back to the example queries given in Section 4.4.9, we can see that the following have been implemented.

(Q1) is simply showing all the sessions from the .sess file.

(Q4), (Q5), (Q6), (Q9), (Q10) can be executed by selecting a single node and clicking the button , followed by Session Data Menu. The following box will allow you to specify the details of the query.

![Figure 5.14 Single Selection Session Filter Box](image)
For example the query `SHOW ALL WHERE ANY=1 FROM 1` can be performed by clicking on the node (corresponding to 1) clicking on `Any` and `From` in the dialog box as shown in Figure 5.14.

(Q14), (Q15), (Q16) can be executed by selecting a several nodes and clicking the button , followed by Session Data. The following box will allow you to specify the details of the query.

![Multiple Selection Filter Box](image)

For example the query `SHOW ALL WHERE ORDER=[1,2]` can be performed by clicking on the nodes (corresponding to 1, then to 2) and then clicking on `Ordered` from the dialog box as shown in Figure 4.9.

### 5.6.3 Snapshot

When a snapshot is taken, the current representation is stored and it will allow the user to view it again later. There is no limit to the number of snapshots that can be taken, the drawback is that each snapshot will take up system resources and may affect the performance.

An application for the use if there were two separate `.sess` file for the same `.files` file which represents the male and the female sessions which were pre-processed. The user can take a snapshot of the first file and then load the second to compare the differences in the patterns.

### 5.6.4 Saving and Loading

The implementation includes an option for saving and loading. The user can save the structure of the directory structure. He can then load it in another time to query the `.sess` file.

Alternatively, the save file can be used in pre-processing the log files in the future to the maximum level of abstraction that is required.

### 5.6.5 Exporting

James Franklin is doing a roughly similar project called “Interactive Visualisation of Website Clickstreams”. At current, the communication between our implementation is by file basis. There are two files that he can take in from my application, the first consists of the groupings of the files. The second is the directory structure, which would represent the level of abstraction to be viewed in his visualisation tool.
6.1 Overview

The visualisation tool was developed to visualise the session information from log files. In order to evaluate the visualisation tool, two case studies will be put to use and the functionality investigated.

The first case study will discuss how the tool functions when the data comes from Department of Computing log file. The second case study will use the tool for a smaller log file. The last section will assess the functionality of the tool.

The rest of the chapter will look at the integration with Kensington and also the performance of the tool.

6.2 Case Study I

The Department of Computing Web server consists of a large number of files that are accessible to users from the Internet. As it would require much memory storage for processing log files with thousands of nodes, further pre-processing was done to an initial file with the number of files equalling 400. The directory structure of the server involving the files from the log file will then be re-created by the tool (left diagram of Figure 6.1). The next step was to read in the corresponding .sess file which contained 436 sessions, the tool then was set to the Single Circle View (right diagram for Figure 6.1).

![Figure 6.1 Case Study 1 (Example 1)](image)

Clearly having this massive amount of red lines and the large number of nodes will not make any sense. To counter this problem, we collapse the tree by
setting the abstract level to 1. At this level, what seen is only the user directories that the files come from.

From Figure 6.2, it can be seen that there is a lot of traffic flow between the pages in \textit{lab} with the pages in \texttt{~svb}, \texttt{rr2000}, \texttt{~ids} and \texttt{~gzy}. However, Figure 6.2 fails to show the pages that are the starting and ending pages, i.e. the pages that the visitors come to and go from.

This problem of starting and ending nodes is solved via the Inner Graph, which can shows that most people coming in will tend to start at \texttt{~ih}. Further expanding of the folders to the file (shown in Figure 6.3) will show that \texttt{/~ih/doc/stepper/} is the file that most visitors tend to start the visit. On changing the settings for the Inner Graph to that for end nodes, \texttt{/~ih/doc/stepper/} is also the file that most visitors tend to leave from.

The user can click on individual folders to answer questions like “\textit{I know that the folder lab has a lot of traffic, can I go further into the folder to find out more?}”. This simply question can be answered by clicking on the folder and expanding it (shown in Figure 6.4).
This case study shows the basic functionality of the visualisation tool. For a large Web site, it will be difficult to visualise all the nodes at the same time. With the ability to collapse and expand the folders containing the files, the simpler alternate view provides an easier platform to understand the log file. There is also the problem of performance of the tool that is determined by the number of file and folder nodes. Section 6.5 will evaluate the performance of the visualisation tool.

6.3 Case Study II

This case study will look at pre-processed log file taken from Christian Brenninkmeijer’s Piping Tool. Pre-processing this data to a suitable form for JVisC also gave the details of the file – which consisted of 20 different files, 23 different users and 1982 different sessions. The log file belonged to the www.inforsense.com website.
Reading the `.files` file as the initial input, the directory structure of the server holding the files can be established (shown in Figure 6.5). Reading in the session information from the `.sess` file, the resulting diagram in shown in Figure 6.6.

At this stage, it can be seen that there is a lot of traffic between the three `index.html` files in the top-right quadrant of the display, which answers the question of “Which area of the Web site do most visitors visit?”.

![Figure 6.6 Case Study 2 (Example 2)](image)

To simplify the view of the data, the level of abstraction was set to 1. And the same question “Which area of the Web site do most visitors visit?” asked.

![Figure 6.7 Case Study 2 (Example 3)](image)

From Figure 6.7, we can then answer there is a lot of traffic between `index.html` and the folder `solutions` and also a substantial amount of traffic between the folders `solutions` and `organisation`, and `solution` and `company`. In addition, by toggling the settings for Inner Graph (described in Chapter 5.6.1) the following questions “Where do most people enter the site?” and “Where do most people leave the site?” can be answered.
The left diagram (in Figure 6.7) shows that most people will enter the site from the index.html page. The right diagram shows that most people leave the site from index.html, with a significant amount of people leaving after visiting the files in the folder solutions.

Clicking on index.html, additional information can also be obtained. From the left diagram (in Figure 6.8), we can see the highlighted paths between index.html with the rest of the nodes. It can also be perceived that most traffic will originate from index.html to the other nodes, except for the folder form where the traffic moves towards index.html. The table on the right reinforces the point by shown that from index.html to form the traffic is 0, and from form to index.html, the traffic is 3. An additional information obtainable from the table is that for sessions where there is clicks from index.html to index.html, the total count of such clicks correspond to 1776 as seen on the table.

To make use of the session filters that the tool possible by consider the scenario that the people from the Web site would like to find out who downloads the kensigton.zip file. The left diagram in Figure 6.9 shows the sessions that include the kensington.zip file. Alternatively, the representation can also show the part of the sessions from the kensington.zip file or part of the sessions to the kensington.zip file.
To demonstrate the grouping functionality of the tool, we group the three *index.html* into a Group INDEX. Displaying the paths between nodes where the traffic is greater than 100, we get the figure as shown in Figure 6.10.

By clicking on the Group INDEX, we can get the ratio of to/from traffic between the file nodes *kensington.zip*, *eScience.html*, *sc99.html* and *inforsense.html* and the three *index.html* in the Group INDEX. Other than sc99.html, most traffic will originate from the Group INDEX to the other files (*kensington.zip*, *eScience.html* and *inforsense.html*).

This case study has demonstrated how the tool can be used with the log file from a Web server with a low number of nodes. The tool performs well to the user changes of the structure and the querying of specific sessions.

By finding out the files that have high traffic, and the files that have low retention (i.e. the visitors leave). The people from the Web site can then make changes to improve the files with low retention by understanding why the files of high traffic have the high amount of traffic. After the changes, the Web site will probably attract much more visitors and provide a useful service for them.

### 6.4 Integration with Kensington

The integration with Kensington required several changes to be made to the source code of the visualisation tool, which was initially to be run as a stand-alone application. The rest of this section will describe the modification performed to the code.

**Input Source**

Used as a visualiser on the client-end of the Kensington system. The visualiser will make use of the Kensington tables held on the user’s workspace. A single table with two columns was the preferred choice of input, as this will then provide a common data source similar to that of the other project “Interactive
The information that the table will hold will include the file requested and the session identifier.

For the input source, the original design took in the two files .files and .sess separately, and would re-read the .sess file when a session filter was performed. The .sess file will have each line representing a session and would be able to perform the filtering quite easily. Under the Kensington system, there would be two options to counter this problem: to re-read in the table each time, or to store the sessions in memory. The final implemented version took the second option of storing the sessions in memory.

**Menu Operations**

Unlike the application version, the Kensington version will not require the pre-processing as a menu operation as the pre-processing is done after reading in the data from the Kensington table. Therefore, these operations have been hidden in the Kensington version. The GUI for the Kensington version can be seen in the figure below.

![Read Table View Directory View Sessions](image)

From the main toolbar (shown in Figure 6.11), the user can then make use of the visualisation tool. The first button from the left, Read Table will read in the data from the Kensington table. The next button View Directory, will then construct and display the directory structure from the information read in. The next button View Sessions, will then present the session paths between the file nodes in the display. The other operations are similar to those described in Chapter 5.

**6.5 Performance Assessment**

This section will basically look at the time required for the painting of the representation to be completed for the application version.
Several tests have been carried out with different number of nodes (for the application version). The details and interpretation of the corresponding timings are:

*Test*

A series of runs of the program were carried out with different number of nodes. 5 readings were taken for each case, which considered the display of the directory structure and the display of the session information separately. The readings of the directory structure will only be interested in the time taken for the `repaint()` method to redraw the representation. The session information will include the time taken for the `repaint()` method to redraw the representation and the traffic between individual nodes.

The table below details the result:

<table>
<thead>
<tr>
<th>No. of Nodes (Files/Folders)</th>
<th>Display of Directory Structure (s)</th>
<th>Display with Session Information (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>28 (20/8)</td>
<td>0.0353</td>
<td>0.0680</td>
</tr>
<tr>
<td>213 (99/144)</td>
<td>0.190</td>
<td>0.236</td>
</tr>
<tr>
<td>539 (400/139)</td>
<td>0.385</td>
<td>0.429</td>
</tr>
<tr>
<td>1382 (882/500)</td>
<td>1.164</td>
<td>1.432</td>
</tr>
<tr>
<td>18501 (15901/2600)</td>
<td>12.568</td>
<td>stack overflow</td>
</tr>
</tbody>
</table>

The time required for the `repaint()` method increases dramatically when nodes are greater than 15 000, which can be seen from the table above. The addition of the session information will not greatly affect the method, but requires a large amount of memory space.

In order to improve the scalability of the program, future work will involve the search for algorithm that will simply the `repaint()` method.
6.6 Final Evaluation

Meeting the Objectives

The objectives of the project were to provide a tool for visualising clickstreams, which would provide the user with a dynamic method of grouping and ungrouping the file nodes. Along these lines, this project was able to provide a visualisation tool that allowed for a relative large number of nodes to be viewed via a circular display. In addition, it provided a starting point for the grouping and ungrouping by including the hierarchical structure of the files requested from the URL. The inclusion of hierarchical information also provided the user with a view of the directory structure of the web server. The user is able to perform two forms of grouping: the first by collapsing the individual folders and the second to group individual nodes by selection. The display will then be updated dynamically, which worked as designed.

The further objectives of providing a design of a query language were attempted with the language definition detailed in section 4.4.9. Certain queries were implemented, and the results provide alternatives for the user. The alternatives will allow him to concentrate the representation on partitions that satisfy the specified conditions. However, this aspect of the project will still require much work to make it more powerful.

The last objective is the integration with Kensington. An alternate version of the tool was created to work within Kensington and the integrated version works with the Kensington tables, and is able to export the files required for use with the other project by James Franklin.

Input Data Source

A different approach was taken in terms of the input data source for the visualisation tool. This was based on the assumption that any pre-processing will not be done by the tool, but by an external source. The design of the tool was then to display the directory structure to allow the user to view the files and select those that are of interest. An external pre-processing will then be done to filter out the rest of the session information. Then the tool will be used to view the session information of the nodes selected. From this point, the user can then dynamically modifying the structure up to the maximum specified in the earlier stage. The difference in this approach from the traditional single file, two columns approach was evident in the difficulty when integrating with Kensington. The Kensington version reads the data in directly from the file, processing is done on the data and held in memory for future use. This was the main difference in the two versions implemented.

Speed

For visualisation tools, speed is of importance. While for a small number of nodes, the perception of a slow algorithm is unnoticeable. For large number of nodes, this tool does not response well. However, the issue of speed can be countered in the future by the better algorithms used and a change of the data structures, specifically changing the adjacency matrix to that of an adjacency list.
This chapter will conclude the project by discussing the applicability to other related work and to discuss the future work of the project.

7.1 Overview

The tool as the result of the project is not complete. To improve the project further, the rest of this chapter will look at how this tool can be used in line with other tools and the future work that could be carried out.

7.2 Applicability

*Pre-processing for log file analysis*

For pre-processing of log files, it will be extremely useful to have the directory structure of the files to set which files are of interest and to which depth should the folder go down to. It would also be useful to provide groupings that are not based on the directory hierarchy of the files. With the tool linked to a pre-processing tool, this could be achieved and provide the user with a dynamic and cost-effective (by not keeping vast amount of information) way to analyse log files.

*Sequence analysis*

Although there will definitely be better methods of sequence analysis already developed. This tool could be used as a visualisation technique for displaying the sequences. Clickstreams are basically sequences, and by adapting the tool to an alternative sequence, *words*. The alphabets can basically be split into vowels or consonants, which are used for the example. By considering an example document with 2078 words, we can analyse the words that make up the documents (seen in Figure 7.1).

![Figure 7.1 JVisC in Sequence Analysis](image)

By setting the limit to only show paths with traffic greater than 200, we note the letters e, l, h, n, r and t. It probably shows that we often used the words then, there and in.
7.3 Future Work

The last part of this section of this report will list the possible improvements and enhancements that can be made to the project. Improvements basically discuss the parts of the tool that should be improved upon and Enhancements are new ideas that can, and should be implemented within the tool.

Improvements

The following is a list of improvements that can be made to the existing visualisation tool. They are listed in the order that will provide the most overall benefit:

1. As mentioned in Chapter 6, algorithms can be changed to improve the response time for the tool. The algorithms will include the method for repainting and the mouse listeners.

2. Extend the functionality of the JVisC Sequence Query Language (SqQL), which will provide a powerful tool for partitioning the session information. A limitation in the current implementation is that the operators AND and OR are not yet developed. An alternative would be to provide the user with a prompt for the queries to be entered.

3. Change the data structure of the adjacency matrix to that of an adjacency list. This improvement will increase the efficiency of memory usage.

4. Improve the scalability (in terms of depth) for the Concentric Circle View by providing allowing the nodes to have varying sizes, such that the nodes further away from the centre of attention will be smaller.

5. Put the tool through a usability test to fine-tune the user interface, such that the user interface will be easily understandable by a wide range of users.

6. The tool can also incorporate the .user file that could provide a method of identifying the sessions belong to a particular user, or a particular group of users and displaying the information using the tool.

Enhancements

The following is a list of enhancements, which are new ideas that can be added to improve the functionality, which are in no order of importance:

1. Provide addition information from the log file. Improvements in this area would require identifying what additional information is required. A possible suggestion is the inclusion of the time spend within each file being used as a weight rather than the transitions between individual pages. However, the time stamp would require a great deal more storage space, but may provide information that is important to certain groups of people.
2. While being inherently a visualisation tool for clickstreams, the functionality can be enhanced by making use of the data mining to correlate the session information with the type of the file.

3. The tool can be connected to algorithm that can be used to identify frequent paths and displayed on top of the display via animation methods. Additional animation methods can also be used to identify the path that an individual user or a particular group takes.

Due to the limitation of time, many suggestions that were made have not been implemented in the visualisation tool. Hopefully, what is included in this section should encompass most of those suggestions that were made.