1998

Performing Cooperative Work and Conferencing on an Intranet Using Java

Thaveeporn Limpanyalers

Edith Cowan University

Recommended Citation

This Thesis is posted at Research Online.
https://ro.ecu.edu.au/theses/1440
You may print or download ONE copy of this document for the purpose of your own research or study.

The University does not authorize you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following:

• Copyright owners are entitled to take legal action against persons who infringe their copyright.

• A reproduction of material that is protected by copyright may be a copyright infringement. Where the reproduction of such material is done without attribution of authorship, with false attribution of authorship or the authorship is treated in a derogatory manner, this may be a breach of the author’s moral rights contained in Part IX of the Copyright Act 1968 (Cth).

• Courts have the power to impose a wide range of civil and criminal sanctions for infringement of copyright, infringement of moral rights and other offences under the Copyright Act 1968 (Cth). Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.
USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.
Performing Cooperative Work & Conferencing on
an Intranet using Java

by

Thaveeporn Limpanyalers

A dissertation submitted in partial fulfillment
of the requirements for the Award of

Master of Science

at the

Faculty of Science, Technology and Engineering,

Edith Cowan University

Date of Submission : 24th Nov 1998
ABSTRACT

The process of organising, preparing and conducting a meeting is a time consuming one. Additionally, geographic separation compounds the amount of effort required.

Within this study, an electronic conferencing system in an Intranet environment is designed and implemented, thus enabling employees in the same organisation to meet up electronically without the dual constraints of time and space. Specifically, the study investigates incorporation of facial expressions and an ongoing polling system.

The results demonstrate that a presence of facial expressions helps users to better engage in on-line discussion and that users find an ongoing poll to be useful in determining direction within an electronic meeting.
DECLARATION

I certify that this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any institution of higher education; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

Signature: ______________________

Date : ______________________
ACKNOWLEDGEMENTS

I never truly believe that I would finish what I've started, yet there are some special people who have inspired and encouraged me along the way. To you, I wish to express my deepest gratitude.

Thank you Dad, Mum and my family for the love and encouragement that you have shown me and most of all for believing in me.

Thank you Bob Cross for urging me on and guiding me through.

Thank you Mona Fong for your love, motivation and endless support through the toughest times.

Thank you Daniel Jongue, Simon Tu and Judy Smith for being there for me.

Thank you Jean Yues Benedeyt for your patience and understanding. You redefined the meaning of friendship in cyberspace.

Finally, to all those people who took part in the experiment – thanks a million.
1. INTRODUCTION

1.1 FACE-TO-FACE INTERACTION

1.2 ELECTRONIC CONFERENCING – AN ALTERNATIVE

1.3 THE NEED FOR FACIAL EXPRESSIONS

1.4 DOCUMENT STRUCTURE

2. FRAMEWORK

2.1 BACKGROUND

2.1.1 The ‘Three Cs’

2.1.2 Communication

2.1.3 Collaboration

2.1.4 Co-ordination

2.1.5 Coordination Through Groupware

2.1.6 Achieving the ‘Three Cs’ with an Intranet

2.2 SIGNIFICANCE OF THE PROJECT

2.3 PURPOSE OF THE STUDY

2.4 RESEARCH QUESTIONS TO BE ANSWERED

2.5 ASSUMPTIONS AND LIMITATIONS OF THE STUDY

2.5.1 Small Testing group
3. LITERATURE REVIEW

3.1 GENERAL LITERATURE

3.1.1 Description of Electronic Conferencing

3.1.2 Examples of Electronic Conferencing systems

3.1.3 Other examples of Electronic Conferencing system

3.2 LITERATURE SPECIFIC TO STUDY

3.3 SIMILAR STUDIES

4. RESEARCH DESIGN

4.1 THE ISSUE OF TIME AND SPACE
4.2 THEORETICAL FRAMEWORK OF ELECTRONIC CONFERENCING SYSTEMS

Example 1 – Using existing framework

Scenario

Activity 1 – Preparation

Activity 2 - Meeting

Activity 3 – Post Meeting Discussion

Example 2 – Using Electronic Conferencing in an Intranet environment

Activity 1 – Preparation

Activity 2 – Meeting and Discussion

4.3 FUNCTIONS BEYOND SCOPE OF CURRENT STUDY

4.3.1 Real-Time Collaboration

4.3.2 Message Threading

4.4 PROGRAM REQUIREMENTS

4.4.1 Web Access

4.4.2 Intranet Access

4.5 Program Interface Design

4.6 Design of Components

4.6.1 Topic & Discussions lists

4.6.1.1 Topic list

4.6.1.2 Discussion list

4.6.2 Smilies

4.6.2.1 Overjoyed Smiley

4.6.2.2 Smiling Smiley

4.6.2.3 Neutral Smiley

4.6.2.4 Sad Smiley

4.6.2.5 Angry Smiley

4.6.2.6 Boring Smiley

4.6.2.7 NoPict/Blank Smiley

4.6.3 Colour Usage
5. EXPERIMENTAL OUTCOMES

5.1 USERS’ REACTIONS TO SMILES

5.2 OUTCOMES FOR RESEARCH QUESTIONS

5.2.1 Research Question 1

5.2.1.1 Analysis of basic expressions

5.2.2 Research Question 2

5.2.2.1 Colours Compared to Expressions

5.2.2.2 Comparison of Different lists

5.2.3 Research Question 3

5.2.4 Research Question 4

5.2.5 Research Question 5

5.2.6 Other Findings

5.2.6.1 Smilies

5.2.6.2 Terms used in the program

5.2.6.3 Topic and discussion names

5.2.6.4 People with strong literacy skills

5.2.6.5 Moderator

5.2.7 Suggestions

5.2.7.1 Real-time chat function

5.2.7.2 Include Message-Threading

SUMMARY

6. CONCLUSION

6.1 PROJECT OVERVIEW

6.2 PROJECT OUTCOME

6.3 FUTURE RESEARCH DIRECTIONS

6.3.1 Number of Subjects

6.3.2 Number of Smilies

6.3.3 Introduce Message Threading
<table>
<thead>
<tr>
<th>CLASS NAME</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITPROGRAMFILES.JAVA</td>
<td>257</td>
</tr>
<tr>
<td>HOLDDATA.JAVA</td>
<td>263</td>
</tr>
<tr>
<td>EXECUTEICONVOTE.JAVA</td>
<td>267</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>318</td>
</tr>
</tbody>
</table>
LISTS OF FIGURES

Figure 3.1 A Talk session in progress. 22
Figure 3.2 Netscape Usenet Newsgroup program. 25
Figure 3.3 Productivity and Information Overload. 27
Figure 3.4 Display of emotion using text. 36
Figure 3.5 Example of an IRC chat session. 41
Figure 3.6 Portion of Collaboration-at-a-Glance program. 41
Figure 4.1 Face-to-face groups. 46
Figure 4.2 On-line groups. 46
Figure 4.3 Timeline for conducting a face-to-face meeting. 48
Figure 4.4 Timeline for conducting a meeting using electronic conferencing system. 50
Figure 4.5 Open Intranet page. 56
Figure 4.6 Logging in to the web server. 57
Figure 4.7 Accessing electronic conferencing 57
Figure 4.8 Topic & discussion lists with empty entries 58
Figure 4.9 Topic creation box. 60
Figure 4.10 Discussion creation box. 61
Figure 4.11 Image selection box 63
Figure 4.12 Overjoyed Smiley. 68
Figure 4.13 Smiling Smiley. 68
Figure 4.14 Neutral Smiley. 69
Figure 4.15 Sad Smiley.
Figure 4.16 Angry Smiley.
Figure 4.17 Boring Smiley.
Figure 4.18 Blank Smiley.
Figure 4.19 Using smilies with same colours.
Figure 4.20 Using smilies with different colours.
Figure 4.21 Implementation of a polling meter.
Figure 4.22 Number of people in favour is GREATER than those against.
Figure 4.23 Number of people in favour is LOWER than those against.
Figure 4.24 Number of people in favour is EQUAL to those against.
Figure 4.25 Submit opinion box.
Figure 4.26 Change opinion box.
Figure 4.27 Entering a topic name.
Figure 4.28 A newly created topic name.
Figure 4.29 Error message prompting for a discussion name.
Figure 4.30 Entering discussion.
Figure 4.31 A newly created discussion name and NoPict/Blank smiley.
Figure 4.32 Creating a new discussion.
Figure 4.33 Selecting Neutral smiley.
Figure 4.34 Entered data with a Neutral smiley.
Figure 4.35 Updated discussion list.
Figure 4.36 Submit opinion box.
Figure 4.37 One user is against an idea. 93
Figure 4.38 Submit a previous opinion. 94
Figure 4.39 Change previous opinion. 94
Figure 4.40 Changed opinion. 95
Figure 4.41 Same number of people In Favour and Against an issue. 95
Figure 5.1 How the discussion list arranges the discussion names and
smilies. 106
Figure 5.2 Questionnaire-type choices. 110

LISTS OF TABLES

Table 1 Main conferencing interface components. 59
Table 2 Explanation of Topic creation box. 60
Table 3 Explanation of Discussion creation box. 62
Table 4 Explanation of Image selection box. 63
Table 5 Test criteria. 87
Table 6 Test outcomes from the operation of the electronic conferencing
program. 97
Table 7 Additional expressions 102
Table 8 Suggested colours for smilies. 116
Table 9 Current implementation of the polling system without message
threading. 125
Table 10 Improvement with message threading. 126
1. INTRODUCTION

Before the invention of the telephone, people communicated either through writing, gestures or by word of mouth. These forms of interaction were important to allow people to discuss ideas, as well as to act upon them. In this chapter, an overview of face-to-face meetings, communicating by word of mouth, is presented. Then, a need for a new medium to be used as a tool to conduct an on-going discussion is discussed. Finally, the aims of the project are described and the contents of the remaining chapters are outlined.

1.1 Face-To-Face Interaction

Humans spend large amount of time interacting with one another. There are many forms of human interaction, of which one is meeting with people. As Mintzberg (cited in Sproull & Kiesler, 1993, p. 57) observes, "people in organisations spend much time in meetings and group discussions". Larkey and Sproull (cited in Sproull & Kiesler, 1993, p. 57) agree by suggesting further that “managers spend most of their time in this way”. Organisations, whether large or small, hold meetings. The frequency of meetings can be daily, weekly or monthly. Schrage (1990, p.119) points out that numerous surveys confirm “that American managers typically spend fully half their working days in meetings.” These meetings can be formal or informal but the purpose of such interactions remains an exchange of information.
Meetings may take several forms. The most basic form is face-to-face meeting, as Dix, Finlay, Abowd & Beale (1993, p. 470) comment, "face-to-face contact is the most primitive form of communication - primitive, that is, in terms of technology." Face-to-face is the most direct means of imparting a message to an audience. Knapp (cited in Harper, Wiens & Matarazzo, 1978, p. 77) point out that:

The face is rich in communicative potential. It is the primary site for communicating emotional states; it reflects interpersonal attitudes; it provides nonverbal feedback on the comments of others; and some say that, next to human speech, it is the primary source of giving information.

People may feel more comfortable talking to one another because of the instant feedback that is inherent in this type of communication. Hollan & Stornetta (1992, p. 119) say that "face-to-face conversation provides a richness of interaction seemingly unmatched by other means of communication." Indeed, Mehrabian is quoted by Sproull & Kiesler (1993, p. 40) as claiming that "93 percent of peoples' intent was conveyed by tone of voice and facial expression". Dix et al. (1993, p. 471) agree and note that if "we consider the style of communication, the interplay between different channels and productivity, we instead find that face-to-face is the most sophisticated communication mechanism available." This view is also shared by Takeuchi & Nagao (1993, p. 187) who state that in a human face-to-face conversation, "communication becomes highly flexible and robust, so that failure of one channel is recovered by another channel, and a message in one channel can be explained by the other channel."
The term channel used in this instance refers to the communication medium such as visual channel that carries facial expressions or an auditory channel that carries speech.

Nunamaker (1997, p. 377) mentions that "every executive wants to reduce time spent interacting face-to-face and aspires to having anytime-anyplace collaborative activities with teams from all over the world." If meeting face-to-face is so important, why do executives want to reduce the amount of time spent in this area? As Schrage (1990, p. 118) points out, the answer is that people dislike meetings.

Johansen, Vallee & Spangler (1979, p. 136) assert that "one limit of face-to-face communication is that everyone must be present simultaneously for communication to occur." Further, they explain (1979, p. 138) that there are two problems for face-to-face meetings - the first is a barrier of geographic distance and the second is a barrier of time. Therefore, if face-to-face meetings are to be successful, all participants must share the same time and the same physical space. A great deal of effort is spent getting prepared for meeting, travelling and taking part in such meetings. This has become increasingly difficult in our hectic world. Thus, a new medium for communication is needed for ongoing discussions to take place.

1.2 Electronic Conferencing - An Alternative

Aronson (cited in Sproull & Kiesler, 1993, p. 151) submits that "the telephone extended peoples' psychological neighbourhood in the first half of the twentieth century by letting people share common context and mundane events independent of physical
proximity". Adding to this observation, Sproull & Kiesler (1993, p. 151) suggest that "computer-based communication may allow people to extend their organisational neighbourhood in much the same way." Hence, electronic or computer conferencing has proven to be an appropriate alternative. According to Eastmond (1995, p.11), electronic or computer conferencing "refers to a group of people who engage in communication, learning, or decision making, mediated by a computer network application."

The introduction of both electronic conferencing and electronic mail (E-mail) has eliminated both the time and space restrictions imposed on face-to-face meetings. Using these systems, employees in the same organisation but at different time zones may also contribute to the discussion.

Apart from this, electronic conferencing has additional benefits. In face-to-face meetings, people normally meet for a short period of time to share information. This is applicable when quick decisions have to be made or new information presented, but when it comes to brain storming sessions, it presents a new problem. One may ask what happen when new suggestions are thought of after a meeting, or when a particular issue needs to be debated and discussed over a long period of time. Holding numerous meetings over an extended period of time is one solution, but requires more resources in terms of time and commitment by all the participants.

According to McGrath (1990, p. 42), short meetings evident in face-to-face meetings may be counter productive. He observes that groups that are required to accomplish a task with short time constraints will tend to work faster but with a lower
level of quality. Such a group may skip certain tasks in a development stage to make up for a short time frame. More importantly, he suggests that much of the communication activities would be eliminated in the process, resulting in poor interpersonal relationships between group members. On the other hand, he observes that when groups are provided with a large amount of time to accomplish a task, the groups normally work at a slower rate and produce a higher level of quality, because a group may pay more attention to tasks which are required to finish the project. Interpersonal relationships between group members may be better since there are constant communication activities. Such a form of interaction can be achieved through computer conferencing.

1.3 The need for Facial Expressions

As previously stated, one of the main advantages of face-to-face meeting that cannot be ignored is the presence of non-facial cues, particularly facial expressions.

With regard to the field of neurophysiology, Perret et al. is quoted by Takeuchi & Nagao (1993, p. 187) as claiming that “a particular region of primates’ brains is dedicated to facial information processing.” They add that it is this capability of processing facial information that enables primates’ to survive natural selection because facial expressions may be viewed as signals that help coordinate conversation.
1. INTRODUCTION

In their study, Takeuchi & Nagao (1993) propose that the availability of facial expression is helpful upon first interaction with the system. They stress that interfaces with facial expressions "reduce the mental barrier between the users and the computing system."

While facial expressions are important, current electronic conferencing systems do not facilitate their use. McDaniel, Olson & Magee (1996, p. 39) point out that one of the main difference between electronic conferencing discussions and face-to-face meeting is that electronic conferencing is text-based and face-to-face discussions are oral. Other non-verbal cues evident in face-to-face meetings are also lacking in current electronic conferencing systems.

In summary, meeting face-to-face means that people are obliged to share the same time and space, which may be considered to be time consuming and less cost effective. Electronic conferencing allows more flexibility, but current electronic conferencing systems are mostly text-based and do not incorporate facial expressions that are evident when meeting face-to-face.

This study will evaluate incorporation of facial expressions into electronic conferencing systems and establish that use of facial expressions enhances on-going discussions by maintaining some of the visual cues normally available in face-to-face meetings. A direct representation of a participant in an electronic conferencing is not presented. Instead, visual cues that depict the reaction and mood of the participant are employed.
1.4 Document Structure

Chapter 2 discusses the background of the study and explains the importance of the 'three Cs' - explained within the chapter. The choice of using Intranet technology compared to Groupware technology is also discussed, together with the reasons for choosing the Java programming language. Additionally, the significance of this study is discussed and the research questions are defined.

Chapter 3 contains a review of relevant literature concerning the advantages and shortcomings of existing electronic conferencing system. A justification for the approach taken in this study is developed through a critique of existing electronic conferencing designs and developments. Specific reference is made to the Collaboration-at-a-Glance program developed by Donath (1994) and to a study of the usefulness of facial expressions by Koda & Maes (1996).

Chapter 4 builds upon the principles outlined earlier in the chapter three. Details about the design, requirement and implementation of the proposed program are elaborated. Moreover, a theoretical framework of existing electronic conferencing programs is explained, followed by a verification of the proposed system with test criteria. Several snapshots of the program interfaces are also illustrated.

Chapter 5 details the outcomes of the experiment. This chapter tackles each research questions through the elucidation of relevant evidence and appropriate discussions. Suggestions made by the participants are also documented.
Chapter 6 concludes the project. A summary of the purpose of the study is documented as well as the overall assessment of the design and implementation of the experimental program. Then, the areas where future research may be undertaken are outlined and discussed in detail.

Finally, this document includes a number of appendices to provide additional information about the program used for the experiment. The following is a summary of each section:

Appendix A contains a glossary of terms used in this document.

Appendix B contains listings of the client program.

Appendix C contains listings of the server program that resides in the web server and monitors incoming requests from the client program.
2. FRAMEWORK

The conferencing system is designed to allow an on-going discussion to be carried out on-line with the focus on improving such interactions with facial expressions. The system should fulfil specific requirements and assumptions as required. In order to establish a framework for the design and implementation presented in later chapters, this chapter examines the background, significance and objective as well as the requirements and limitations of the electronic conferencing system developed by this study.

2.1 Background

2.1.1 The 'Three Cs'

Following vast improvements in technology, people are capable of working together employing means other than face-to-face meetings. However, whichever way we work together, people have to communicate, collaborate and co-ordinate in order to successfully achieve a common goal.

The following sections explain how each of the components of the 'three Cs' plays a part in helping people to achieve their task.


2.1.2 Communication

Communication is perhaps one of the most important activities for human beings. According to Cherry (cited in Short, Williams & Christie 1976a, p. 153), communication refers to "the physical signals whereby one individual can influence the behaviour of another". Short, Williams & Christie (1976b, p. 169) stress the importance of communication when they state that "two isolated individuals unaware of one another's existence and with no communication can naturally have no influence on one another. but as soon as contact occurs, performance is affected." Therefore, whether people are working together on a project or engaging in a common sport such as playing basketball, they need to communicate effectively. This may be achieved through verbal means such as a formal meeting and also through informal discussion; or in the sporting example, through non-verbal means such as eye contacts and body gestures so as to signal another player to pass a ball.

Communication may take on many forms. People meet up socially to engage in interesting topics and discussions. The introduction and the wide spread use of telephony and of email enable people to converse almost anywhere in the world and at any time. As a result, there are usually misunderstandings and misgivings between colleagues and professionals working together.

Schrage (1990, p. 6) believes that communication does not merely involve the exchange of information because the use of symbols, images, words and models are also significant for people "to construct relevant meanings from both the available
information and their individual expertise.” He says (1990, pp. 30-31) that there is a model of communication consisting of a person talking and another person listening, however he says that as a result, there are usually misunderstandings and misgivings between colleagues and professionals working together. Additionally, another explains a need for a different quality of interaction, one that helps to establish a shared understanding between everyone involved. This is the function of collaboration, which is explained in the next section.

2.1.3 Collaboration

Lotus Corporation (1995a) asserts that “collaboration relies on a shared space . . . [which] serves as a touchstone for the act of collaboration, and it is essential as a medium to manage the ambiguity inherent in human interaction.”

In any business enterprise, a general manager normally conducts numerous meetings so as to interact with employees. These meetings typically involve the employees reporting on their latest progress and this may result in further discussions. Schrage (1990, p. 31) observes that these people are not really collaborating with one another, instead, they are merely exchanging information when he says “collaboration means that people are less interested in displaying data than in creating a shared space to play collectively with ideas and information.”

Thus, the key to collaboration is not merely to collect data but to understand the data provided. It is the understanding part that should be achieved.
2.1.4 Co-ordination

The third element that forms the ‘three Cs’ is co-ordination. So far, we know that people have to communicate and collaborate in order to share information as well as to increase their knowledge about particular information. Lotus Corporation (1995c) claims that much of these interactions occur in a very unstructured nature.

People normally access information only when required. They interact with one another in a dynamic and as needed basis. This may be viewed as being logically and principally correct but Lotus Corporation (1995c) insists that organisations are run in a much more structured manner, one that requires all employees to obey company policies and go about accomplishing their jobs in a structured and step by step manner. For example, a marketing report has to be properly checked, approved and presented before it is carried out. All these activities need to be properly co-ordinated so that a project may be achieved accordingly within budget to achieve a predefined set of requirements. Co-ordination, therefore, acts as the blanket that covers the two important areas of communication and collaboration.

2.1.5 Coordination Through Groupware

The concepts of groupware complements coordinated work. Implementation of the ‘three Cs’ does not guarantee success unless undertaken by all parties involved. Argyle (cited in McConnell 1994, p. 12) defines this factor as “acting together, in a
coordinated way at work, or in social relationship, in the pursuit of shared goals, the enjoyment of the joint activity, or simply furthering the relationship."

Lotus Corporation (1995d), supported by the Yankee Group (1996), assert that coordination is a fundamental principle within a groupware environment. The concept of groupware is now being seriously challenged by Internet technologies. Leon (1994, p. 229) defines the Internet as "a set of protocols for data transmission and control, a set of services or utilities, and the people who use them." The Internet represented a common platform whereby information can be accessed easily and conveniently. According to the author, the Internet is also described as "a highway of ideas and as a library without walls". The accessing and transferring of this large amount of information and ideas are made easier via the use of Internet World Wide Web browsers such as Netscape Corporation's Navigator. These browsers have a 'point and click' interface which link each user to a huge mass of information.

2.1.6 Achieving the ‘Three Cs’ with an Intranet

To embrace the ‘three Cs’, a new concept was born to take advantage of the Internet, namely an Intranet – a term used to describe implementation of Internet technologies within a corporate organisation instead of external connection to the global Internet. An Intranet takes full advantage of Internet open technology to achieve co-operation in an internal organisation.
Australian Personal Computer (1996, 70) states that although “traditional groupware solutions can streamline an organisation’s communications and enhance collaboration, they also have a reputation for complexity which can result in high startup and running costs.” Moreover, the editor of Australian Personal Computer (1996, 70) suggests that “Intranets, which take the open technology of the Internet and apply it to internal company networks, offer a cheap and relatively simple alternative.”

From the definition of electronic conferencing and its potential advantages discussed in Section 1.2, there is an understanding that electronic conferencing offers a new way for people to interact without regard for time and space. Therefore, if electronic conferencing is to be implemented within an Intranet environment, it will allow people within the same organisation to hold an ongoing discussion without the knowledge of the wider community residing outside the organisation. This could bring about numerous benefits not evident in current electronic conferencing systems. This benefit is discussed in the next section.

2.2 Significance of the Project

A great deal of time is spent organising, travelling to and attending meetings which might only last for a very short time. Many people are constrained either by being geographically distant or by limitations of time. In the business world, ‘Time is money’ and therefore costs incurred due to wasted time may add up to quite substantial amounts.
To encourage a more productive outcome from meetings and an on-going discussion, a common platform is needed that is readily available to everyone and is cost efficient. Internet technology offers such flexibility. It is not only far-reaching but also easily accessible. That means that the Intranet contains not only the same services evident in Internet technology but has the added function of being able to restrict the access of that information for internal use only. Therefore, the Intranet is a suitable medium for an electronic conferencing system to be built upon.

Having established a suitable technology, a tool to use the technology is required. Such a tool is the Java programming language. Developed by Sun Microsystems, it is designed to be small, simple and portable across heterogeneous platforms and operating systems. More importantly, Java can be programmed to work with Internet Web browsers or as a stand-alone program.

Java is a fairly new programming language at the time of this writing, and has mostly been programmed for trivial tasks such as executing animation programs (applets) on the Web browsers. A significant part of this project is also to determine if Java can be used to develop more complex applications. This study subjects the Java programming language to the rigours of developing a complex application.

2.3 Purpose of the Study

This study investigates the possibility of developing an electronic conferencing system that uses facial expressions and textual information to conduct on-going
discussion. By developing such a conferencing system in an Intranet environment using Java, a common medium is available for an organisation's personnel to contribute their ideas without time or space constraints, allowing people in an organisation to become more involved in discussions during a meeting.

2.4 Research Questions to be answered

The focus of the study is in the following areas:

- Will a single presentation of facial expressions make electronic conferencing more acceptable?
- Is the use of colours to represent different facial expressions useful?
- Will an ongoing poll make electronic conferencing more acceptable?
- Do the users find the colours implemented in the polling system helpful?
- Will the electronic conferencing program promote better discussion?

2.5 Assumptions and Limitations of the Study

Since the project deals mainly with human test subjects, the result is a subjective one. Some of the main drawbacks of this study include:
2. Framework

2.5.1 Small Testing group

The project was tested on eighteen subjects, providing indicative results only.

2.5.2 Java Programming Language

The Java programming language is the software for the development of the proposed system. Java is employed because it has been designed to be platform independent (Gosling & McGilton, 1995; Lemay, 1996). As Gosling & McGilton (1995, 14) explain, “Java is designed to . . . be deployed into heterogeneous networked environments . . . . [Hence], your programs are the same on every platform – there are no data type incompatibilities across hardware and software architectures.” Mendelson (1997, 103) emphasises that “the advantage to using Java programs is that you can download them from the Internet or a server when you need them, and then run them on any platform with a Java virtual machine.”

Another important aspect of Java, substantiated by writers such as Naughton & Schildt (1997) and Enck (1997), is that it supports the TCP/IP network protocol. Enck (1997, 67) stresses that “Java applications can be standalone or client/server applications that ride over TCP/IP, the most popular network protocol on the planet.” Furthermore, Dragan & Seltzer (1997, 100) observes that Java “has evolved from a tool for dressing up Web pages to the
next great platform for Internet/Intranet computing." Therefore, Java makes the task of developing a client/server program in an Intranet environment much easier with its rich set of networking classes.

Nevertheless, at the commencement of this project, Java was in its infancy and the earliest version of Java (version 1.02) was used. Much of the other functionality of Java's later versions such as database access (JDBC) was not available, leading to a less efficient method for storing discussion information. Although this did not lead to problems with the program, it did slow the system down during the experiment.

Summary

To work harmoniously and effectively, people need to Communicate, Collaborate as well as Coordinate their activities together to achieve a common goal. Therefore, co-operation by all parties involved must be achieved.

In traditional meetings where people meet face-to-face to conduct discussion and exchange information, people have to maintain the same place and time to make it possible. Electronic conferencing eliminates that constraint and enables a more prolonged and thorough discussion.
While electronic conferencing offers advantages, the majority of systems available today are mostly text-based, which is restrictive because text often results in ambiguity and misunderstanding. This study ascertains that facial expressions may be used to make the discussion more productive and to reduce some of the ambiguity presented in a text only environment.
3. **LITERATURE REVIEW**

The aim of this chapter is to discuss the usefulness and limitations of electronic conferencing systems in general. The main focus is on specific studies undertaken by researchers in this area and the different approaches taken thus far.

3.1 General Literature

3.1.1 Description of Electronic Conferencing

Electronic conferencing has been established in Section 1.2 as an alternative to face-to-face meetings. Gehris (1998, p. 140) defines computer conferencing as "an electronic means of sending, viewing, and sharing real-time communications in areas of common interest via one or more of the following media formats: text, graphics, audio, video, and/or animation."

Eastmond (1995, p. 68) offers an alternative view of electronic conferencing as:

having a room full of typewriters, perhaps set in groups on different desks, clustered by subject area. And, each one is labelled with a specific writing topic. During the week, day and night, men and women enter the room from different doors, rarely at the same time. They spend . . . . time reading what
3. Literature Review

others have typed . . . on each topic and adding their own ideas. Then they leave, only to return in a couple of days to see who has responded to them . . . and how the various topical discussions have progressed.

3.1.2 Examples of Electronic Conferencing systems

According to Gehris (1998, p. 141), there are two types of computer conferencing application programs - Unix-based and Web-based. This section examines each in more detail.

3.1.2.1 Unix-Based programs

Two Unix-based programs are discussed. The first is a program known as Talk and the other is a more popular program called Internet Relay Chat or IRC. Both programs are text-based and easily accessible.

Talk may be considered to be the simpler of the two programs. The concept of Talk follows very closely the practices found in the face-to-face model. In such a model, there must be at least two people for a conversation to take place. Similarly, to establish a Talk session, there must be someone available to accept a request for a chat to commence—that is, the person who is initiating the Talk must find someone on the host computer or other computers in different networks to
establish a link. Talk was not designed to handle multiple connections with many people, therefore, only two people may 'talk' at one time. An example of a 'talk' session is shown in Figure 3.1.

![A Talk session in progress.](image)

*Figure 3.1* A Talk session in progress.

The most often employed program of the two is Internet Relay Chat. The reason for its popularity is that Internet Relay Chat allows a user to 'chat' on various topics with many people simultaneously. The only clue to a user's identity is through an alias that each one of the users has assigned for themselves. Thousands of topics are available on an Internet Relay Chat system.
3.1.2.2 Web-Based programs

Gehris (1998, p. 141) explains that the aforementioned text-based programs, Talk and Internet Relay Chat, have been available for use on the Internet for some time and claims that web-based programs "open up communication possibilities that are not possible or ineffective with the text-based programs." With improvements in Internet technology, numerous electronic conferencing programs have been designed to make full use of this open technology. Web-based programs are easily accessible and thus capture a wider number of users who wish to discuss various topics available on the system.

Gehris (1998, p. 142) provides two examples of such systems. The first program is called Netscape Chat and the other is known as WebChat. Netscape Chat is a program developed by Netscape Corporation, the same company that designed the World-Wide-Web browser - Netscape Navigator. The Netscape Chat concept is similar to the text-based Internet Relay Chat program counterpart, but additionally offering an ability to chat with multiple users on several topics simultaneously and an ability to share World-Wide-Web addresses or URLs (Universal Resource Locator) with other chat users.

The second Web-based program, Webchat, employs a different approach. It offers all of the functionality of Internet Relay Chat and, additionally, users have an ability to incorporate images and video clips as well as text into their chat. Donath & Robertson (1996) have developed similar type of Web-based program
known as 'The Sociable Web'. These new types of Web-based programs provide an improvement to systems that are mainly text-based.

While this represents an improvement to text-based systems, there is no specific implementation of facial expressions incorporated into these or other types of systems seen thus far. Furthermore, since each session is real-time, there is no record of a conversation, thus users cannot refer to previous discussions.

3.1.3 Other examples of Electronic Conferencing systems

All of the systems previously described provide real-time communication. However, the same problem of face-to-face meetings remains, namely that all participants must meet in the same space and time to conduct such sessions.

Nevertheless, there are systems that are designed to allow users to communicate without space and time restrictions. One such system is the Usenet Newsgroup program developed by Netscape Corporation and the other is the use of electronic mail or email.

3.1.3.1 Usenet Newsgroup program

Dyson, Coleman & Gilbert (1997, p. 10) define Usenet as "a network of computers that distribute Usenet newsgroups." Further, Crumlish (1995, pp. 77-79) says that "Usenet is a network of other networks . . . all of which have made
bilateral agreements with other members of Usenet to share and exchange ‘news’” which has been called the biggest bulletin board in the world . . . . [similar to] an enormous magazine rack, filled with magazines, each of which have infinitely long letters-to-the-editor pages and no articles.”

According to Crumlish (1995, p. 80), a newsgroup is “an ‘area’ for posting articles on a given topic. The topic names are arranged hierarchically, in order that people might be able to look for and find newsgroups of interest in a systematic way.” Users may access Usenet at anytime and read messages under different topics, either contributing their own ideas or responding to messages left by others. The newsgroup program is far more flexible in terms of time and space than other types of systems mentioned thus far. An example of a newsgroup program is shown in Figure 3.2.

![Figure 3.2 Netscape Usenet Newsgroup program.](image-url)
In recent years, the usage of email has increased considerably. As the name suggests, the main purpose of email is to allow people to send mail electronically to one another across time and space. Whittaker & Sidner (1996, p. 2) suggest that the use of email today has changed from a communication application only, using the term *email overload* "to describe the use of email for functions that it was not designed for." They provide some examples of email usage, including task tracking, appointment scheduling and storing names and addresses. Additionally, use of email has extended to include ongoing discussions, much like a newsgroup program. This presents a problem since it normally takes a few exchanges of email to resolve an issue; and there is no standard to follow with regard to whether a history of previous discussions should be included in the email as a conversation progresses.

According to Whittaker & Sidner (1996), such an activity leads to a big increase in the number of emails that people receive whilst they are involved in discussions and decision making. Since everyone who is participating in a discussion needs to be updated regularly with the latest information, each copy of an updated email should be forwarded to everyone. Besides having to organise the current volume of emails they receive, users must also take a considerable amount of time and effort to work out threads of a conversation within a list of emails. On the one hand, if previous discussions are included together with new input, an email will be lengthy and prove to be difficult to read. It is not included, it may be
difficult to follow a discussion, especially if several discussions are carried out simultaneously. Studies such as these by Grudin (1990) and Whittaker & Sidner (1996) suggest that too much time and effort is needed to read and reply to meetings based on emails, leading to users not responding to emails.

LotusCorp (1996b) shares similar views and states that "as the volume and complexity of information increases, users reach the point of diminishing returns in productivity." This is illustrated by a graph drawn from LotusCorp (1996b) shown in Figure 3.3.

![Graph showing Productivity and Information Overload](image)

**Productivity and Information Overload**

*Figure 3.3 Productivity and Information Overload.*
Summary of General Literature

The study undertaken appears to be significant based on literature relating to electronic conferencing. There are few electronic conferencing systems designed specifically to deal with facial expressions. In addition, no attempt has been made to allow user to display their opinion through a polling system.

The following section will address specific limitations of electronic conferencing systems, lack of non-verbal cues and a need for facial expressions. Use of colours and a concept of voting will also be addressed.
3.2 Literature Specific to Study

This section describes factors such as the benefits and limitations of current electronic conferencing systems, including the need for non-verbal cues such as facial expressions.

3.2.1 Current Limitations of Electronic Conferencing

Research into the usefulness of electronic conferencing systems has been conducted since the late 1970s (Johansen, Vallee & Spangler, 1979). There have been numerous studies into the behaviour of electronic group interactions and electronic conferencing systems, such as Olgren & Panica (1983); Cross (1984); McGrath (1990); Kakuta & Yoshida (1992); McNeese, Zaff & Brown (1992); McNeill (1992); Sproull & Kiesler (1993); Wilkins & Nantz (1994); Anderson (1995); Brandt & Briggs (1995); Eastmond (1995); McDaniel, Olson & Magee (1996); Whittaker (1996) and Gehris (1998).

A common theme among the literature is that most of the electronic conferencing systems are text-based, with the exception of WebChat and Netscape Chat, which are Web-based. Text-based systems have some limitations that will be discussed in the following section.
3.2.1.1 Issues with Text Only

Kinney & Dennis (1994, p. 22) observe that “when people communicate in traditional face-to-face interactions, they have both visual and auditory sensory channels to receive and send information -- a communication ‘rich’ environment.”

However, Dix et al. (1993, p. 489) note that “for asynchronous groupware (and even some synchronous systems), the major form of direct communication is text based.” This is restrictive, because “messages sent via computer text are often perceived as being more task focused and less personal” (Hiltz, Johnson and Turoff, cited in Kinney & Dennis, 1994, p. 22).

Another problem inherent in computer-mediated communication, as Kinney & Dennis (1994, p. 22) point out, is that it “takes longer to type . . . because users often send computer messages as one block of text, making it difficult to interrupt the sender.” This view is shared by McGrath (1990, p. 51) who suggests that “typing is much slower than talking even for the best of typists.” Several students in Eastmond’s (1995, p. 70) study also thought that it takes too much time to type in the messages as well as reading other peoples’ comments. The students claimed that the same task could be accomplished in a regular classroom in a shorter amount of time. Therefore they viewed this means of communication as inefficient.
While McDaniel et al. (1996, p.44) expressed similar views, they added, "contributions were, on average, almost three times longer" in computer communication than in face-to-face communication. Collaboration software that has the same capacity as E-mail messages allows users to view text messages in their own time. Similarly, when old conversations are available on-line, users (both old and new) have the advantage of being able to refer to those discussions before contributing their own. This allows them to not only familiarise themselves with how the group interacts but also allows users to catch up with previous conversations (Whittaker. 1996, p. 411).

This findings is consistent with Ochs's (cited in McDaniel et al., 1996) belief that systems which allow users to produce their discussion off-line before submission enable users to plan and reflect on their contribution. Thus, users may organise their thoughts much more thoroughly, which can be an advantage compared to face-to-face meetings where instantaneous feedback may lead to users saying things they did not necessarily mean.

In studies with distance learning students by Eastmond (1995. p. 69), it has been shown that some subjects say that "the text environment more truly conveyed ideas and aspects of peoples' personalities than in-person setting because these communiques were unhampered by irrelevant physical characteristics and cues." This observation supports a suggestion by McConnell (1994. p. 67) that "participants can use the time to dig deep into issues . . . [as well as] revisit 'old' conversations and restart them." Gehris (1997, p. 140) also
agrees that participants can easily review comments made by others. The same author says that some of his subjects believe that the availability of text written by others enables subjects to pay more attention to what is being said. Some think that by using a text-based environment, participants tend to think thoroughly about what they want to present to the recipients (whether the written words are too harsh or too general), before submitting their discussion. Nevertheless, Eastmond (1995, p. 69) warns that text only communication lacks body language. In interviews with one of his subjects, he observes that a large amount of body language is involved.

3.2.1.2 Lack of Non-Verbal Cues

Programs such as Webchat allows users to incorporate images, audio and video clips, however there appear to be no specific studies into the use of non-verbal cues in both the text-based and web-based systems. From discussion in Section 1.3, non-verbal cues are understood to be as important as verbal communication. According to Inoue, Okada & Matsushita (1995, p. 227), “non-verbal communication conveys feelings more deeply than verbal communication.” Birdwhistell is quoted by Inoue et al. (1995, p. 227) as claiming that “65-70% of a whole message is nonverbal”. Mehrabian (cited in Inoue et al., 1995, p. 227) indicated that the figure could be as high as 93%.
Non-verbal cues can be translated in many forms. Some examples include the use of gestures, facial expressions and body language. However, Kakuta & Yoshida (1992, p. 257) declare that computer-mediated communication “lacks the presence of non-verbal communication channels such as facial expressions, gestures, etc and therefore it tends to be said that it reduces social context cues.” This is particularly true for the majority of existing text-based electronic conferencing systems.

3.2.1.3 Lack of Facial Expressions

Several studies into the use of electronic conferencing have indicated a lack of non-verbal cues. Among the literature, Inoue et al. (1995, p. 228) suggest that non-verbal communication cannot be properly translated in traditional character based network communication. Nunamaker (1997, 375) goes further by blaming unstructured meeting process and missed signals as the culprit for failed and counterproductive meetings in existing conferencing systems and claims that the lack of “non-verbal cues such as gesturing, pointing, eye-gaze, handshake and other components of body language that encourage engagement in a meeting” attributed to this failure. One of the most common non-verbal signals that people rely on is facial expression.

Short, Williams & Christie (1976a, p. 157) assert that “the face is the most expressive part of the body. It is also the part of the body that is most visible and most attended to during the interaction and thus usually the area that is most
consciously controlled.” To support this argument, a study by Shapiro, Foster & Powell (cited in Short et al., 1976a, p. 157) found that “in judging genuineness, empathy and warmth in therapists, subjects responded more to facial than body cues; with body cues alone performance was negligible.” In their experiments, Takeuchi & Nagao (1993, p. 191) conclude that users who hold conversations with a system incorporating facial expressions have far better results than users of a system displaying only short phrases using text.

Users of electronic conferencing attempt to make up for an inability to express feelings. The following section is an overview of two conventions that are currently practiced by many people to express their feelings and moods through the use of text.

3.2.1.3.1 Text Symbols (Smilies)

The limitation of text has led to a number of users resorting to using symbols, otherwise known as Smilies, to express how they feel. These smilies are typically typed using standard text symbols and must be viewed sideways. McConnell (1994, p. 77) gives examples of smilies, as follows:

:)

I’m happy

:(

I’m sad

:S

I have mixed feelings

8)

I’m wide awake
3. LITERATURE REVIEW

This suggestion is echoed by Sproull & Kiesler (1993, p. 51) and Dix et al. (1993, p. 491) who report similar use of the smilies in well-established electronic mail communities. Additional examples of the smilies are as follows:

:-)     -     a joke or happiness
:-(     -     Bad news or unhappiness
;-)     -     winking face, humorous

Turoff (cited in Cowan, 1984, p. 230) suggests more:

( +)     -     Smile
( -)     -     Frown
-!          -     Decrease the emotion
+!          -     Increase the emotion

Although Sproull & Kiesler (1993, p. 51) agree that these symbols show the mood of a person writing a message, the authors argue that "such cues weakly signal mood, they are flat and stereotyped". They further suggest that the textual expression, such as a happy facial reaction, expressed by one person does not differ to that of another. i.e. the smilies do not suggest whether it belongs to the first or second correspondent.
3. Literature Review

3.2.1.3.2 "Emotional" Text

Nojima (cited in Inoue et al., 1995, p. 227) suggests that although a smiley does reduce misunderstanding, “the meanings of ‘Smilies’ are not always understood properly.” Inoue et al. (1995, p. 228) clarify this by stating that “receivers of a smiling ‘Smiley’ sometimes take it [as] an irony due to its context and shortage of non-verbal information.”

Besides translating text into pictorial form as smilies, users also use written text to express their emotions. Examples of this practice include a display of emotion such as <*Chuckle*> or a display of movement such as <*Shrug*>. Mynatt, Adler, Ito & O’Day (1997, p. 9) give a further example of how users might translate speech and bodily movement into text to express how they feel. Figure 3.4, reproduced from Mynatt et al. (1997, p. 9), illustrates how this might be done.

A warm fuzzy feeling approaches you ...


Tinlizzie

You’ve been enveloped in a warm hug by Hobbes.

Figure 3.4 Display of emotion using text.
3. LITERATURE REVIEW

While the text itself can be used to translate various emotions, the letters in the text are also important. The use of upper and lower case letters portray different meanings when included in a message. For instance, the message ‘DO NOT DISTURB ME!’ can have a different impact from using lower case letters such as ‘Do not disturb me!’ When upper case is used, it normally indicates “shouting and strong words . . . to suggest emphasis” (Eastmond, 1995, p.118). Therefore, great care must be taken when deciding on which case to use since a wrong choice will result in misunderstandings and be regarded as offensive.

3.2.2 Advantages of Electronic Conferencing

According to Eastmond (1995, p. 147), factors such as age, gender and race are not on-line issues, enabling more honest communications to take place. This view supports that of Zuboff (cited in Sproull & Kiesler (1993, p. 43), who state that

the remote and textural qualities of the computer . . . eliminated peoples’ advantage or disadvantage over one another. Those who regarded themselves as physically unattractive reported feeling more lively and confident when they expresses themselves in a computer conference.
3. LITERATURE REVIEW

Harasim (cited in Eastmond, 1995, p. 16) shares a similar view when he remarks that computer conferencing offers "a more equitable distribution of communication . . . because everyone has an equal opportunity to express themselves, and no one can easily dominate the conversation." McDaniel et al. (1996, p. 44) further adds that users contribute more in a computer-mediated environment than in face-to-face meetings where space constraints limit the number of participants.

Furthermore, Harasim (cited in Eastmond, 1995, p. 16) suggests that computer conferencing "amplifies intelligent development by supporting divergent thinking through idea-generating activities." Sproull & Kiesler (1993, p. 51) express a similar view. Studies done by the two authors concluded that:

As compared with their face-to-face counterparts, electronic groups will consult more people, which will increase the number of alternatives considered. They will more often ignore faulty reasoning promulgated by people who, face-to-face, have good social skills or organisational status.

Anderson (1995, p. 118) claims that one of the reasons for the popularity of electronic conferencing systems is that "groups can communicate without space and time restrictions . . . [thereby] creating potentially continuous dialogues." According to Olgren & Panica (1983), this is possible because electronic conferencing offers both synchronous (real-time) and asynchronous communication (messages are
3. Literature Review

stored). The authors state that “research has indicated that users strongly prefer asynchronous computer communication. Generally only 20 percent of conference groups bother to meet synchronously.” According to Johansen et al. (1979, p. 19), because people do not have to meet physically, they are safeguarded from uncomfortable meetings in unfamiliar surroundings.

In spite of these advantages, electronic conferencing will not replace face-to-face meetings completely. Sproull & Kielser (1993, pp. 73-74) argue that face-to-face is ideal when a complicated decision involving multi party negotiation must be made, since it is hard to persuade subtly via electronic means. The authors add that face-to-face meetings are more focused and productive since participants have a chance to review issues electronically before a meeting, hence, there is a likelihood of clearing up confusion and identifying key issues quickly. The two authors (1993, p. 153) sum it up best when they assert that even though electronic conferencing “does not substitute for face-to-face communication, it can keep people reminded of one another so that every new face-to-face meeting is not a meeting of strangers but a meeting of neighbours.”

3.3 Similar Studies

This section will examine studies undertaken by other researchers, especially Tomoko’s work which justifies the use of smilies. Another system investigated is the Collaboration-at-a-Glance program.
3. LITERATURE REVIEW

Commercial electronic conferencing systems have enabled the use of facial expressions, body gestures and other types of body language, for which studies have been conducted. One such study is that of Donath (1994), who developed a program called Collaboration-at-a-Glance.

3.3.1 Collaboration-at-a-Glance program

Donath (1994) suggests that a substantial number of programs have been created to support collaborative work. Nonetheless, these programs mostly emphasise task performance. The author argues that apart from tools that allow people to collaborate on specific task, it is also important to allow casual collaboration, which are unplanned meetings that take place frequently between colleagues. Although these meetings might take place along a corridor or in other informal environments, they are a highly productive part of work experience. Thus, the Collaboration-at-a-Glance program was built to support casual collaboration among members of electronic communities.

The Collaboration-at-a-Glance program provides a graphical representation of participants where each members can detect the presence of others, working in a similar manner to the Internet Relay Chat in which names of participants are denoted by nicknames assigned at the commencement of a session. Figure 3.5 provides an example.
3. LITERATURE REVIEW

<table>
<thead>
<tr>
<th>Nickname</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Zombie&gt;</td>
<td>Hi Guys! How are we today?</td>
</tr>
<tr>
<td>&lt;Batty&gt;</td>
<td>=8) Just Lovely !!!</td>
</tr>
<tr>
<td>&lt;Quack&gt;</td>
<td>No complaint as yet!</td>
</tr>
</tbody>
</table>

*Figure 3.5 Example of an IRC chat session.*

The Collaboration-at-a-Glance program displays images of each of participants’ face on the screen in addition to names. Figure 3.6 provides a snapshot of the Collaboration-at-a-Glance program.

*Figure 3.6 Portion of Collaboration-at-a-Glance program.*

The Collaboration-at-a-Glance screen in Figure 3.6 depicts a typical session among users of a system showing a group of faces turning toward one another as if in conversation. As a session progresses, there are occasional movements of the faces, with some of the faces turning to face a different person as they engage in conversation.
The Collaboration-at-a-Glance program attempts to use eye contact as a source of information. The program reproduces the same action as in a normal conversation where one would turn to face one another when initiating a conversation. Hence, all faces on the screen will in turn be looking towards the user that is the centre of attention. This provides a user with information on which members of a discussion group are speaking and listening. Thus, the program provides a graphical interface for an electronic group, using real images instead of caricatures or smilies. This will be discussed in more detail within this section. The Collaboration-at-a-Glance program appears to be the only attempt that uses non-text facial representation in a conferencing system.

In her study, Donath (1994, p. 494) uses snapshots of the users' head movements from every angle for use in the system. However, she maintains the same facial expression for each user of the system. This means that although the user might know with whom they are conversing, they have no indication of how the rest of the participants are feeling about a particular discussion apart from an exchange of text messages.

3.3.2 Interface Agents

There is a considerable amount of literature on the study of facial expressions, where the emphasis is on the use of real images, as evident in the work of Terzopoulos & Waters (1993), Tatsuno, Suzuki, Yokoya, Iwasa and Takemura
3. Literature Review

(1996) and Donath (1994); or in the use of synthesised three-dimensional images, as described by Takeuchi & Nagao (1993) and Takeuchi & Naito (1995). There have been very few studies into the use of other types of displays such as caricatures and smilies. However, there are studies in the fields of computer interface agents or social agents. Maes (1994) explores the usefulness and different approaches taken in building interface agents. Parise, Kiesler, Sproull & Waters (1996) show that a high level of cooperation may be achieved with a computer agent that looks like a person. On the other hand, a lower level of cooperation is achieved when a picture of a real dog and a cartoon dog are used to represent the interface agents. The study is less specific about the types of human-like interface agents that may be used.

In a similar study, Koda & Maes (1996) demonstrate that having a face does not have a negative impact on users. Instead, personified interfaces help to make a user more engaged in a task. While the two experiments yield the same result, the representation of interface agents used in the two studies varies. Parise et al. (1996) use real images and a cartoon dog but Koda & Maes (1996) include both male and female realistic images and caricatures, a cartoon dog and smilies. The Koda and Maes (1996, p.6) study shows that, in judging perceived intelligence between personified interfaces, caricatures and smilies, the subjects “rated all three faces to be equally intelligent, and they thought that the Smiley (less realistic) face represented the true level of competence more appropriately.” On the basis of this finding, smilies are used in the current study.
Summary

While there are a considerable number of studies available which investigate facial expressions, only the Collaboration-at-a-Glance program explores its use in an electronic conferencing system. The Collaboration-at-a-Glance program provides a graphical interface where people can meet socially in an on-line environment. The images of the users' face are manipulated in ways that create the impression that people are looking at one another when engaging in a conversation. However, only a single representation of facial expression is presented for each user in the program. There is no way of presenting emotions through the use of facial expressions except through a text-based messaging system. Thus, the program lacks a means of gauging general moods and feelings of participants in the group. Few studies exist that examine the use of other types of display such as caricatures and smilies except the experiments employing interface agents undertaken by Parise et al. (1996) and Koda & Maes (1996). Both studies demonstrate that having personified interfaces enhance users' co-operation and engage them in a task. When compared to real images, smileys provide a simplification of human-like expressions that may be represented easily and appropriately in an electronic conferencing system. For this reason, this study employs smileys instead of real facial images.
4. RESEARCH DESIGN

4.1 The issue of Time and Space

Within section 1.1, it has been established that face-to-face meetings are the richest form of communication compared to other medium such as email, video, audio and electronic conferencing because of immediate graphical and auditory feedback available when we communicate face-to-face. As McConnell (1994, p. 67) comments, "the sense of immediacy is perhaps one of the most important aspects of face-to-face meetings." However, Section 1.1 also indicates that meeting face-to-face has its drawbacks. One such drawback is time.

There is no doubt that time plays an important role in the running of our lives. According to McConnell (1994, p. 67), "time is important and is a limiter". That is, in most situations, we do not dictate time but rather time dictates us. In the business world, "time is money", therefore we have to maximise every effort and ensure that these efforts become fruitful. It follows, therefore, that time is a factor when conducting meetings. McConnell (1994, p. 67) states that in face-to-face meetings, "the work of the group is punctuated by periods of time when members work together, and periods of time when they are apart . . . . [which may result in] a loss of relationship between the group members . . . and some effort has to be put in rebuilding those relationships when members next meet." This observation is illustrated in Figure 4.1 which has been reproduced from McConnell (1994, p. 67).
A time restriction is more problematic when people are geographically separated, because people need to travel to meetings which requires more time. On the other hand, McConnell (1994, p. 67) suggests that meetings held on-line are continuous. There are no physical breaks evident between the current meeting and the next. More importantly, users of the system do not have to physically leave to attend these meetings. Hence, Johansen et al. (1979, p. 20) claim that distant experts can consult with a group more effectively since they avoid tiring travel and they remain close to their resources. Figure 4.2 is also reproduced from McConnell (1994, p. 67) and illustrates the use of time in an on-line group.

Figure 4.1  Face-to-face groups.

Figure 4.2  On-line groups.
4.2 Theoretical Framework of Electronic Conferencing systems

The previous section shows how time and space affects meetings. Within this section, Example 1 shows that existing methods of communication differ from electronic conferencing. Both video and audio conferencing suffer from a time constraint evident in face-to-face meetings but not a space restriction, because video and audio conferencing can be conducted while users are geographically separated. However, such meetings are generally conducted in real-time, which means that users must still be present at the same time to make the meeting possible.

The scenario given in Example 2 attempts to demonstrate how the same activities shown in Example 1 are carried out when electronic conferencing is employed.

**Example 1 - Using existing framework**

Events leading up to a meeting may involve numerous activities. Using existing methods for carrying out face-to-face meetings, video conferencing and audio conferencing, events may be categorised into three basic activities. Based on McConnell's (1994, p. 67) diagram on face-to-face groups shown in Figure 4.1, the diagram may be expanded further to include an example. The expanded diagram is illustrated in Figure 4.3.
Scenario

Jack is a state manager of a Western Australian cosmetics company. He wants to make a proposal about ways to increase the company's overall profit. He will be presenting his proposal to his general manager based in Sydney.

Activity 1 - Preparation

Before Jack leaves for Sydney to submit his proposal to the general manager, there are several activities that must be carried out. During this phase, several meetings are held between state managers to promote as well as discuss issues relating to the proposed plan. To gear up for support as well as to promote
awareness about the proposed plan, there will also be ongoing telephone discussions with colleagues in the Sydney branch.

In order to convince his general manager, Jack must plan how to present his proposal. He must also prepare his report, notes and presentation slides for the actual presentation. Finally, he will have to ring the general manager to make an appointment.

Activity 2 - Meeting

The actual meeting with the general manager and the board of directors will take a much shorter time than the preparation in Activity 1. The reason for this is that many of the issues will have been resolved during the ongoing discussion during Activity 1. By this time, Jack’s proposal should be either rejected or agreed to. For simplicity sake, assume that the proposal is accepted. Although the plan was approved formally in the meeting, some issues may be still unresolved.

Activity 3 - Post Meeting Discussion

After the formal meeting with the general manager, the changes are carried out. During this implementation stage, some issues, however small, might be unresolved. These unresolved issues will be continually discussed until the changes are finally made.
It can be seen that the whole process of making a decision and implementing a proposal is quite lengthy and time-consuming. From the preparation stage (Activity 1) to the final agreement of the proposed plan (Activity 3), much time and resources are consumed.

The whole scenario above assumes that the general manager and board of directors are available on the day of the discussion and that the proposal is resolved in only one session. Thus, if any of these assumptions are not true, the whole proposal may take much longer.

**Example 2 - Using Electronic Conferencing in an Intranet environment**

Using the same scenario as before, a likely outcome of using an electronic conferencing system on an Intranet based environment is demonstrated in Figure 4.4.

![Timeline for conducting a meeting using electronic conferencing system](image)

**Figure 4.4** Timeline for conducting a meeting using electronic conferencing system.
Activity 1 - Preparation

The activities that Jack has to carry out remain the same. The only difference is that instead of printing out all his notes and reports, he may now store all the data on his computer and allow the general manager and the board of directors to access these materials through the Intranet environment. This saves a significant amount of time. Furthermore, the travelling time required for Jack to travel to Sydney has been eliminated, reducing both time and cost.

Activity 2 - Meeting and Discussion

Activity 3 shown in Figure 4.3 has now been eliminated because all such activities have been combined into Activity 2. In an electronic conferencing session, there are continual discussions about the proposed plan. This is consistent with earlier suggestion made by McConnell that meetings in on-line groups are continuous.

Since it is conducted in an Intranet environment, all other state managers may also join in discussions without leaving their work-area. This means that all the participants of the meeting will remain in their 'comfort zone' close to their resources, with an expectation that they may respond more effectively.
4. Research Design

It may be seen from these examples that by implementing electronic conferencing on the Intranet, issues may be raised and discussed without many of the constraints imposed by time and space, leading to more productive and effective outcomes from meetings. Additionally, introduction of the technology described will not reduce participation rate of those involved. McDaniel et al. (1996) demonstrates that the use of technology does not prevent users of computer mediated conferencing from carrying out the same amount of interactions as they would have in a face-to-face meeting.

4.3 Functions beyond Scope of Current Study

Having established the advantages of electronic conferencing, the next step is to explain the design for the proposed system. The system is not designed to fulfill the complete requirements of an electronic conferencing system. Instead, the proposed system is simplified so as to satisfy the research questions outlined in Section 2.4. Some of the functions that are available in commercial electronic conferencing programs such as a shared white board or automatic generation of minutes are ignored in this study. Other functions omitted are real-time collaboration and message threading.

4.3.1 Real-Time Collaboration

Most of the electronic conferencing programs available, whether for research or commercial use, emphasise real-time collaboration. Talk, Internet Relay Chat,
NetChat and WebChat are limited by time constraints. Users have to be at the terminal at the same time to make such discussions possible. Thus, these interactions must occur in real-time.

Sometimes, collaborating in real-time does not always serve its intended purpose. Managers generally spend considerable time in meetings where their specific interest is only discussed for a short time. This study addresses a reduction in time required to attend long meetings for brief discussions, which may be best achieved if the proposed system closely resembles the newsgroup concept, enabling users to log into the program at a time convenient to them. This not only adds flexibility to the way discussions may be conducted but also eliminates time and space constraints.

Although the proposed system is not specifically designed for real-time collaboration, it is important to update the discussions posted to the system quickly. This way, users logged onto the program may view messages posted to the system by others without delay. This flexibility enables users to contribute whenever time permits or to attend an interactive (but not real-time) discussion with their fellow peers.

4.3.2 Message Threading

The Usenet Newsgroup program developed by Netscape Corporation uses message threading to organize messages posted to the program, enabling users to response to an individual instead of to a group. In the newsgroup environment,
message-threading is viewed as ideal since the program is normally connected to a specific server and participants are usually people all over the world. Allowing users to respond to specific individuals through message threading reduces confusion and enables users to respond only to the discussion that they are most interested in.

The Usenet News group programs are linked to the Internet, therefore any discussions held in the program are opened to the world. However, there is a mixed group of people taking part in this type of open discussion. Some of these people may be very interested in a discussion while others may be only 'passing by'. In terms of group diversity, Whittaker's (1996, p. 413) study shows conflicting results. Some of his participants believe that opening a discussion up to many groups allows different ideas and useful contributions to pour into the group, thus enriching the group's knowledge. However, some believe that people working closely together tend to be more committed to a discussion since their objectives and goals are the same, so that they may be more motivated to answer and contribute to discussions.

Unlike the newsgroup program, the proposed program is designed for a closed discussion group. In a group environment, Whittaker (1996, p. 413) stresses that having a small focused group allows participants to set ground rules so that contributions are relevant to the group's ultimate goals. Therefore, in contrast to an open discussion group, a closed discussion group allows everyone in a group to contribute to the group instead of to individuals. Thus, the provision of a message threading function is not employed in this study.
4.4 Program Requirements

In order to allow members of a discussion group to share data safely in a closed environment without the constraint of both time and space, several steps must be satisfied. This section provides a brief explanation of the required steps and outlines necessary software and hardware requirements.

4.4.1 Web Access

To overcome time and space restrictions, there must be a common platform where users of a program may gain access in their own time and space, such as that provided by the Internet. In recent years, access to the Internet has increased significantly. One reason for this has been the use of the World-Wide-Web browser made popular by Netscape Corporation. The program takes advantage of the popularity of this browser, thus it is built for the following configuration:

- Pentium 100 MHz personal computer
- Microsoft Windows 95 as the operating system
- Internet access
- Netscape Navigator Gold version 3.0
- Visual Cafe version 1.5
4. RESEARCH DESIGN

At the time of this study, another popular Internet browser, Internet Explorer 3.0 developed by Microsoft, does not fully support the Java programming language, thus it is not used.

4.4.2 Intranet Access

Safe discussions may be carried out among users of the program constructed for this experiment by creating an Intranet environment. Within a closed environment, an Intranet page may be accessed, employing Internet technology, only by people granted permission.

To develop an Intranet environment, a personal computer must be set up as a web server. There are several web servers currently in the market, but one such product, FastTrack Server: mentioned in Dyson et al. (1997, p. 169), is used because it is readily availability and easy to set up. A home page is established on the server as a starting point for users to access the program. A user must access the home page, as shown in Figure 4.5.

![Open Intranet page.](image)

Figure 4.5 Open Intranet page.
4. Research Design

When an Intranet site is found, the browser automatically prompts for the user to enter a password for verification. Figure 4.6 illustrates the authentication process.

![Username and Password Required](image)

*Figure 4.6 Logging in to the web server.*

When access is granted, the user is free to explore the page and find out more about the current study. Then, the user may access the demonstration program by choosing an option designated to each user of the program, as shown in Figure 4.7.

![Accessing electronic conferencing](image)

*Figure 4.7 Accessing electronic conferencing.*
4.5 Program Interface Design

Within this section, various interfaces used in the program are introduced to form the basis for later sections. The program main interface is shown in Figure 4.8.

![Program Interface Design](image)

*Figure 4.8* Topic & discussion lists with empty entries.

Figure 4.8 shows the main interface that participants use most frequently. Most interactions take place on this screen since it is where users view current and previous discussions. It resembles a meeting room where discussions are produced, debated upon and finalised. Through topic and discussion lists, users may read discussions and, in their own time, respond to comments made by others. Table 1 explains various components of the topic and discussion lists described in Figure 4.8. The topic and discussion lists will be described in a later section. Other interfaces used in the program are included in Appendix B.
### Components Field | Explanation
--- | ---
**Topic Of Discussion** | Contains topic names or titles used to depict a topic of interest for discussion purposes.
**Discussion** | Contains discussion names or titles used to depict a response to a topic name created by users.
**Author** | Name of the participant.
**Response Date** | Date discussion name created.
**Total Participants** | Total number of people taking part in discussion.
**#Submitted Opinion** | Number of people who have submitted an opinion.
**#Not Submitted Opinion** | Number of people who have not submitted an opinion.
**#In Favour** | Number of people in favour of current discussion.
**#Against** | Number of people against current discussion.
**Discussion** | Contents of discussion.
**System Messages** | Error message when user operates system incorrectly.

### Buttons
- **Create Topic** | Create the topic names for discussion purposes.
- **Create Discussion** | Create the discussion name to voice opinion or response to comments made by others.
- **Submit Opinion** | Submit an opinion into the polling system.
- **Change Opinion** | Change previous opinion made in the polling system.

*Table 1* Main conferencing interface components.

One particular feature that distinguishes this program from other text-based electronic conferencing programs is an ability to display a range of expressions in the discussion name list, as shown in Figure 4.19. The use of facial expressions enables users to accompany their comments with a facial reaction in a similar manner to face-to-face meetings, thus providing feedback that allows users to gauge the
4. RESEARCH DESIGN

general consensus of members of the group. Various facial expressions used in the program will be examined and developed within this thesis.

To begin a new discussion, the user must first create a topic name. A new discussion name cannot be created unless a user has already created a new topic name or chosen from an existing topic name in the topic list. The topic name suggests a topic of interest that a group may wish to discuss and is constructed using a topic creation dialog box shown in Figure 4.9. The user enters a topic name in the box and presses the Create Topic button.

![Create Topic window](image)

*Figure 4.9 Topic creation box.*

Table 2 describes various components used in the topic creation box.

<table>
<thead>
<tr>
<th>Components</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic Name</td>
<td>Contains topic name created by user.</td>
</tr>
<tr>
<td>Buttons</td>
<td></td>
</tr>
<tr>
<td>Create Topic</td>
<td>Create topic name.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancel current operation.</td>
</tr>
</tbody>
</table>

*Table 2 Explanation of Topic creation box.*
Once the topic name has been created, the user may respond to a topic name by creating a discussion name using the discussion creation box in Figure 4.10. The discussion name is a title given to comments that the user wishes to make to the chosen topic name. All the required fields must be completed so that other users know who made a comment, the time it was made and the comments text. Since the program is only a prototype, a mechanism is not implemented to automatically detect the users' information when they first log on. Thus, users are asked to undertake this task.

![Figure 4.10 Discussion creation box.](image)
Table 3 describes various components used in the Discussion creation box.

<table>
<thead>
<tr>
<th>Components Fields</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic Under Discussion</td>
<td>Contains topic name or title used to depict a topic of interest.</td>
</tr>
<tr>
<td>Discussion Title</td>
<td>Contains discussion name created by user.</td>
</tr>
<tr>
<td>Response Date</td>
<td>Date discussion name created.</td>
</tr>
<tr>
<td>Author Name</td>
<td>Name of participant.</td>
</tr>
<tr>
<td>Discussion</td>
<td>Contents of discussion.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buttons</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Select an Image</td>
<td>Select a smiley image to represent emotion.</td>
</tr>
<tr>
<td>Response</td>
<td></td>
</tr>
<tr>
<td>Submit</td>
<td>Submit a discussion.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancel current operation.</td>
</tr>
</tbody>
</table>

Table 3 Explanation of Discussion creation box.

When creating a discussion name, a user may choose to accompany comments with a facial response to denote similar tone otherwise conveyed through facial expressions in a face-to-face meeting. The various facial expressions used for this program come in the form of smilies as shown in Figure 4.11. An in depth explanation of each smiley will be made in Section 4.6.2. The purpose of the Image selection box is to provide a detailed explanation of each smiley to ease the selection process. All components used in the Image selection box are explained in Table 4.
4. RESEARCH DESIGN

![Image Selection Window](image.png)

**Figure 4.11** Image selection box.

<table>
<thead>
<tr>
<th>Components Fields</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click the ImageList to</td>
<td>Prompts the user to select one of the facial expressions available on the</td>
</tr>
<tr>
<td>select the image</td>
<td>image list.</td>
</tr>
<tr>
<td>Icon's Representation</td>
<td>Explains the meaning of the selected smiley.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buttons</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>Confirms the image selection process</td>
</tr>
<tr>
<td>Cancel</td>
<td>Dismiss the current operation</td>
</tr>
</tbody>
</table>

*Table 4* Explanation of Image selection box.
4.6 Design of Components

The following section describes the basic design and implementation for each of the components used in the study.

4.6.1 Topic & Discussions lists

In order to support group discussions, the research program is designed to display and organize data in a coherent way. To achieve this, the program makes use of two lists. The first is called a topic list and the second is a discussion list.

4.6.1.1 Topic list

A topic name describes a topic of interest for a group to discuss. A topic list may contain many topic names, which can be created by any users in a discussion group. It is the responsibility of each user to create a meaningful topic name that other users can understand. The most recently created topic name will be at the top of the topic list.
4. Research Design

4.6.1.2 Discussion list

The second list that is implemented in the program is a discussion list. A discussion name describes a response to a topic name. As in the case of topic names, a discussion name must be meaningful to other users. The most recently created discussion name will be at the top of the discussion list.

4.6.2 Smilies

This study examines users interacting in a newsgroup type system using smilies to represent different facial expressions so as to compensate for a loss of visual feedback that is present in face-to-face meetings. Harper et al. (1978, p. 77) reported that numerous studies have suggested that most human behaviour is culture-bound except for facial expressions which may be an inborn characteristic of humans, asserting that there are two primary reasons why facial expressions are an important nonverbal channel. They are as follows:

1) The amount of information that facial expression can convey in a short period of time.
2) The emotional and attitudinal behaviour that is conveyed through facial expressions.
According to Terzopoulous & Waters (1993, p. 570), a human face is capable of generating more than “55000 distinguishable facial expressions with about 30 semantic distinctions . . . [which is a result of] a confluence of voluntary muscle articulations that deform the neutral face into an expressive face.” However, identifying and incorporating such a massive number of facial expressions is beyond the scope of this study. Several authors have identified a number of primary facial expressions that can easily be represented within a system. Among these authors, Ekman (cited in Terzopoulous & Waters, 1993, p. 570) claims that there are six basic expressions that may be identified as being universally recognisable in most cultures. The six facial expressions are anger, disgust, fear, happiness, sadness and surprise. Many authors such as Tatsuno et al. (1996), Kobayashi, Hara, Ikeda and Yamada (1993) and Koda & Maes (1996) have all used these six basic facial expressions as the basis for their respective studies. However, not all expressions are deemed appropriate for a study of this type, thus, only anger, happiness and sadness emotions are included. Nevertheless, participants were asked at the conclusion of the experiment whether the study would be better served if the disgust, fear and surprise expressions were included and the outcomes are discussed in the next chapter.

The smilies implemented in this study have been designed to be clear and simple to ensure that the smilies are instantly recognisable by users without assistance. Although the smilies are obvious when identified individually, users
may be unable to instantly tell them apart if they are displayed close together. Hence, colours were introduced so that individual smilies are obvious within a range of emotions.

There are many uses for colours and Salomon (1990, p. 271) suggests that "we can create interfaces where colour either provides the user with information not available otherwise, or where colour redundantly reinforces information imparted through another medium, such as text to shape." However, Salomon (1990, p. 277) stresses that incorrect use of colour may cause false interpretation. Therefore, different colours must be carefully applied to the appropriate smilies in order to impart specific meanings to users. The section below contains a closer examination of smilies that have been designed for the program.

4.6.2.1 Overjoyed Smiley

The first expression is an overjoyed smiley which, as its name suggests, represents an emotional feeling of overwhelming joy in response to a particular event or issue raised. The user may choose this smiley to indicate agreement with current discussion. It is a sign of absolute approval with the issue being raised. To emphasise the optimistic point of view portrayed
through this smiley, a bright yellow colour is used. An overjoyed smiley is shown in Figure 4.12.

![Overjoyed Smile](image)

*Figure 4.12 Overjoyed Smiley.*

4.6.2.2 Smiling Smile

The second expression is the *smiling* smiley. The smiling smiley indicates an emotion of pleasure in response to a particular event or issue raised. Choosing this smiley indicates that the user is in favour of the current discussion. It differs from an overjoyed smiley, which represents a sign of absolute approval, whereas a smiling smiley indicates only general agreement. Since the purpose of this smiley is also to portray positive thoughts, a dark yellow colour is used to help distinguish an overjoyed smiley from a smiling smiley. A smiling smiley is shown in Figure 4.13.

![Smiling Smile](image)

*Figure 4.13 Smiling Smiley.*
4.6.2.3 Neutral Smiley

The third of these expressions is a neutral smiley, which is used to indicate that the user is not supporting or assisting discussion and not taking side in a disagreement. This smiley calls for a fairly neutral colour such as beige. A neutral smiley is shown in Figure 4.14.

![Neutral Smiley](image)

*Figure 4.14 Neutral Smiley.*

4.6.2.4 Sad Smiley

The fourth expression is the sad smiley, which represents a disappointed or unhappy response. Choosing this smiley suggests that one is upset or unhappy with comments made or a particular issue raised by members of the group. This smiley indicates that the user is not in favour or disapproves of current discussion. To emphasise the pessimistic point of view portrayed through this smiley, a light red colour is used. The sad smiley is shown in Figure 4.15.

![Sad Smiley](image)

*Figure 4.15 Sad Smiley.*
4.6.2.5 Angry Smiley

The *angry* smiley is the fifth of the expressions used in the study, which indicates a strong feeling caused by extreme displeasure in response to a particular event or issue raised. Choosing this smiley suggests that a user is deeply upset with current discussion or comments made by others. To portray a more aggressive and unhappy mood, the angry smiley is coloured red. This helps to distinguish the sad smiley from the angry smiley. Figure 4.16 shows a picture of an angry smiley.

![Angry Smiley](image)

*Figure 4.16 Angry Smiley.*

4.6.2.6 Boring Smiley

The sixth smiley representing one of the facial expression is the *boring* smiley, which suggests to other members of the group that the user is disinterested with the topic or comments made under discussion. This is a clear indication that participants wish to change the topic of discussion. A
bright Gray colour is used for this smiley because it complements the mood portrayed by the smiley. A boring smiley is shown in Figure 4.17.

![Boring Smiley](image1)

*Figure 4.17 Boring Smiley.*

4.6.2.7 NoPict/Blank Smiley

The seventh and the last expression used in the study is the **nopict** or **blank** smiley. The blank smiley is used when the user does not wish to express any thoughts about comments made by others. A dark Gray colour is used for this smiley and the result is shown in Figure 4.18.

![Blank Smiley](image2)

*Figure 4.18 Blank Smiley.*
4.6.3 Colour Usage

The previous section suggests that the use of colours will help make the smilies more obvious to a user, which will be demonstrated in this section by using examples. We hypothesise that the usage of colours will enhance the meaning of the smilies, making them easily identifiable. Some examination during the design phase ensures that research outcomes will be satisfactory.

In Figure 4.19, smilies with the same colours were used. Since all the smilies use the same yellow colour, it is difficult for users to instantly tell them apart. This becomes a problem when a user needs to express natural emotion.

![Figure 4.19 Using smilies with same colours.](image-url)
On the other hand, by introducing different colours to each of the emotions, smilies become unique. This makes smilies more identifiable and easy for users to differentiate. Figure 4.20 shows discussion names with different coloured smilies for one topic name. Therefore, the task of gauging the overall reaction of a particular topic name becomes much easier when different colours are used for each smiley. The result is clearly shown in Figure 4.20.

![Figure 4.20 Using smilies with different colours.](image)

4.6.4 Polling Concept

At this point, the design of the electronic conferencing system provides users with an interface enabling them to submit their discussions and provide graphical responses at the same time. Participants have the freedom to follow up on the discussions and decide for themselves if they agree with various arguments placed into the discussion. However, there is no means for users to demonstrate by ‘a
quick show of hands’ whether they agree or disagree about a particular discussion. Such a function may be beneficial, indeed Forum, an application developed by Isaacs, Morris, Rodriguez & Tang (1995), offers such a capability.

Forum is a video-based electronic conferencing system that broadcasts live video, audio, and slides from a speaker to a remote audience. All interactions between the two parties take place at the same time but at different places. One feature of the Forum program that pertains to this study is an implementation of a live polling system. Forum employs a poll meter to enable a 'speaker' to prompt a response from an audience about a particular query. All questions posted by the speaker are multiple-choice type questions with two choices available, namely yes and no. Such interaction between the speaker and an audience is done anonymously and the poll meter is updated in real-time when the audience makes a selection. Figure 4.21 shows an implementation of the poll meter (circled).

![Figure 4.21 Implementation of a polling meter.](image-url)
The purpose of a polling meter in the Forum application is to provide both the speaker and audience with feedback requested in a discussion. The principle behind the polling system follows closely the voting requirements suggested by Dahl (cited in Harrop & Miller, 1987, p. 5). According to the author, several criteria must be met during each phrase of the voting process for an election to function effectively in an ideal, perfectly democratic society. They include:

Before the voting period

1. All electors can insert their preferred choice among scheduled alternatives.

2. All electors have the same information about choices.

During the voting period

3. All electors vote.

4. All votes carry the same weight.

5. The choice with most votes wins.

After the voting period

6. Winning choices are implemented.

7. All other decisions are subordinate to those arrived at by voting.
The design of the polling system for this study uses a set of criteria suggested by Dahl as a guideline. Additionally, several useful features are drawn from the polling meter employed in the Forum program. A set of figures is used to indicate the number of people who agree or disagree with the topic under discussion.

This study makes use of a set of figures similar to that provided by the Forum program and includes the use of colours as suggested by Salomon (1990, p. 271). The result of the poll is displayed in the main interface using two boxes, the #In Favour and the #Against boxes, providing users with an instant feedback of the poll. This enables a user to obtain an immediate illustration of how well the discussion is doing.

The polling system utilises green for the #In Favour and red for the #Against boxes. The green colour is employed because it a fairly light colour, thereby suggesting that a user is in agreement. Salomon (1990, p. 271) asserts that “most agree that red elicits the strongest reaction from individuals. By implementing the red colour with the #Against box, a clear indication of disapproval can be drawn and made obvious to the user.”

Another criteria of Dahl’s (cited in Harrop & Miller, p. 5) approach states that ‘The choice with most votes wins’. Therefore, when the number of people in favour is greater than the number of people who are against the discussion, the #In Favour box lights up with a green colour as shown in Figure 4.22.
4. RESEARCH DESIGN

Figure 4.22 Number of people in favour is GREATER than those against.

On the other hand, when the number of people in favour is lower than the number of people who are against the discussion, the #Against box lights up with a red colour as shown in Figure 4.23.

Figure 4.23 Number of people in favour is LOWER than those against.

In circumstances where the number of people in favour is equal to the number of people who are against the discussion, both the #In Favour and #Against boxes will both light up with their respective colours as shown in Figure 4.24.

Figure 4.24 Number of people in favour is EQUAL to those against.
4. RESEARCH DESIGN

The actual activities involved in selecting a choice for the poll is done using a Submit opinion box [Figure 4.25] and a Change opinion box [Figure 4.26].

---

**Figure 4.25** Submit opinion box.

---

**Figure 4.26** Change opinion box.
4.6.5 Client/Server Program Relationship

The conventional type of interaction between a user and a static web page is a simple one. When a user accesses a web page, information is retrieved from a server in which the web page resides and data is displayed within a user's web browser. The program implemented in this study follows the same general concept, however there is a constant interaction between the program loaded by the user and the program that resides in the server. The program loaded by a user is termed a client program and the program that resides in a server is called a server program. This form of interaction is known as a client/server relationship.

Using the Java programming language, several clients programs may be running but only one server program may be running at any one time. The client program requests data and the server retrieves the relevant data and passes it to the client for display. In the next section, program design is examined in detail.

4.6.6 Client Program Design

This section contains brief explanation of the classes used in the client program. All the source code for each class used to implement the client program is supplied in Appendix B of this document.
4. Research Design

The client program, mainly responsible for the interactions between program and user and generating the main interface, is shown in Figure 4.8. It uses fourteen classes, each of which are explained in the following sections.

4.6.6.1 NewsGrpIconVote Class

Implement the main interface. Organize all incoming topic and discussion names and display them.

4.6.6.2 ImgBox Class

Implement the image selection box. Provide the interface for users to select seven smilies. Return the name of a selected smiley when requested. Track each selected smiley.

4.6.6.3 Create_Topic4 Class

Provide the topic creation box interface. Handle all interactions to create a topic name. Verify that a topic name is in the correct format. Send a topic name to the server program.
4. RESEARCH DESIGN

4.6.4 Create_Replpy4 Class

Create the discussion creation box. Handle all data entered by a user. Mark and store data for subsequent presentation. Detect instances when a user fails to select a smiley.

4.6.5 MyDate Class

Create the required date format. Return time, date and day information when requested.

4.6.6 VoteWin2 Class

Implement polling system and initiate Submit opinion box. Handle all opinion poll data and undertake calculations. Store user’s previous opinion

4.6.7 Change_OpinionBox Class

Provide VoteWin2 class with users’ previous opinion.

4.6.8 AboutBox Class

Provide information to user when Help/About selected.
4. RESEARCH DESIGN

4.6.6.9 QuitBox Class

Prompt user to confirm request to exit the program.

4.6.6.10 Topic_Cancel Class

Display a message to inform a user that a new topic has not been created when Cancel is selected.

4.6.6.11 Reply_Cancel Class

Display a message to inform a user that a new discussion has not been created when Cancel is selected.

4.6.6.12 TopicMsg Class

Display an error message prompting a user to enter a topic name on blank entry.

4.6.6.13 ReplyMsg Class

Display an error message prompting a user discussion name on blank entry.
4. RESEARCH DESIGN

4.6.6.14 ErrorMsg Class

Display an error message when Agree or Disagree radio buttons not selected.

4.6.7 Server Program Design

The server program is responsible for incoming data from a client program. It processes data from a client program and returns requested information. The server program resides only within the web server. Placing the server program in the designated directory recommended by the web server enforces security because users cannot access and corrupt files residing in other directories within the hard disk. All the source code for each classes used to implement the server program is supplied in Appendix C of this document.

4.6.7.1 InitProgramFiles Class

Server program creates and initialises system files. Set environment so that it can process and send requested data back to client program. In order to handle multiple client programs, a Producer/Consumer relationship type program is implemented within the server program.
4. RESEARCH DESIGN

The concept behind the Producer/Consumer relationship is that a **producer** produces required data and a **consumer** consumes generated data. A consumer must finish consuming generated data before a producer produces new data. Care must be taken to prevent the producer from overwriting old data before the consumer has consumed it. Additionally, the consumer must not consume data before the producer produces the next data.

4.6.7.2 *SvrIconVote1 Class*

Establish and handle all connection between server and client programs. Implement the Producer/Consumer model.

4.6.7.3 *StoreClientData Class*

This is the producer class - when the *SvrNoIconNoVote1* class detects an incoming call from a client, it sends requested data to the *StoreClientData* class. Upon receiving this data, the *StoreClientData* class writes the data to a *shared object* within the *HoldData* class that can be accessed by both the producer and consumer classes.
4. RESEARCH DESIGN

4.6.7.4 ExecuteIConvote Class

This is the consumer class which uses data in the shared object in the HoldData class. When the StoreClientData class is accessing the shared object, the ExecuteIconvote class is forbidden from accessing the shared object to ensure that neither class may corrupt data while the other is accessing it.

4.6.7.5 HoldData Class

Contain shared object for transfer duties. All methods in this class are synchronised methods – that is, the action of the previous object determines the action of the next. Only when the producer produces new data can the consumer consume it, otherwise the consumer has to wait to be notified by the producer.

4.7 Experimental Procedure

Experimental outcomes are obtained from eighteen participants, each with a completed ethical release form. In order to encourage lively electronic discussion, the experiment consists of participants formed into four groups. Ten subjects are in their twenties, two in their thirties, three in their forties and three in their fifties. All
4. Research Design

of the participants are experienced in the use of an Internet browser. All of the initial experimental activities take place by accessing the main page of the conferencing web site, which resides in an Intranet environment. The main page contains detailed instructions and explanations, enabling subjects to learn about the whole experiment before agreeing to take part in the experiment. At the commencement of the experiment, each participant completes a questionnaire, then usernames and passwords are supplied and each group member is allocated a terminal. All the functions of the program are thoroughly explained to the subjects at the start of the experiment.

Once subjects are ready, an electronic conferencing Java applet is downloaded to each terminal. Then, a moderator initiates the first topic and subjects respond or create a new discussion. An identical questionnaire is completed by participants at the conclusion of the experiment, followed by a short interview.

4.8 Testing Criteria

Table 5 shows test procedures that were developed to test the functionality and accuracy of the program.
<table>
<thead>
<tr>
<th>Test Number</th>
<th>Procedures</th>
<th>Expected Outcomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initiate discussion by creating a topic name called ‘Budget For 1998’</td>
<td>The topic name ‘Budget For 1998’ is displayed in the topic list.</td>
</tr>
<tr>
<td>2</td>
<td>Verify that users cannot submit an opinion if a discussion name is not</td>
<td>Both the Submit Opinion and Change Opinion buttons are disabled.</td>
</tr>
<tr>
<td></td>
<td>created.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Open discussion creation box and press OK without entering a discussion</td>
<td>An error message is displayed and user is prompted to enter discussion name.</td>
</tr>
<tr>
<td></td>
<td>name.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Create a discussion name called ‘Give Computer Department priority’, enter</td>
<td>The discussion name ‘Give Computer Department priority’ is displayed in discussion</td>
</tr>
<tr>
<td></td>
<td>text response press OK button.</td>
<td>list with a NoPict/Blank smiley.</td>
</tr>
<tr>
<td>5</td>
<td>Create a discussion name called ‘How much money are we talking about?’,</td>
<td>Image selection box is displayed.</td>
</tr>
<tr>
<td></td>
<td>enter text response press image selection button.</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Select ‘Neutral’ smiley and press OK button.</td>
<td>A ‘Neutral’ smiley appears in discussion box together with response previously</td>
</tr>
<tr>
<td></td>
<td></td>
<td>entered by the user.</td>
</tr>
<tr>
<td>5.2</td>
<td>When discussion box is displayed, press the OK button.</td>
<td>The discussion name ‘How much money are we talking about?’ is displayed in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>discussion list.</td>
</tr>
<tr>
<td>6</td>
<td>Submit an opinion by pressing Submit Opinion button.</td>
<td>Opinion box is displayed.</td>
</tr>
<tr>
<td>6.1</td>
<td>Select Disagree radio button and press OK.</td>
<td>When the ‘Give Computer Department Priority’ discussion name is selected, the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Against box turns red with a ‘1’ displayed.</td>
</tr>
<tr>
<td>7</td>
<td>Select ‘Give Computer Department Priority’ discussion name and press</td>
<td>A change opinion box is displayed.</td>
</tr>
<tr>
<td></td>
<td>Change Opinion button</td>
<td></td>
</tr>
<tr>
<td>7.1</td>
<td>Select Disagree radio button as the previous opinion.</td>
<td>The Disagree radio button is selected in change opinion box.</td>
</tr>
<tr>
<td>7.2</td>
<td>Select Agree radio button to change the opinion and press OK.</td>
<td>When the ‘Give Computer Department Priority’ discussion name is selected, the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In Favour box turns green with a ‘1’ displayed.</td>
</tr>
<tr>
<td>8</td>
<td>For discussion name ‘Give Computer Department Priority’, submit three</td>
<td>The main interface is displayed with the In Favour box showing green colour and</td>
</tr>
<tr>
<td></td>
<td>opinions, one in favour and two against.</td>
<td>the number ‘2’. The Against box also shows the number ‘2’ and turns red.</td>
</tr>
</tbody>
</table>

Table 5 Test criteria.
4.9 Testing the Program

The objective of this section is to satisfy testing requirements and demonstrate that the design is both effective and reliable. The section will illustrate, with a scenario and with diagrams, the steps required for a user to initiate and undertake a discussion.

4.9.1 Scenario

A single member of a group wishes to initiate a discussion about a budget allocation for the coming year. A topic name is created called ‘Budget For 1998’ by choosing File from the pull-down menu and selecting Create Topic (or clicking the Create Topic button) to load the topic creation box. The topic name is entered as shown in Figure 4.27.

![Create Topic window](image)

*Figure 4.27 Entering a topic name.*

When the Create Topic button in the Topic creation box is selected, the main interface is updated with the new topic name as illustrated in Figure 4.28. The
Submit Opinion and Change Opinion buttons are disabled.

![Figure 4.28](image) A newly created topic name.

If the user wants to be the first to start an ongoing discussion, comments are entered and Submit is selected. If a discussion name is not entered in the Discussion Title field, an error message is displayed and the user is prompted to enter a discussion name, as shown in Figure 4.29.

![Figure 4.29](image) Error message prompting for a discussion name.
Figure 4.30 shows a discussion name correctly entered.

![Discussion Name Correctly Entered](image)

**Figure 4.30 Entering discussion.**

A user may choose **Select an Image Response** to display the Image Selection Box. If a user selects the **Submit** button without selecting an image response, the newly created discussion name is associated with a blank smiley. The result of this action is shown in Figure 4.31.

![Newly Created Discussion Name and NoPict/Blank Smile](image)

**Figure 4.31 A newly created discussion name and NoPict/Blank smiley.**
4. Research Design

Figure 4.32 shows the criteria of another discussion.

Figure 4.33 shows selection of a neutral smiley.

Figure 4.32 Creating a new discussion.

Figure 4.33 Selecting Neutral Smiley.
4. RESEARCH DESIGN

Figure 4.34 shows the output following selection of a neutral smiley.

Once discussion has been entered and an image response finalised, a user may select Submit and the main interface will be automatically updated with the new discussion name. As shown in Figure 4.35, the new discussion topic ‘How much money are we talking about?’ is displayed at the top of the discussion list.
Other users of the program may introduce their own discussion and submit their opinion by selecting Submit Opinion. Disagreement with discussion is achieved by selecting Disagree, as shown Figure 4.36.

![Submit opinion box.](image)

**Figure 4.36** Submit opinion box.

Such an action results in the #Against box changing to red and the contents changing to the number ‘1’, as shown in Figure 4.37.

![One user against an idea.](image)

**Figure 4.37** One user against an idea.
A user may decide to change a previous opinion to In Favour by selecting Change Opinion. For this experiment, the user must be honest and acknowledge the previous opinion by selecting the Disagree radio button in the change opinion box, as shown in Figure 4.38.

![Validate Previous Opinion Window](image)

*Figure 4.38* Submit a previous opinion.

In this case, after the previous opinion has been confirmed, the Submit opinion box is displayed again, allowing a user to change the previous opinion, as shown in Figure 4.39.

![Polling Window](image)

*Figure 4.39* Change previous opinion.
Such an action results in the #In Favour box turning green and the #Against box changing from red to gray, providing users with instant graphical feedback that the group is in agreement. Figure 4.40 illustrates such a change of event.

![Figure 4.40 Changed opinion.](image)

In a case where the same number of users are in favour and against the discussion trend, both the #In Favour and #Against boxes will light up with their respective colours, providing feedback that opinion is evenly divided, as shown in Figure 4.41.

![Figure 4.41 Same number of people In Favour and Against an issue.](image)
### 4.9.2 Testing Outcomes

Test outcomes that show correct operation are shown in Table 6 as follows:

<table>
<thead>
<tr>
<th>Test</th>
<th>Procedures</th>
<th>Expected Outcomes</th>
<th>Verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Verify that a discussion name cannot be created without first creating a topic name.</td>
<td>An error message is displayed and the Create Discussion button is disabled.</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>Verify that the users cannot voice their opinion in the opinion poll if a discussion name is not created.</td>
<td>Both the Submit Opinion and Change Opinion buttons are disabled.</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>Initiate the discussion by creating a topic name called ‘Budget For 1998’.</td>
<td>The topic name ‘Budget For 1998’ is displayed in the topic list.</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>Open the discussion creation box and press OK without entering a discussion name.</td>
<td>An error message is displayed and the user is prompted to enter the discussion name.</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>Create a discussion name called ‘Give Computer Department priority’, enter the text response press OK button.</td>
<td>The discussion name ‘Give Computer Department priority’ is displayed in the discussion list with a NoPic/Blank smiley</td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>Create a discussion name called ‘How much money are we talking about?’, enter the text response press the image selection button.</td>
<td>The image selection box is displayed.</td>
<td>✓</td>
</tr>
<tr>
<td>6.1</td>
<td>Select the ‘Neutral’ smiley and press the OK button.</td>
<td>The ‘Neutral’ smiley appear in the discussion box along with the response previously entered by the user.</td>
<td>✓</td>
</tr>
<tr>
<td>6.2</td>
<td>Once the discussion box is displayed, press the OK button.</td>
<td>The discussion name ‘How much money are we talking about?’ is displayed in the discussion list.</td>
<td>✓</td>
</tr>
<tr>
<td>7</td>
<td>Submit an opinion by pressing the Submit Opinion button.</td>
<td>The opinion box is displayed.</td>
<td>✓</td>
</tr>
<tr>
<td>7.1</td>
<td>Select the Disagree radio button and press OK.</td>
<td>When the ‘Give Computer Department Priority’ discussion name is selected, the Against box turned into a red coloured box with the number ‘1’ displayed in it.</td>
<td>✓</td>
</tr>
<tr>
<td>8</td>
<td>Select the ‘Give Computer Department Priority’ discussion name and press the Change Opinion button</td>
<td>The change opinion box is displayed.</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>8.1</td>
<td>Select the Disagree radio button as the previous opinion.</td>
<td>The Disagree radio button is selected in the change opinion box.</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>Select the Agree radio button to change the opinion and press OK.</td>
<td>When the 'Give Computer Department Priority' discussion name is selected, the In Favour box turned into a green coloured box with the number '1' displayed in it.</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Under the same discussion name 'Give Computer Department Priority', submit three more opinions, one in favour and two against.</td>
<td>The main interface is displayed with the In Favour box showing green colour and the number '2'. The Against box also shows the number '2' and turned into a red coloured box.</td>
<td></td>
</tr>
</tbody>
</table>
5. **EXPERIMENTAL OUTCOMES**

The electronic conferencing system has been developed specifically in Java to allow the program to be deployed effectively via the Internet and Intranet. This enables the author to demonstrate the usefulness and potential of the developed methods through different approaches employed by this study. The following section presents the outcomes of the implementation undertaken by this study.

5.1 Users' Reactions to Smilies

From the interviews, it appears that no participant experiences any problem in identifying each smiley, indeed one member commented that "the smilies are clear and obvious."

Participants are asked to comment on the appropriateness of the smilies with regards to its use as a representation of different facial expressions. All agree that smilies are used appropriately in the electronic conferencing program and most participants think that smilies are simple, non-intrusive and serve their intended purpose. Others think that the smilies are clear and interesting.

Users' opinions on blank smilies are divided. Of eighteen participants, twelve people are either confused or unsure about the blank smiley. Three people use the blank smiley instead of the neutral smiley. As one user explains, "I normally choose the blank
5. EXPERIMENTAL OUTCOMES

smiley because I am trying to make a point, not necessarily feeling happy or sad.” On the other hand, five people use the neutral smiley instead of the blank smiley. As one user remarks, “I find that the blank smiley is annoying because it means that a person does not want to share emotions with others.” Apart from the dislike of a blank and neutral smiley, the use of smilies appear to be considered clear, simple and non-intrusive.

5.2 Outcomes for Research Questions

The following section restates the research questions posed in chapter two of this document, addressing each with a detailed discussion of the outcomes.

5.2.1 Research Question 1

a) Will a single presentation of facial expressions make electronic conferencing more acceptable?

In their study on the effects of personification, Koda & Maes (1996, p. 6) assert that “having a face is considered to be more likeable, engaging and comfortable” and hypothesise that having a face may be useful in educational and training applications. This study affirms that belief with the following experimental outcomes.
Among the participants who took part in the study, sixteen people agree that having a face is considered more likeable, engaging and comfortable to see in an electronic conferencing environment. One participant feels that the comments do not help since the text alone does not give the true interpretation of what the author intends to portray. This conforms to suggestions made by Sproull & Kiesler (1993, p. 69) who assert that when "compared with face-to-face discussion, people communicating electronically have more trouble imagining what others are feeling. It is hard to tell how confident others are and whether they are ready to come to final agreement."

This study indicates that facial expression helps to provide a illustrative feedback that reveals what others are thinking or feeling. One participant confirms this belief when he states that "in a normal face-to-face meeting, sometimes people say something and mean something else and at other times, you just don't know what they mean. The smilies provide graphical feedback which helps convey the message to others."

Most participants agree that the presence of smilies makes them more aware of how the rest of the group members are feeling. They view smilies as a form of illustrative feedback that provides an indication of how the group is responding to a particular message. As one participant puts it, "facial expressions give me an idea of the various reactions of other people in the group. It also gives me an idea of the general consensus or the lack of it."
Several people believe that being able to gauge the general feeling of a group not only influences their future responses but also allows them to prepare for the next comment. One subject feels that "the smilies allow me to select what mood and emotions I want the rest of the group members to see."

While many view the usefulness of smilies as providing a graphical solution to text-only systems, others find that facial expressions compensate for the loss of body language that exist in a normal face-to-face meeting. As a result, one subject feels that she is interacting more with humans than a machine.

Of the two people who believe that smilies are not useful, one of the subjects claims that he does not take much notice of the smilies, therefore he does not use them as a reference to gauge how others are feeling. The other person feels the same way and points out that "the facial expressions do not necessarily portray what I feel when submitting the discussion. Furthermore, the smilies do not mean much to me so I ignore them since they are not at all helpful."

Perhaps two people do not find smilies useful because of the small set of facial expressions available to the program during the experiment? Of the eighteen people who participated, thirteen people suggest that the smilies are useful, although more limiting than real faces.

The majority of users support the idea of being able to create their own smilies. Twelve people like the idea of being able to create smilies that best represent their
emotions at the time of discussion. As one participant remarks, the "current set of smilies is a bit limiting. There are no specific smilies that expressed emotion such as 'a little bit angry' or 'a little bit disappointed.'" However, he stresses that allowing users the ability to create their own set of smilies may be problematic since a "smiley created by one individual may not be understood by other people." Four people find the current set of smilies to be sufficient while the two people who do not find smilies to be useful do not think that it is a good idea to allow users the ability to create their own smilies.

5.2.1.1 Analysis of basic expressions

Ekman (cited in Terzopoulous & Waters, 1993, p. 570) asserts that a total of six basic expressions are universally identifiable in most cultures. To this end, participants are only exposed to the anger, happiness and sadness smilies and it is suggested that the disgust, fear and surprise expressions suggested by Ekman (1971) are perceived as inappropriate for use in the program. Nevertheless, subjects provided comment on whether disgust, fear and surprise expressions could be included.

<table>
<thead>
<tr>
<th>Suggested Icons</th>
<th>Should be Included</th>
<th>Should NOT be Included</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surprised</td>
<td>15 people</td>
<td>3 people</td>
</tr>
<tr>
<td>Disgust</td>
<td>10 people</td>
<td>8 people</td>
</tr>
<tr>
<td>Fear</td>
<td>8 people</td>
<td>10 people</td>
</tr>
</tbody>
</table>

*Table 7 Additional expressions.*
Table 7 shows that most participants agree that the surprise smiley should be included, as it will be useful to have such an expression on the system. On the other hand, participants are more divided on the possible inclusion of the disgust and fear smilies, apparently because of the nature of these emotions in that they generate a negative response. In contrast, the smiling and happiness smilies portray a positive response. A total of eight participants disagree with the disgust smiley because they view it as being too strong and very impolite. Similarly, the fear smiley generates a negative response. Additionally, one participant warns that an emotion such as fear is too vague and not applicable. It would appear to be inappropriate to include disgust and fear smilies.

From interviews, positive smilies are preferred to negative ones. As one participant points out, "I use a lot more of the smiling and overjoyed smilies because I prefer to portray a more positive response to others." This statement is consistent with that of another participant who suggests that Asian people like to maintain good relationship with others so they may not wish to use smilies that represent negative moods. She further suggests that negative emotions may be portrayed using the 'neutral' smiley.
5.2.2 Research Question 2

b) Is the use of colours to represent different facial expressions useful?

Seventeen people do not have problems with the colours used. Only one user feels that the use of colours for the facial expressions will not make much difference since the expressions are not helpful in the first place.

The electronic conferencing program uses primarily red and yellow colours with different shades of colour for differentiation. Although most users feel comfortable with these colours, there are some who disagree. Those who agree state that the red colour is obvious and suitable to represent an angry emotion. As one participant remarks, "the use of a red colour to represent the angry smiley is suitable because people naturally go red in the face when they get upset or embarrassed." Another user favours the different shades of red that exist in the various smileys because each smiley stands out when they are displayed in a discussion list.

Some of the participants disagree with the use of red colour for a number of reasons. The first reason is one of culture. One user claims that in the Chinese culture, red colour is used as a sign of good fortune and good luck. The second reason is due to users' interpretation of the colour itself. For instance, one participant associates red colour as a more romantic colour and another associates red with
danger. In an informal survey carried out by Wagner and Salomon (cited in Salomon, 1990, p. 273), the two authors found “red to be one of the colours most frequently used by people devising their own coding schemes. It was used for a number of different reasons, none of which was inherently tied to danger.” The study anticipates such factors early in the design phrase, thus each smiley is designed to be obvious so that colour matching will enhance the smilies and produce a recognisable set of common expressions.

Salomon (1990, p. 273) submits that “a brief user test should quickly indicate if a code contradicts common belief.” Judging from users’ response to the smilies implemented in the program, where seventeen people agree that the overall smilies are correctly applied, it appears that colours are used appropriately to represent emotions.

Having established that users have no problems dealing with the colours imposed on the smilies, the next step is to determine whether the colours are useful to users.
5. EXPERIMENTAL OUTCOMES

5.2.2.1 Colours Compared to Expressions

When users post a discussion name to a program, it is displayed on the discussion list, arranged with the coloured smilies as a sequential list with the most recently posted discussion name occupying the top of the list. This is shown in Figure 5.1.

![Figure 5.1](image)

*Figure 5.1 How the discussion list arranges the discussion names and smilies.*

Participants are asked if they notice the colours of the smilies first or the smilies' expressions. This is to determine if the colours enhance their ability to quickly identify the smilies. The results reveal that ten people notice the colour first followed by the smilies' expression. According to one user, the use of colours enhances her ability to differentiate the different smilies straight away. She explains that she does not have to study each of the smilies to clarify meanings.
However, seven people think otherwise and report that they notice the expressions first, followed by the colour of the smilies. One person says that smilies are not useful.

Nevertheless, the result also reveals that sixteen people agree that the combination of both colours and expressions are very effective and useful, with the exception of two people who insist that expressions help them more.

5.2.2.2 Comparison of Different lists

To satisfy the belief that the usage of colours indeed enhances users' graphical feedback, the users are asked to comment on the usefulness between the lists introduced in Figure 4.19 and Figure 4.20. The result indicates that 15 people prefer to see the discussion list filled with smilies of different colours, compared with smilies of the same colour. Only two people prefer the smilies to have the same colour. This confirms that the use of colour enhances the meaning of smilies by making them easily identifiable.

This study suggests that the use of colours to represent different facial expression are indeed useful and confirms that if instant visual feedback is attainable users are able to respond much more actively to discussion posted to the system.
5. Experimental Outcomes

5.2.3 Research Question 3

c) Will an ongoing poll make electronic conferencing more acceptable?

The previous two research questions probed the effectiveness of facial expressions in an electronic conferencing environment. The following question examines whether the polling system is effective in an electronic conferencing environment.

This study reveals that fifteen people find the ongoing poll to be very useful because of the feedback that it provides to users. The majority of users feel that the opinion poll allows them to constantly check on their own comments or comments made by others. It is apparent that users like to receive feedback on how a particular discussion is doing. One user offers this notion, "I would constantly check back on my own comment that was posted earlier to the system to see whether the rest of the group agreed with me. This allows me to plan for my next response."

It seems that the need for feedback by other members is an important one. One user in particular thinks that the opinion poll is good for decision-making. Others echoed the same view but added that the polling system also helps to terminate unpopular ideas so that the group can move on to more productive discussions.
Among these participants, one of the users suggests that the polling system minimises the need for every user to submit a new discussion or comment every time participants decide to vote on a particular issue. Once a discussion is posted to the system, the users need only to refer back to the discussion under consideration, saving time. Furthermore, the polling also gives an indication of the number of people reading the discussion, which is useful when determining if a group should move on to a new topic.

One aspect worthy of discussion is the users' view on the Change Opinion button. The Change Opinion button was initially designed to enable the users to change a previously submitted opinion. During the course of the discussion, users are exposed to more facts and convincing arguments, which may prompt a change of mind on issues under consideration. Hence, the system has facilities to accommodate such changes. For those who agree with the polling system, the ability to change their opinion would appear to be essential. Those who disagree with the polling system express their disapproval of the Change Opinion feature. They argue that the group members should only be allowed to submit their opinion once, one participant saying that “I strongly feel that people should not be allowed to change their original opinion after the event has taken place. If required, then a new discussion should be created for another vote.”
The outcomes presented so far support the hypothesis that an ongoing polling system makes electronic conferencing more acceptable. However, the participants make several suggestions that may be considered. The Submit opinion box introduced in Figure 4.25 is designed to allow users to select from only two options – Agree or Disagree. This is consistent with Isaacs, Morris, Rodriguez & Tang (1995) polling meter which implements a different set of options but maintains a similar set of choices – Yes or No. However, several users feel that there should be more options available for selection. The format proposed was to follow a questionnaire-type of choices such as the one shown in Figure 5.2.

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Undecided</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>()</td>
<td>()</td>
<td>()</td>
<td>()</td>
</tr>
</tbody>
</table>

*Figure 5.2 Questionnaire-type choices.*

One participant adds that besides registering an opinion, a user should also be allowed to explain the reason for their approval or disapproval. Others suggest that deadlines be implemented in the polling system to assist in decision-making, regarded as imperative when important issues need to be quickly resolved. In order to raise awareness, some people have suggested the use of flags to indicate that new opinions have been submitted.
5. Experimental Outcomes

5.2.4 Research Question 4

d) Are the various colours implemented in the polling system helpful?

Altogether, sixteen people agree that the #In Favour and #Against boxes shown in the main interface, Figure 4.8, are well represented by red and green colours respectively. Many users find colours to be appropriate and intuitive. The combination of both colours and data are informative and useful. Nevertheless, one participant voices his disapproval with regards to the #In Favour and #Against boxes.

Whenever the number of people in favour of a discussion is greater than the number of people against, the #In Favour box changes to green. If the number of people in favour of a discussion is less than the number of people against, then the #Against box changes to red. In the event where both the number of people for and against a discussion are equal, both the #In Favour and #Against boxes will light up with their respective colours. The overall response suggests that the majority of users find this concept to be useful, as users can often tell whether members of the group are in agreement by simply referring to the #In Favour and #Against boxes. One user explains that the colours capture her attention towards the poll. It appears that colours are effectively employed in both the facial expressions and the polling system to impart useful information to a user. Of the eighteen participants, ten
people claim to notice the colours first followed by the numbers in the #In Favour and #Against boxes. When asked if the combination of both colours and numbers is preferred, participants are almost unanimous with sixteen people declaring their support.

However, there are those who feel that the green and red colours should not be displayed concurrently. As one user comments, “the #In Favour and #Against boxes should not exhibit both colours simultaneously because it is misleading. The same colour such as blue should be displayed on both boxes instead.” Another user offers the same view further suggesting that one box should be used instead of two. Only a single person has problems comprehending the red colour. He confesses that he does not feel at ease with the colour representation used in the program due to his own interpretation of the colour.

In spite of the overwhelming support for the polling system, several participants feel that the polling system could have been presented differently. Most users are attracted to the idea of a bar chart, implemented in the Forum program by Isaacs, Morris, Rodriguez & Tang (1995). Additionally, colours may be incorporated within a bar chart to distinguish between the #In Favour and #Against arguments, further enhancing the clarity of the data.
5. Experimental outcomes

5.2.5 Research Question 5

c) Will the electronic conferencing program promote better discussion?

Within this study, sixteen people agree that the electronic conferencing program promotes better discussion. There are two major reasons to support this. Firstly, the users feel that the program allows them to be more frank when using the system. One user stresses that "the system promotes better discussion simply because it allows for a more open question and discussion environment. People who will most likely be shy or reserved in face-to-face meetings will be encouraged to speak out more when using an electronic conferencing system." Another user adds, "users will say things that they will not normally say in a normal face-to-face meeting."

A second reason is time. One user comments that "the system allows users to think about what they want to say before they say it. There is no pressure and the user can contribute at their own time. Under this environment, it is ideal for decision-making." This statement agrees with that of another user who finds that the system "allows users to reflect on their own point of view before submitting further comments."

Other users have suggested that the program promotes better discussion because it encourages people to show their own opinion instead of following others, as may occur in face-to-face meetings. Yet another thinks that the system will be favourable for those who are geographically separated. Only one user believes that the system
5. EXPERIMENTAL OUTCOMES

does not promote better discussion. He prefers face-to-face meetings because he believes that, in such meetings, people are more focused. He compares the electronic conferencing program to email, which is similar in the aspect of time delays during which people have to wait for the response by others. Regardless of this, he believes that geographically dispersed people would benefit from electronic conferencing.

5.2.6 Other Findings

Apart from the aforementioned support of the research questions, this study has also produced other findings that will be addressed in the following section.

5.2.6.1 Smilies

The smilies are well received by users. Many users feel that smilies are used appropriately in the program, making statements such as “the smilies are suitable for this program because the user does not have to consider much when using it. It is quick and immediate to use since the emotion portrayed by the smilies are clear and obvious.”

After opinions are gathered on employing smilies, users are asked to comment on whether the use of caricatures and real images of human facial expressions are applicable to the program. The first response is on the use of caricatures similar to the ones implemented in Koda & Maes (1996). Eight people suggest that caricatures may be confusing and distracting because “caricatures contain too
much detail. Facial expressions should be kept simple, like the smilies.” As for the remainder of the group, five people do not feel that caricatures are distracting and the other five are unsure.

The second response is on the use of real images of human facial expressions. The outcomes are more definite on this issue. Among the eighteen users, nine people find real images to be most distracting and inappropriate, one user says that “real images will not make a huge difference in terms of being able to portray my feelings to the group. The existing smilies already do the job pretty well. The real images will only be useful if one needs to know what the other looks like.” Six people support the use of real images but three others remain unsure. Although the number of participants is small, the result presented thus far maintains that smilies best represent facial expressions in an electronic conferencing environment.

Participants are also in agreement about the colours allocated to the seven smilies. Generally, everyone who took part agree that the colours are well assigned. Nevertheless, some suggestions were made, which are shown in Table 8.
<table>
<thead>
<tr>
<th>Smilies Name</th>
<th>Suggested Colour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angry</td>
<td>Green, Orange, Black</td>
</tr>
<tr>
<td>Happy</td>
<td>Blue</td>
</tr>
<tr>
<td>Sad</td>
<td>Gray, Blue</td>
</tr>
<tr>
<td>Boring</td>
<td>Gray</td>
</tr>
<tr>
<td>Neutral</td>
<td>Lighter Gray, White</td>
</tr>
<tr>
<td>NoPict/Blank</td>
<td>White</td>
</tr>
</tbody>
</table>

*Table 8* Suggested colours for smilies.

### 5.2.6.2 Terms used in the Program

The experimental program uses a number of terms, many of which relate closely to electronic conferencing.

Several users point out that the term 'discussion' is misleading because it seems to relate to both topic and discussion names. These users propose that a different terminology such as 'comment' would be more suitable. Another concern is the term 'opinion'. One particular user believes that the term 'vote' would be better served under this circumstance. Nonetheless, the term 'opinion' is adopted by the study because the polling system is designed to be an ongoing one. An opinion may vary from time to time but a vote seems to suggest a more permanent fixture of opinion.
5. Experimental outcomes

5.2.6.3 Topic and discussion names

The users largely control the syntax used in an electronic conferencing program. For example, the participants create both the topic and discussion names. Thus, users are chiefly responsible for the naming convention practiced on the system. As one user explains, "the topic as well as discussion names created by other users are up to individuals, therefore it is the users' responsibilities to make the names meaningful."

When questioned on the clarity of these names, ten people are not confused by topic and discussion names created by other users. The rest of the participants say that the names provided by others are unclear and confusing. All participants agree that great care must be exercised when creating names so that they are easily understood by everyone using the system.

5.2.6.4 People with strong literacy skills

The outcomes suggest that people with strong literacy skills will dominate electronic conferencing. A total of twelve people agree with this argument. Four people are unsure and two people do not agree. One member states that "those with poor literacy skills are at a disadvantage. People with better literacy skills will always dominate the floor, much like those in face-to-face meeting where one or two people normally dominate the discussion."
Of the twelve participants, nine suggest that this problem may be overcome by the use of facial expression. One user elaborates, "I think that facial expression will be useful. I can always rely on the smilies to let the others know how I feel about the topic even if my comments may not account for much." On the other hand, the other three people do not think that the existing smilies will be helpful. One says that "it doesn't matter what smilies you choose, people will judge you on the comments that you have submitted."

5.2.6.5 Moderator

Twelve people think that it is an excellent idea to have a moderator present during an electronic conferencing session. One major incentive, according to one user, is "to control the flow of the meeting. If a deadlock occurs, then the moderator can promote further discussion." Others also think that a moderator may be able to delete unwanted messages or discussions that are no longer actively used. The task of a moderator is not restricted to monitoring the discussion in the electronic conferencing session. One user adds that "a moderator is also good for following up on group members who have not been involved in the discussion or have stopped contributing altogether."

While some participants are in favour of employing a moderator, others disagree. Six people do not think that a moderator should be present during the
discussion. They point out that users are capable of ‘clearing the air’ for themselves. One user rationalises this argument by stating that “virtual discussion should be free since the group controls the approval of views. When users are engaged in electronic conferencing, they should be free to say what they want as long as it is within the group culture.”

5.2.7 Suggestions

During the duration of the experiment, several users have suggested additional features for inclusion in an electronic conferencing program.

5.2.7.1 Real-time chat function

This study provides a medium whereby users can contribute at their own time and space. This method generally means that users do not have to meet at specific times to conduct a discussion. Notwithstanding, these events do occur and people might be on-line at the same time, leading to a belief by some users that there is a need for provision of a real-time chat program. However, it is suggested that extensive use of such a tool will defeat the concept of meetings separated by time if users become too dependent on the real-time chat program to carry out their discussions. A more innovative way of utilising such a facility is for people to take part in informal chats. Authors such as McConnell (1994, p. 74) assert that “much learning takes place when people just ‘chat’... [since] free-flowing, informal talk
has a learning value in itself, and one which is just as important and educationally acceptable as structure, 'logical', formal talk.” Consequently, people may carry on with their normal discussions, as proposed in this study, reserving their casual talk for a real-time chat program. A real-time chat facility may also be used when users need to get quick response from group members.

5.2.7.2 Include Message-Threading

A decision was made in this study to exclude the use of message threading and implement a different technique of message handling through topic and discussion lists. However, some people were interested in the notion of message threading similar to that introduced in the newsgroup program. They view this function as necessary to organize the discussion names as well as enabling them to follow a thread of discussion that might interest them.

Summary

The experimental outcomes presented in this chapter verify the usefulness of facial expressions and the polling system in an electronic conferencing environment. Accordingly, answers were provided to research questions stated in chapter two.
6. CONCLUSION

6.1 Project Overview

The objectives for this study are firstly, to research the effectiveness of human facial expressions in the form of smilies in an electronic discussion environment and secondly, research the effectiveness of an opinion polling system.

Nunamaker JR (1997, 376-377) contends that there has been a gradual shift from a physical workplace to a virtual workplace. The author states that in a physical workplace, everyone meets up in the same physical venue and at the same time. In a virtual workplace, meetings are held using a computer as a medium and everyone meets electronically without having to travel to a common physical venue to conduct a meeting.

Current electronic conferencing programs offer much functionality, however there have been few studies into the usefulness of facial expressions in such systems. More importantly, facial expressions transformed as smilies have often been overlooked.

In response, this study implements an electronic conferencing program in an Intranet environment to evaluate possibilities for people in the same establishment to further their discussions without the constraint of both time and space. An experiment in which eighteen people took part is explained. Finally, the result of this study is
6. CONCLUSION

presented and discussed in chapter five of this document.

6.2 Project Outcome

The outcome of the study is that human basic facial expressions may be represented by smilies. Considering the colour coding and simplicity of the smilies, most participants have no difficulties understanding and applying smilies to discussions.

Despite the small data sample, overall assessment demonstrates users’ acceptance of smilies as a practical way of representing emotions in an electronic conferencing environment. The study also suggests that informed use of colours enhance the smilies, not only by making them easier to identify but also imparting beneficial information to users. This has considerable advantages, such as an ability to assess the general consensus of the group, which may influence a speaker’s direction. Additionally, this form of illustration allows users to relate more to comments made in textual form.

The study shows that users are eager to create their own set of smilies, however permitting users to manufacture their own smilies may result in smilies created by one individual not being understood by others. As a result, misunderstandings may occur if members of a group fail to grasp the meaning of the smilies.
From a users' perspective, a poll presents useful feedback to users. Within the experiments, several people utilise the poll as a tool to gather support from other members of a group while others used it for referencing. The study suggests that the polling system is useful for people who are undecided about a topic and for those who are joining a discussion group. For some, a poll undoubtedly influences their thoughts, which consequently affects their future opinion.

Improving the polling interface may further enhance usage of such a device. Numerous users support the polling idea but prefer a bar chart such as that implemented in the Forum program by Isaacs, Morris, Rodriguez & Tang (1995).

In summary, the following research hypothesis are shown to be correct: Facial expressions do indeed contribute positively to discussions in an electronic conferencing setting.

6.3 Future Research Directions

Whilst the experimental outcomes demonstrate an effective line of research, they also suggest a further research direction, as follows:
6.3.1 Number of Subjects

It is the author's opinion that a larger pool of participants will draw a more definite conclusion, although it is expected that the overall result will not change significantly.

6.3.2 Number of Smilies

A positive result from this research indicates that further experimentation with a greater range of smilies would be a useful future direction. Further, enabling users to create their own sets of smilies may be investigated. However, it is suggested that colour matching of smilies should be strictly monitored to avoid unnecessary confusion and distraction.

6.3.3 Introduce Message Threading

The outcomes described in chapter five indicate that message threading may be implemented in the program. Although topic and discussion lists regulate messages sent to the system, message threading may better accommodate situations when correspondents need to answer a specific discussion of another user.
Improving the overall design of this function may yield some additional benefits. For instance, the task of looking up related messages as well as following up on them will be greatly reduced. Coincidentally, the polling system will profit most from this modification. Currently, users of the system are authorised to poll their opinion to every discussion names on the discussion list. That may no longer be necessary if message threading is introduced. A more coherent method of polling might be to permit users to poll only main discussion names. For example, the concept currently implemented in this study is illustrated in Table 9. The main topic and discussion names are italicised and the result of the poll is denoted by the following numbers, (6 | 2), which indicates that six people are in favour and two against an argument. In the current model, users are allowed to place a vote on every discussion name. This may sometimes be redundant and unnecessary because not every discussion name requires a vote. For instance, a discussion name entitled "What a ludicrous suggestion" is merely a comment and it is unnecessary for users to offer an opinion. On the other hand, the discussion name "Should emails belong to the university?" requires some response from the rest of the group members.

<table>
<thead>
<tr>
<th>Topic Name</th>
<th>Discussion Name</th>
<th>Poll [For/Against]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Budget for 1998</td>
<td>Computer department should get priority</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>State your reasons</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>What would happen to other departments?</td>
<td>0</td>
</tr>
<tr>
<td>Emails ownership</td>
<td>Should emails belong to the university?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>What a ludicrous suggestion!</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Where is our privacy?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Guidelines must be used to draw the line.</td>
<td>8</td>
</tr>
</tbody>
</table>

*Table 9* Current implementation of the polling system without message threading.
Table 10 illustrates an implementation of a polling system with message threading. Notice that only the primary discussion names, “Computer department should get priority” and “Should emails belong to the university?” offer users a chance to participate in a poll. All related comments submitted thereafter are restricted from polling. Hence, this new format enables users to either read other users’ comments or state their own before making up their mind whether to agree or disagree. The complexity of previous design has been eliminated and replaced with a much simpler model, which may be presented graphically as previously discussed.

<table>
<thead>
<tr>
<th><strong>Topic Name</strong></th>
<th><strong>Discussion Name</strong></th>
<th><strong>Poll [For/Against]</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Budget for 1998</td>
<td>Computer department should get priority</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>State your reasons</td>
<td></td>
</tr>
<tr>
<td></td>
<td>What would happen to other departments?</td>
<td></td>
</tr>
<tr>
<td>Emails ownership</td>
<td>Should emails belong to the university?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>What a ludicrous suggestion!</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Where is our privacy?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Guidelines must be used to draw the line.</td>
<td></td>
</tr>
</tbody>
</table>

*Table 10* Improvement with message threading.

6.3.4 Increase Speed

One of the elementary issues that have to be dealt with in future research is the speed of the program. Numerous subjects mention that initiating a program requires considerable amount of time. However, once a program is loaded, time taken to operate the program reduces considerably.
This problem is not unexpected under current testing circumstances. At the beginning of this project, the Java programming language was in its infancy. Data required by the program is retrieved and written to the web server as files, a slow operation. Future research could employ better data access. This may be fulfilled with Java Database Connectivity, better known as JDBC. According to Ahlan (1996, p. 120) JDBC "offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMSs." Redesigning the current program to operate under JDBC may improve the overall execution of the program.

Summary

The need for this project arises from a lack of research into the use of facial expressions in an electronic conferencing program. In response to this need, the study explores the potential effectiveness of incorporating fundamental human facial expressions in the form of smilies into an electronic discussion forum that removes constraints of both time and space. Additionally, the usefulness of a polling system is also investigated.

A prototype demonstrates that incorporating a face is considered acceptable in an electronic conferencing environment. Although the data sample is relatively small, it appears that the research questions posed have been answered.
In conclusion, the implementation of facial expressions and a polling system are perceived as beneficial and invaluable to people who are engaging in discussions employing an electronic conferencing environment.
APPENDIX A: A GLOSSARY OF TERMS USED IN THIS DOCUMENT

**URLs**

Stands for Universal Resource Locator. The URLs is similar to the address of a house. In order to deliver a letter to a particular house, the user have to know the address to sent it to. Similarly, to access a particular known page on the Internet, users must have the Internet address of this home page.

**Home Page**

A home page is an Internet site set up on the Internet to allow people to access information. In previous years, these "pages" were mostly static, which means that they contain only text. In recent years, with the emergence of the Java programming language, these "pages" now dynamic, that means that developers can now include animations and programs that users can access and update information.
**Java**

Java is an object-oriented programming language that was developed by Sun Microsystems the company best known for its high-end UNIX workstations. Java is modeled after the C and C++ programming languages.

**Applets**

An Applet is a dynamic and interactive program that can run inside a Web page displayed by a Java-capable browser such as HotJava and Netscape 2.0.

Applets appear in a web page much in the same way as images do but unlike images, applets are more dynamic and interactive. Information viewed in other web browsers is limited to static text, illustrations, low-quality sounds and videos. With HotJava, applets can be used to create animation, and other interactive figures. Applets can also be used to invent creative areas that can respond to input from the user.

**Electronic Conferencing**

Electronic conferencing systems facilitate the process of sharing, organising and navigating information through an electronic space that serves as a common repository for contribution.
Groupware

Computer software that supports groups of people engaged in a common task and that provides an interface to a shared environment (Burns, 1995).
APPENDIX B : SOURCE CODE [CLIENT SIDE]

AboutBox.java

/ *===================================================================*
Program Name : AboutBox.java
Purpose : This program implements a dialog box that display information about the system and the author.

import java.awt.*;
class AboutBox extends Dialog {
    // Program Declaration
    Label label1;       // Contains the title of the system
    Label label2;       // Contains the name of the author
    Button OKButton;    // This is the OK button used in the dialog box.

    / *===================================================================*
    / * AboutBox() CONSTRUCTOR *
    / *===================================================================*
    / * Add all user interface components onto the dialog box *
    / *===================================================================*/
    public AboutBox( Frame parent ) {
        super (parent, "About", true);
        setResizable(false);  // Make sure that the dialog box is not resizable
        setLayout(null);      // No layout style for this dialog box
        addNotify();
        resize( insets().left + insets().right + 300, insets().top + insets().bottom + 143);

        // Initialise label1 object then add it to the dialog box by positioning the object using the reshape function.
        label1 = new Label("<< ONLINE CONFERENCING SYSTEM >>", Label.CENTER);
        add(label1);
        label1.reshape(insets().left + 12, insets().top + 13, 264, 20);

        // Initialise label2 object then add it to the dialog box by positioning the object using the reshape function.
        label2 = new Label("Programmed By : Thaveeporn Limpanyalers"
APPENDIX B: SOURCE CODE [CLIENT SIDE]

, Label.CENTER);
label2.setFont(new Font("Dialog", Font.BOLD,10));
add(label2);
label2.reshape(insets().left + 17,insets().top + 50,262,15);

// Initialise the OK button and add it to the dialog box
OKButton = new Button("OK");
add(OKButton);
OKButton.reshape(insets().left + 113,insets().top + 85,72,26);

} // AboutBox() Constructor

/*=================================================================*/
/* show() Method
/*=================================================================*/
/* Display the dialog box onto the screen */
/*=================================================================*/

public synchronized void show()
{
    Rectangle bounds = getParent().bounds();
    Rectangle abounds = bounds();
    move(bounds.x + (bounds.width - abounds.width)/ 2,
        bounds.y + (bounds.height - abounds.height)/2);
    super.show();
} // show() method

/*=================================================================*/
/* wakeUp() Method
/*=================================================================*/
/* This method notify other objects about its state. */
/*=================================================================*/

public synchronized void wakeUp()
{
    notify();
} // WakeUp() method

133
```java
/* This is the event handler of the dialog box. It detects the */
/* states of the buttons, windows and other components that is in */
/* the dialog box. */
/**=================================================================*!

public boolean handleEvent(Event event)
{
    // If the user clicked on the OK button then call the
    // clickedOKButton() method.
    if (event.id == Event.ACTION_EVENT
        && event.target == OKButton)
    {
        clickedOKButton();
        return true;
    } // if
    else

    // Otherwise, if the user decide to destroy the window, the
    // hide the current AboutBox dialog box.
    if (event.id == Event.WINDOW_DESTROY)
    {
        hide();
        return true;
    } // if

    // If the event received by handleEvent is not the appropriate
    // event, call the superclass handleEvent to send the event to
    // the appropriate event handling method.
    return super.handleEvent(event);
} // handleEvent() method

/*=================================================================*!
/* clickedOKButton() Method */
/*=================================================================*!

public void clickedOKButton()
{
    handleEvent(new Event(this, Event.WINDOW_DESTROY, null));
} // clickedOKButton() method
```
Change_OpinionBox.java

/*=================================================================
 Program Name Change_OpinionBox.java
 Purpose This class is used to verify the user's previous show of opinion. Since the user is changing their opinion, we have to know their previous opinion so that we can determine the changes that have to be made to the 'For' and 'Against' figures
=================================================================*/

import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.StringTokenizer;

public class Change_OpinionBox extends Frame
{
    // Program Declaration
    Label Instruction_Label; // User's Instruction
    CheckboxGroup PreviousOpinion; // CheckboxGroup containing // the user's opinion of // the Topic of discussion
    Checkbox Agree, // Previous opinion is AGREE
             Disagree; // Previous opinion is DISAGREE
    Button Submit_Opinion, // Changes the previous // opinion and submit a new // one
             Cancel; // Exit and save to file.
    int Total = 0, // Total number of voters
         Voted = 0, // Number of people voted
         NotVoted = 0, // Number of people who have // NOT voted
         InFavour = 0, // Number of people who are // in favoured of the // discussion
         Against = 0; // Number of people who are // NOT in favoured of the // discussion
    String Topic_Name = "", // The Topic of Discussion
                 Discussion = "", // discussion contents
                 Reply_Data = "", // Other Discussion details // Needed to update the // Reply?.txt file
                 VoteStatus = "", // Determine if user has // voted
                 Reply_Filename=""; // Contains the Reply // Filename to write data to

/*=================================================================* /
/* Change OpinionBox() CONSTRUCTOR */
/* This constructor provides the user interface by which the user */
/* can enter their vote on a chosen discussion name */
public Change_OpinionBox(String Selected_TName,
String Reply_Info,
String Reply_Content,
String Reply_File,
int TotalNumber,
int Number_Voted,
int Number_NotVoted,
int Number_InFavour,
int Number_Against,
String Vote_Status)
{
    // Contains the title of the frame
    super("Validate Previous Opinion Window");

    // Assign parameter variables to Global variable to be passed
    // on to the VoteWin2 class.
    Topic_Name = Selected_TName;
    Discussion = Reply_Info;
    Reply_Data = Reply_Content;
    Reply_Filename = Reply_File;
    Total = TotalNumber;
    Voted = Number_Voted;
    NotVoted = Number_NotVoted;
    InFavour = Number_InFavour;
    Against = Number_Against;
    VoteStatus = Vote_Status;

    // Initialise all AWT used
    Instruction_Label = new Label("Please select your previous
    opinion : ");

    PreviousOpinion = new CheckboxGroup();
    Agree = new Checkbox("Agree",PreviousOpinion
    ,false);
    Disagree = new Checkbox("Disagree",PreviousOpinion
    ,false);
    Submit_Opinion = new Button("Submit Opinion");
    Cancel = new Button("Cancel");

    // Setting the main container layout style to use the
    // BorderLayout style
    this.setLayout(new BorderLayout());

    // Initialising the primary panels to use on the main
    // container
    Panel Vote_Status_Info = new Panel(); // Uses FlowLayout
    Panel Button_Info = new Panel(); // Uses FlowLayout

    // Arrange the Voting information
    /*=================================================================*-
    /* Vote_Status_Info Panel */
    /*=================================================================*-
    /* This panel contains the radio buttons that prompt the user */
    /* to indicate whether their previous vote was 'Agree' or */
    /* 'Disagree'.
    /*=================================================================*-

APPENDIX B: SOURCE CODE [CLIENT SIDE]

// Setting the Vote_Status_Info panel layout style to use the
// FlowLayout style then add the buttons to the Vote_Status_Info
// panel

```
Vote_Status_Info.setLayout(new FlowLayout());
Vote_Status_Info.add(Agree);
Vote_Status_Info.add(Disagree);
```

/*===============================================================================
/* Button Info Panel */
/*===============================================================================
/* The purpose of the Button_Info panel is to organize the */
/* Change_OpinionBox interface using previously declared panels*/
/*===============================================================================

// Setting the Button_Info panel layout style to use the
// FlowLayout style then add the buttons to the Button_Info
// panel

```
Button_Info.setLayout(new FlowLayout());
Button_Info.add(Submit_Opinion);
Button_Info.add(Cancel);
```

this.add("North", Instruction_Label);
this.add("Center", Vote_Status_Info);
this.add("South", Button_Info);

resize(350,150);
setResizable(false);

// Set the main panel background to Cyan

this.setBackground(Color.cyan);
```
public String GetVoterStatus()
{
    // Store user's previous vote. Checked if user has selected the
    // 'Agree' radio box. If he/she has, then the users' previous
    // vote is in favour of the discussion. Otherwise, their
    // previous vote is against the discussion.
    String PreviousVote = "";
    if (Agree.getState() == true)
    {
        // User's previous vote is Agree
        PreviousVote = "Agree";
    } // if
    else
    {
        // Otherwise the previous vote is disagree
        PreviousVote = "Disagree";
    } // else
    return PreviousVote;
} // GetVoterStatus() Method
public boolean handleEvent(Event evt) {
    String InitialVote = ""; // Determine the user's opinion
    if (evt.target instanceof Button) {
        if (evt.arg.equals("Cancel")) {
            // Call this method if the user click on the
            // CANCEL button,
            // then hide and dispose of the current object.
            clickedCancelButton();
            this.hide();
            this.dispose();
            return true;
        } // if
    } else {
        // Determine if the user's previous vote is AGREE or
        // DISAGREE. Since neither the radio buttons are
        // initialised (i.e neither one is selected), we have to
        // check that the user selects at least one.
        if (evt.arg.equals("Submit Opinion")) {
            if ((Agree.getState() == false) &&
                (Disagree.getState() == false)) {
                // If the user did not select either the 'Agree'
                // or 'Disagree' buttons, display an error
                // message. Initialise an object of type
                // ErrorMsg and send the current frame object
                // because ErrorMsg is a dialog box.
                ErrorMsg Message = new ErrorMsg(this);
                Message.show();
                return true;
            } // if
        } else {
            // Otherwise, find out which radio buttons the
            // user selected and pass the information to the
            // VoteBox object so that user can vote again.
            InitialVote = GetVoterStatus();

            VoteWin2 VoteBox;
            VoteBox = new VoteWin2(Topic_Name, Reply_Data,
                                   Discussion, Reply_Filename,
                                   Total, Voted, NotVoted,
                                   InFavour, Against, VoteStatus,
APPENDIX B: SOURCE CODE [CLIENT SIDE]

InitialVote);

VoteBox.show();

// Dispose & Hide window once data is handled
this.hide();
this.dispose();

return true;

} // else
} // else
} // if

// If the user destroy the window then hide and dispose
// existing window.
if (evt.id == Event.WINDOW_DESTROY)
{
    this.hide();
    this.dispose();
    return true;
}

// If the event received by handleEvent is not the appropriate
// event, call the superclass handleEvent to send the event to
// the appropriate event handling method.
return super.handleEvent(evt);

} // handleEvent() Method

/**************************************************************************/
* clickedCancelButton() Method
**************************************************************************/
* This method is triggered when the user clicked on the CANCEL button.
**************************************************************************/

public void clickedCancelButton()
{
    // Call the NewsGroup Main Menu
    NewsGrpIconVote MyMenu = new NewsGrpIconVote();
    MyMenu.show();

    // Hide and dispose of the Change_OpinionBox
    this.hide();
    this.dispose();

} // clickedCancelButton() Method

} // Change_OpinionBox class
ErrorMsg.java

/*=================================================================
Program Name: ErrorMsg.java
Purpose: This class informs the user when the Agree and Disagree radio buttons is not selected when the user is submitting an opinion.
===================================================================*/
import java.awt.*;
class ErrorMsg extends Dialog {
    // Program Declaration
    Label label1; // Contains the error type
    Label label2; // Contains the instruction to rectify the error
    Button OKButton; // This is the OK button used in the dialog box.

    /*=================================================================* /
    /* ErrorMsg(Frame) CONSTRUCTOR */
    /*-----------------------------------------------------------------*/
    public ErrorMsg(Frame parent) {
        // Indicate the title of the dialog box and disallow the window to be resizable
        // The title of the ErrorMsg dialog box
        super(parent, "Previous Opinion not Selected Window", true);
        setResizable(false);

        // No layout style for this dialog box
        setLayout(null);
        addNotify();

        // Initialise label1 object then add it to the dialog box by positioning the object using the reshape function.
        label1 = new Label("<< AGREE/DISAGREE BUTTONS NOT SELECTED! >>", Label.CENTER);
        add(label1);
        label1.reshape(insets().left + 30, insets().top + 13, 350, 20);

        // Initialise label2 object then add it to the dialog box by positioning the object using the reshape function.
        label2 = new Label("<< AGREE/DISAGREE BUTTONS NOT SELECTED! >>", Label.CENTER);
        add(label2);
        label2.reshape(insets().left + 20, insets().top + 60, 350, 20);

        // Add all user interface components onto the dialog box
    }
}
// Initialise label2 object then add it to the dialog box by
// positioning the object using the reshape function.

label2 = new Label("PLEASE INDICATE YOUR
OPINION",Label.CENTER);
label2.setFont(new Font("Dialog",Font.BOLD,10));
add(label2);
label2.reshape(insets().left + 45,insets().top + 50,295,15);

// Initialise the OK button and add it to the dialog box

OKButton = new Button("OK");
add(OKButton);
OKButton.reshape(insets().left + 165,insets().top + 85,72,26);
}

/*=================================================================*/
/* show() Method
/* Display the dialog box onto the screen */
/*=================================================================*/

public synchronized void show()
{
    Rectangle bounds = getParent().bounds();
    Rectangle abounds = bounds();
    move(bounds.x + (bounds.width - abounds.width)/ 2,
    bounds.y + (bounds.height - abounds.height)/2);
    super.show();
}

/*=================================================================*/
/* wakeup() Method
/* This method notify other objects about its state. */
/*=================================================================*

public synchronized void wakeup()
{
    notify();
}

} // ErrorMsg() Constructor
public boolean handleEvent(Event event)
{
    // If the user clicked on the OK button then call the
    // clickedOKButton() method.
    if (event.id == Event.ACTION_EVENT &&
        event.target == OKButton)
    {
        clickedOKButton();
        return true;
    } // if
    else

    // Otherwise, if the user decide to destroy the window, then
    // hide the current AboutBox dialog box.
    if (event.id == Event.WINDOW_DESTROY)
    {
        this.hide();
        return true;
    } // if

    // If the event received by handleEvent is not the appropriate
    // event, call the superclass handleEvent to send the event to
    // the appropriate event handling method.
    return super.handleEvent(event);
} // handleEvent(event)

/**
* This method handles the OK button. Once the button is clicked,
* the newsgroup program will be displayed.
*/
public void clickedOKButton()
{
    handleEvent(new Event(this, Event.WINDOW_DESTROY, null));
} // clickedOKButton() method

/**
* This is the event handler of the dialog box. It detects the
* states of the buttons, windows and other components that is in
* the dialog box.
*/
public boolean handleEvent(Event event)
{
APPENDIX B: SOURCE CODE [CLIENT SIDE]

NoTopicMsg.java

/*==================================================================*/

Program Name : NoTopicMsg.java

Purpose : This dialog box is displayed to inform the user that there are no topic available for discussion.

==================================================================*/

import java.awt.*;

public class NoTopicMsg extends Dialog {
    // Program Declaration
    Button OK_Button;

    /*==================================================================*/
    /*
    NoTopicMsg(Frame) CONSTRUCTOR
    */
    /*
    This constructor provides the interface by which the user can interact with.
    */
    /*==================================================================*/

    public NoTopicMsg(Frame parent) {
        // The title of the NoTopicMsg dialog box
        super(parent, "No Topic Available Message", true);

        // Declare the panels to use for this frame
        Panel p1 = new Panel();
        Panel p2 = new Panel();

        // Add the objects to the panels
        p1.add(new Label("NO TOPIC CURRENTLY AVAILABLE"));
        p1.add(new Label("Select <Create Topic> menu option to create a Topic Name");
        add("Center", p1);

        p2.add(OK_Button = new Button("Ok"));
        add("South", p2);
        resize(370, 130);

        // Set the size of the Create_Reply frame
        // Disallowed the dialog box to be resized
        setResizable(false);
    } // NoTopicMsg CONSTRUCTOR

    /*==================================================================*/
} // NoTopicMsg CONSTRUCTOR

144
public synchronized void show()
{
    Rectangle bounds = getParent().bounds();
    Rectangle abounds = bounds();
    move(bounds.x + (bounds.width - abounds.width)/2,
         bounds.y + (bounds.height - abounds.height)/2);
    super.show();
} // show() method

/*=================================================================*!
/* wakeUp() Method */
/*=================================================================*!
public synchronized void wakeUp()
{
    notify();
} // wakeup() method

/*=================================================================*!
/* handle_event() Method */
/*=================================================================*!
public boolean handleEvent(Event evt)
{
    // Handles the button event
    if (evt.target instanceof Button)
    {
        if (evt.arg.equals("Ok"))
        {
            clickedOKButton();
            return true;
        } // if
    } // if

    // Handles the window event
    if (evt.id == Event.WINDOW_DESTROY)
    {
        hide();
        dispose();
        return true;
    } // if

    // If the event received by handleEvent is not the appropriate
    // event, call the superclass handleEvent to send the event to
    // the appropriate event handling method.
    return super.handleEvent(evt);
} // handleEvent

/*=================================================================*!
/* clickedOKButton() Method */
/*=================================================================*!
APPENDIX B : SOURCE CODE [CLIENT SIDE]

/*--------------------------------------*/
/* This method handles the OK button. Once the button is clicked, */
/* the newsgroup program will be displayed */
/*--------------------------------------*/

public void clickedOKButton()
{
    hide();
    dispose();
} // clickedOKButton() method

} // NoTopicMsg class

QuitBox.java

/*=================================================================#*/
Program Name : QuitBox.java
Purpose : This class prompt the user to quit the program.

import java.awt.*;
class QuitBox extends Dialog
{
    // Program Declaration
    Frame ParentFrame;
    Button yesButton;
    Button noButton;

    /*=================================================================#*/
    /* QuitBox(Frame) CONSTRUCTOR */
    /*=================================================================#*/
    /* Add all user interface components onto the dialog box */
    /*=================================================================#*/

    public QuitBox(Frame parent)
    {
        // The title of the QuitBox dialog box
        super(parent,"Quit Application?", true);
        // Disallowed user to resize the dialog box
        setResizable(false);
        this.ParentFrame = parent;
        setLayout(null);
        addNotify();
        resize(insets().left + insets().right + 257, insets().top
            + insets().bottom + 64);
        // Instantiate the AWT objects and add it to the dialog box
        yesButton = new Button("Yes");
        add(yesButton);

    }
APPENDIX B: SOURCE CODE [CLIENT SIDE]

```java
yesButton.reshape(insets().left + 68, insets().top + 10, 46, 23);

noButton = new Button("No");
add(noButton);
noButton.reshape(insets().left + 135, insets().top + 10, 47, 23);
}

public synchronized void show()
{
    Rectangle bounds = getParent().bounds();
    Rectangle abounds = bounds();
    move(bounds.x + (bounds.width - abounds.width)/ 2,
         bounds.y + (bounds.height - abounds.height)/2);
    super.show();
}

public synchronized void wakeup()
{
    notify();
}
```
public boolean action(Event event, Object arg) {
    // Handles the button event
    if (event.target instanceof Button) {
        if (event.arg.equals("No")) {
            clickedNoButton();
            return true;
        }
    } // if
    // Handles the window event and the 'Yes' button
    if ((event.id == Event.WINDOW_DESTROY) ||
        (event.id == Event.ACTION_EVENT &&
         event.target == yesButton)) {
        // Dispose and hide the Quit Dialog Box
        this.hide();
        this.dispose();
        // Dispose and hide the Parent Frame
        this.ParentFrame.hide();
        this.ParentFrame.dispose();
        // If this is run as application. System.exit(0) works. If run on Applet, you just have to
        // hide() & dispose() it
        return true;
    } // if
    // If the event received by handleEvent is not the
    // appropriate event, call the superclass handleEvent to
    // send the event to the appropriate event handling
    // method.
    return super.handleEvent(event);
} // action(Event, Object) method
APPENDIX B: SOURCE CODE [CLIENT SIDE]

/*=================================================================*/
/* clickedNoButton() Method */
/*=================================================================*/
/* This method handles the 'No' button. Once the button is clicked, */
/* the newsgroup program will be displayed again so that the users */
/* can continue their discussion. */
/*=================================================================*/

public void clickedNoButton()
{
    this.hide();
    this.dispose();
} // clickedNoButton() method

} // QuitBox() method

Reply_Cancel.java

/*===================================================================*/
Program Name : Reply_Cancel.java
Purpose : This class handles the event when the user clicked
the "Cancel" button in the Create Reply4 frame.
Once click, the class will load the parent frame
(NewsGrpIconVote) and informs user that
the discussion name has not been created.
/*===================================================================*/

import java.awt.*;
public class Reply_Cancel extends Dialog
{
    // Program Declaration
    Label Reply_Cancel_Dialog;
    Button Cancel_OK_Button;

    public Reply_Cancel(Frame parent)
    {
        // Title of the Reply_Cancel frame
        super(parent, "Reply Cancel Box", true);
        setLayout(null);
        addNotify();
        resize(insets().left + insets().right + 266, insets().top +
            insets().bottom + 180);
        // Instantiate the AWT objects and add to the frame
        Reply_Cancel_Dialog = new Label("Reply File was not
            created !!!");
        add(Reply_Cancel_Dialog);
APPENDIX B: SOURCE CODE [CLIENT SIDE]

```java
Reply_Cancel_Dialog.reshape(insets().left + 49, insets().top + 46, 168, 13);

Cancel_OK_Button = new Button("OK");
add(Cancel_OK_Button);
Cancel_OK_Button.reshape(insets().left + 77, insets().top + 85, 91, 26);

// Disallowed the user to resize the window
setResizable(false);
}
} // Reply_Cancel(Frame) CONSTRUCTOR

/*================================================================*/
/* show() Method */
/*================================================================*/
/* Display the dialog box onto the screen */
/*================================================================*/

public synchronized void show()
{
    Rectangle bounds = getParent().bounds();
    Rectangle abounds = bounds();
    move(bounds.x + (bounds.width - abounds.width)/2, 
        bounds.y + (bounds.height - abounds.height)/2);
    super.show();
} // show() method

/*================================================================*/
/* wakeup() Method */
/*================================================================*/
/* This method notify other objects about its state. */
/*================================================================*/

public synchronized void wakeup()
{
    notify();
} // wakeup() method
```
APPENDIX B: SOURCE CODE [CLIENT SIDE]

public boolean handleEvent(Event event)
{
    // Handles the button event
    if (event.id == Event.ACTION_EVENT
        && event.target == Cancel_OK_Button)
    {
        clickedCancelOKButton();
        return true;
    } // if
    else
    {
        // Handles the window event
        if (event.id == Event.WINDOW_DESTROY)
        {
            hide();
            dispose();
            return true;
        } // if
    } // if

    // If the event received by handleEvent is not the appropriate
    // event, call the superclass handleEvent to send the event to
    // the appropriate event handling method.
    return super.handleEvent(event);
} // handleEvent(Event) method
APPENDIX B : SOURCE CODE [CLIENT SIDE]

ReplyMsg.java

/*================================================================================*/
Program Name : ReplyMsg.java
Purpose : This class prompts the user to enter the discussion name again when the user fails to edit the Discussion Title field. Every discussion must have a discussion name so that they can be displayed on the discussion list.
===================================================================*/
import java.awt.*;

class ReplyMsg extends Dialog
{
    // Program Declaration
    Label label1;
    Label label2;
    Button OKButton;

    /*================================================================================*/
    /* ReplyMsg(Frame) CONSTRUCTOR */
    /*================================================================================*/
    public ReplyMsg(Frame parent)
    {
        // Title of the ReplyMsg frame
        super(parent, "Discussion Not Entered", true);
        setLayout(null);
        addNotify();
        resize(insets().left + insets().right + 300, insets().top +
                insets().bottom + 143);

        // Instantiate the AWT objects and add it to frame
        label1 = new Label("<<DISCUSSION NAME IS NOT ENTERED>>", Label.CENTER);
        add(label1);
        label1.reshape(insets().left + 12, insets().top + 13, 275, 20);

        OKButton = new Button("OK");
        add(OKButton);
        OKButton.reshape(insets().left + 113, insets().top + 85, 72, 26);

        label2 = new Label("PLEASE ENTER A DISCUSSION NAME", Label.CENTER);
        label2.setFont(new Font("Dialog", Font.BOLD, 10));
        add(label2);
        label2.reshape(insets().left + 17, insets().top + 50, 262, 15);

        // Allow the user to resize the window
        setResizable(true);
    } // ReplyMsg(Frame) CONSTRUCTOR

    /*================================================================================*/
    /* ReplyMsg(Frame) CONSTRUCTOR */
    /*================================================================================*/

} // ReplyMsg
APPENDIX B: SOURCE CODE [CLIENT SIDE]

/*=================================================================*
/* show() Method */
/*=================================================================*\npublic synchronized void show()
{ 
    Rectangle bounds = getParent().bounds();
    Rectangle abounds = bounds();
    move(bounds.x + (bounds.width - abounds.width)/ 2,
         bounds.y + (bounds.height - abounds.height)/2);
    super.show();
} // show() method

/*=================================================================*
/* wakeup() Method */
/*=================================================================*\npublic synchronized void wakeup()
{ 
    notify();
} // wakeup() method

/*=================================================================*
/* handleEvent() Method */
/*=================================================================*\npublic boolean handleEvent(Event event)
{ 
    // Handles the Button event
    if (event.id == Event.ACTION_EVENT &&
        event.target == OKButton)
    {
        clickedOKButton();
        return true;
    } // if
else

    // Handles the windows event
    if (event.id == Event.WINDOW_DESTROY)
    {
        hide();
        return true;
    } // if

    // If the event received by handleEvent is not the appropriate
APPENDIX B : SOURCE CODE [CLIENT SIDE]

// event, call the superclass handleEvent to send the event to
// the appropriate event handling method.

return super.handleEvent(event);

} // handleEvent(Event)

/*=================================================================* /
/* clickedOKButton() Method */
/*=================================================================* /
/* This method handles the OK button. Once the button is clicked, */
/* the newsgroup program will be displayed. */
/*=================================================================* /

public void clickedOKButton()
{
    handleEvent(new Event(this, Event.WINDOW_DESTROY, null));
}

} // clickedOKButton() method

} // ReplyMsg() Class

Topic_Cancel.java

/*=================================================================* /
/* Program Name : Topic_Cancel.java */
/*=================================================================* /

import java.awt.*;

public class Topic_Cancel extends Dialog
{
    // Program Declaration
    Label Topic_Cancel_Dialog;
    Button Cancel_OK_Button;

    /*------------------------------*/
    /* Topic_Cancel(Frame) CONSTRUCTOR */
    /*------------------------------*/
    /* Add all user interface components onto the dialog box */
    /*------------------------------*/

    public Topic_Cancel(Frame parent)
    {
        // Title of the Topic_Cancel frame
        super(parent, "Topic Cancel Box", true);
        setLayout(null);
        addNotify();
    }

} // Topic_Cancel.java
APPENDIX B: SOURCE CODE [CLIENT SIDE]

```java
resize(insets().left + insets().right + 266, insets().top +
insets().bottom + 180);

// Instantiate the AWT objects and add it to the frame
Topic_Cancel_Dialog = new Label("Topic File is not
created !!!");
add(Topic_Cancel_Dialog);
Topic_Cancel_Dialog.reshape(insets().left + 49,insets().top +
46,168,13);

Cancel_OK_Button = new Button("OK");
add(Cancel_OK_Button);
Cancel_OK_Button.reshape(insets().left + 77,insets().top +
85,91,26);

// Disallowed the user to resize the frame
setResizable(false);
}

// Topic_Cancel( Frame) CONSTRUCTOR

public synchronized void show()
{
    Rectangle bounds = getParent().bounds();
    Rectangle abounds = bounds();
    move(bounds.x + (bounds.width - abounds.width)/ 2,
         bounds.y + (bounds.height - abounds.height)/2);;
    super.show();
}

public synchronized void wakeUp()
{
    notify();
}

public synchronized void handleEvent()
{

```
public boolean handleEvent(Event event)
{
    // Handles the Buttons event
    if (event.id == Event.ACTION_EVENT
        && event.target == Cancel_OK_Button)
    {
        clickedCancelOKButton();
        return true;
    } // if
    else

    // Handles the windows event
    if (event.id == Event.WINDOW_DESTROY)
    {
        hide();
        dispose();
        return true;
    } // if

    // If the event received by handleEvent is not the appropriate
    // event, call the superclass handleEvent to send the event to
    // the appropriate event handling method.
    return super.handleEvent(event);
} // handleEvent(Event)

/**
 * clickedCancelOKButton() Method
 */
public void clickedCancelOKButton()
{
    hide();
    dispose();
}

} // Topic_Cancel Class
APPENDIX B : SOURCE CODE [CLIENT SIDE]

TopicMsg.java

/*===================================================================*/
Program Name : TopicMsg.java
Purpose : This class prompts the user to enter the topic name again when the user fails to enter the topic name in the Topic_Create4 frame when prompted. Every topic must have a topic name so that they can be displayed on the topic list.

===================================================================*/

import java.awt.*;
class TopicMsg extends Dialog {
    // Program Declaration
    Label label1;
    Label label2;
    Button OKButton;

    /*===================================================================*/
    /*
       TopicMsg(Frame) CONSTRUCTOR
    */
    /*===================================================================*/
    /* Add all user interface components onto the dialog box */
    /*===================================================================*/

    public TopicMsg(Frame parent) {
        // Title of the TopicMsg frame
        super(parent,"Topic Not Entered Message",true);
        setLayout(null);
        addNotify();
        resize(insets().left + insets().right + 300, insets().top + insets().bottom + 143);

        // Instantiate the AWT objects and add to frame
        label1=new Label("<< TOPIC NAME IS NOT ENTERED >>", Label.CENTER);
        add(label1);
        label1.reshape(insets().left + 12,insets().top + 13,264,20);
        OKButton=new Button("OK");
        add(OKButton);
        OKButton.reshape(insets().left + 113,insets().top + 85,72,26);
        label2=new Label("PLEASE ENTER A TOPIC NAME", Label.CENTER);
        label2.setFont(new Font("Dialog",Font.BOLD,10));
        add(label2);
        label2.reshape(insets().left + 17,insets().top + 50,262,15);
    }
APPENDIX B : SOURCE CODE [CLIENT SIDE]

// Disallowed the user to resize the frame
setResizable(false);

} // TopicMsg(Frame) CONSTRUCTOR

/**
 * show() Method
 * Display the dialog box onto the screen
 */
public synchronized void show()
{
    Rectangle bounds = getParent().bounds();
    Rectangle abounds = bounds();
    move(bounds.x + (bounds.width - abounds.width)/ 2,
         bounds.y + (bounds.height - abounds.height)/2);
    super.show();
} // show() method

/**
 * wakeup() Method
 * This method notify other objects about its state.
 */
public synchronized void wakeup()
{
    notify();
} // wakeUp() method

/**
 * handleEvent() Method
 * This is the event handler of the dialog box. It detects the
 * states of the buttons, windows and other components that is in
 * the dialog box.
 */
public boolean handleEvent(Event event)
{
    // Handles the button event
    if (event.id == Event.ACTION_EVENT
        && event.target == OKButton)
    {
        clickedOKButton();
        return true;
    } // if

    else
APPENDIX B: SOURCE CODE [CLIENT SIDE]

// Handles the windows event
if (event.id == Event.WINDOW_DESTROY)
{
    hide();
    return true;
} // if

// If the event received by handleEvent is not the appropriate event, call the superclass handleEvent to send the event to the appropriate event handling method.
return super.handleEvent(event);

} // handleEvent(Event) method

/*==============================================================================================
* clickedOKButton() Method
*/
/*==============================================================================================
* This method handles the OK button. Once the button is clicked, the newsgroup program will be displayed.
*/
/*==============================================================================================

public void clickedOKButton()
{
    handleEvent(new Event(this, Event.WINDOW_DESTROY, null));
} // clickedOKButton() method

} // TopicMsg() Class
Create_Reply4.java

/===================================================================
Program Name : Create_Reply4.java
Purpose : This class will create the discussion name and write the new incremented Counter value to the Server so that the Reply_Count.txt file can be updated.
=====================================================================

import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.StringTokenizer;

public class Create_Reply4 extends Frame
{
    // Program declaration
    // Used to store the Socket Object sent by NewsGrpIconVote Main
    // Frame
    Socket NewsGrpSocket;
    // Output the data to the server, through the socket, to update
    // necessary files
    PrintStream ToServer = null;
    // Get the data from the server through the socket
    DataInputStream FromServer = null;
    // Using port 8003 to create the discussion name.
    int NewsGrpPort = 8003;
    // Contains command to server as a string object
    String Command = ",
    Hostname = "krakatoa.fste.ac.cowan.edu.au";
    // Uses Visual Cafe to display user's selected 'Smiley' on
    // Create_Reply Frame
    ImageViewer imageViewer1;

    // The Topic name under discussion ReplyName
    TextField TopicName,
    DateBox; // Date that the discussion name
    // was created
String Topic_Name = "", // Topic name
Reply_Name = "", // Discussion name
UserDate = "", // Discussion date
UserMessage = "", // User's discussion
ImageName = "", // Name of image
    // chosen by user in // ImgBox Frame
ImageLocation = "http://krakatoa.fste.ac.cowan.edu.au/
                   Images/Thesis_Images/New_Color/";

int UName_Index = 0; // Since users' name
                    // are stored in a pop-
                    // up list, the placing
                    // of the user's name
                    // in the pop-up list
                    // has to be stored.
public Create_Reply4(Socket MySocket,
        String CommandString,
        String Topic_Name)
{
    // The title of the Create_Reply4 frame
    super("Create Discussion window");

    // Contains the client's request to the server.
    Command = CommandString;

    // Passing the NewsGrpiconVote frame socket to the current
    // frame so that socket connection with the server can be
    // established
    NewsGrpSocket = MySocket;

    // Declare the panels to use for this frame
    Panel Reply_Info = new Panel();  // Contains all the fields
    // that must be entered by
    // the user
    Panel Content_Info = new Panel();  // Contains the discussion
    // label and the discussion
    Panel My_Buttons = new Panel();  // Contains the Submit and
    // Cancel buttons.

    // Initialise all AWT used
    TopicLabel = new Label("Topic Under Discussion");
    TopicName = new TextField(50);
    ReplyLabel = new Label("Discussion Title");
    ReplyName = new TextField(50);
    UserLabel = new Label("Author Name");
    UserName = new Choice();
    UserName.addItem("Guest");
    UserName.addItem("Carmen Castillo");
    UserName.addItem("Jacinta Arokiasamy");
    UserName.addItem("Judy");
    UserName.addItem("Ming Nget");
    UserName.addItem("Mona Fong");
    UserName.addItem("Rweyunga");
    UserName.addItem("Sin Yi");
    UserName.addItem("Thaveeporn Limpanyalers");
    DateLabel = new Label("Response Date");
    DateBox = new TextField(50);
    ImgResponse_Label = new Label("Image Response");
    PickImage_Button = new Button("Select an Image Response");
ContentsLabel = new Label("Discussion");
Contents = new TextArea(15, 15);
Submit = new Button("Submit");
Cancel = new Button("Cancel");

// Organises the layout of the entire Create_Replv4 frame
this.setLayout(new BorderLayout());

// Arrange the discussion information
Reply_Info.setLayout(new GridLayout(15, 2));
Reply_Info.add(TopicLabel);
Reply_Info.add(TopicName);
TopicName.setText(Topic_Name);
TopicName.setEditable(false);

Reply_Info.add(ReplyLabel);
Reply_Info.add(ReplyName);
ReplyName.setText("Must Edit Discussion Title");

Reply_Info.add(DateLabel);
Reply_Info.add(DateBox);
DateBox.setEditable(false);

// Set the DateBox field to display the current date
MyDate date = new MyDate();
String DisplayDate = date.TodayDate();
DateBox.setText(DisplayDate);

Reply_Info.add(UserLabel);
Reply_Info.add(UserName);

Reply_Info.add(ImgResponse_Label);
Reply_Info.add(PickImage_Button);

// Arrange the discussion contents
Content_Info.setLayout(new BorderLayout());
Content_Info.add("North", ContentsLabel);
Content_Info.add("Center", Contents);

// Finally, arrange the buttons
My.Buttons.setLayout(new FlowLayout());
My.Buttons.add(Submit);
My.Buttons.add(Cancel);

this.add("North", Reply_Info);
this.add("Center", Content_Info);
this.add("South", My.Buttons);

resize(490, 450); // Set the size of the Create_Replv frame
setResizable(true); // Set the Create_Replv4 frame to be resizable.

} // Create_Replv4(Socket, String, String) CONSTRUCTOR
The second constructor is used if the user clicked on the PickImage_Button. When ever the button is clicked, the data has to be written from the Create_Reply4 frame to the ImgBox frame then back again to the Create_Reply4 so that the fields can be updated. Thus the original data entered by the user (if any) will be correctly set to each of the fields in the Create_Reply4 frame.

```java
public Create_Reply4(Socket MySocket,
                       String DataFromImgBox)
{
    // Title of the Create_Reply4 frame.
    super("Create Discussion window");

    // Passing the NewsGrpIconVote frame socket to the current
    // frame so that socket connection with the server can be
    // established
    NewsGrpSocket = MySocket;

    // Organise the data retrieved from ImgBox and set the
    // Create_Reply4 frame with the updated information after ALL
    // the AWT objects are created.
    // Note : You cannot set the fields unless they are created
    // first.
    ArrangeData(DataFromImgBox);

    // Declare panels to use in this frame
    Panel Reply_Info = new Panel();       // Contains all the fields
                                            // that must be entered by
                                            // the user
    Panel Content_Info = new Panel();     // Contains the discussion
                                            // label and the discussion
    Panel My.Buttons = new Panel();       // Contains the Submit and
                                            // Cancel buttons.

    // Initialise all the AWT used
    TopicLabel = new Label("Topic Under Discussion");
    TopicName = new TextField(50);
    ReplyLabel = new Label("Discussion Title");
    ReplyName = new TextField(50);
    UserLabel = new Label("Author Name");

    UserName = new Choice();
    UserName.addltem("Guest");
    UserName.addltem("Carmen Castillo");
    UserName.addltem("Jacinta Arokiasamy");
    UserName.addltem("Judy");
    UserName.addltem("Ming Nget");
    UserName.addltem("Mona Fong");
    UserName.addltem("Rweyunga");
```
UserName.additem("Sin Yi");
UserName.additem("Thaveeporn Limpanyalers");

UserName.additem("Sin Yi");
UserName.additem("Thaveeporn Limpanyalers");
UserName.select(UNameIndex);

DateLabel = new Label("Response Date");
DateBox = new TextField(50);
ImgResponse_Label = new Label("Image Response");
PickImage_Button = new Button("Select an Image Response");
ContentsLabel = new Label("Discussion");
Contents = new TextArea(15,15);
Submit = new Button("Submit");
Cancel = new Button("Cancel");

// Organises the layout of the entire Create_Reply frame
// this.setLayout(new BorderLayout());

// Since all the AWT are created, we may set the fields with
// the appropriate information.

Reply_Info.setLayout(new GridLayout{5,2});
Reply_Info.add(TopicLabel);
Reply_Info.add(TopicName);

// Update Topic name field with data from ImgBox frame
TopicName.setText(Topic_Name);
TopicName.setEditable(false);

Reply_Info.add(ReplyLabel);

// Update Discussion name field with data from ImgBox frame
Reply_Info.add(ReplyName);
ReplyName.setText(Reply_Name);

Reply_Info.add(DateLabel);
Reply_Info.add(DateBox);

// Update the date field with data from ImgBox frame
DateBox.setText(UserDate);
DateBox.setEditable(false);

Reply_Info.add(UserLabel);
Reply_Info.add(UserName);

Reply_Info.add(ImgResponse_Label);
Reply_Info.add(PickImage_Button);

// Arrange the 'Smiley' selected by user in ImgBox frame
imageViewer1 = new ImageViewer();
imageViewer1 reshape(50,60,150,113);
add(imageViewer1);
imageViewer1.setCenterMode(true);

// Take care of the ImgName given
// First obtain image absolute URL then display it onto
// the current frame.
String ModifiedImgName = ImageLocation + ImageName;
// Add the 'Smiley' to the Create_Reply4 frame
addImage(ModifiedImgName);
// Arrange the discussion contents
Content_Info.setLayout(new BorderLayout());
Content_Info.add("North",ContentsLabel);
Content_Info.add("Center",Contents);
Contents.setText(UserMessage);

// Finally, arrange the buttons
MyButtons.setLayout(new FlowLayout());
MyButtons.add(Submit);
MyButtons.add(imageViewer1);
MyButtons.add(Cancel);
this.add("North",Reply_Info);
this.add("Center",Content_Info);
this.add("South",MyButtons);
resize(490,450); // Set the size of the Create_Reply
// frame
setResizable(true); // Set the Create_Reply frame to be
// resizable
} // CONSTRUCTOR Create_Reply4(Socket, String)

/*****************************/
/* ArrangeData(String) CONSTRUCTOR */
/*****************************/
/*As pointed out earlier, the original data entered by user in the */
/*Create_Reply4 frame must be send to ImgBox frame and back again. */
/* */
/*This method arranges the String of data that is sent to this */
/*class from ImgBox frame. All the data are then updated to the */
/*fields in the Create_Reply frame. */
/*=================================================================*/
public void ArrangeData(String Reply_Info) {
String MyCommand = "", // Contains the client's
// request to the server.
TName = "", // Topic Name
Rname = "", // Discussion Name
DateData = "", // Discussion Date
Message = "", // Discussion contents
Modified_Reply = "", // Replace '*' & 'n' char
// with blanks for display
// purposes
APPENDIX B: SOURCE CODE [CLIENT SIDE]

```java
int MaxNumbers = 7; // Max number of items allowed in the array
UNName = 0; // Author's Name

// Declare an array of strings to store the data
String ReplyData[] = new String[MaxNumbers];

// Attempt to read the Reply_Info data string which is seperated by the '#' symbol. Use the StringTokenizer method to separated each of the required string and store them in the array. If any error occurs, notify programmer.

try {
    StringTokenizer Tokens = new StringTokenizer(Reply_Info, "#");

    for (int i=0; i<7; i++) {
        if (Tokens.hasMoreTokens()) {
            ReplyData[i] = Tokens.nextToken();
        }
    }
}

// When each string is retrieved properly, start storing the string in the array.

try {
    // Assign the appropriate data from ImgBox to correct variables
    // Contains the client's request to the server
    MyCommand = ReplyData[0];

    // Contains the TOPIC Name
    TName = ReplyData[1];

    // Contains the REPLY Name
    RName = ReplyData[2];

    // Contains the Reply date
    DateData = ReplyData[3];

    // Contains the user Name
    UNName = Integer.parseInt(ReplyData[4]);

    // Contains the Reply Message
    Message = ReplyData[5];

    // Contains the Image name chosen by user
    Image_Name = ReplyData[6];

    ...
// Assign Array items to Global Variables

Command = MyCommand;
Topic_Name = TName;
// The following code below test the scenario when the user
// clicked on the PickImage_Button in the Create_Reply4 frame
// BUT have not entered any data.
//
// The program uses the 'NO DATA' tag to symbolise that data is
// not entered. Thus, we have to replace the tag with an empty
// string so that data can be displayed properly in the
// Create_Reply4 frame.

if (RName.equals("NO DATA"))
{
    Rname = "";
    // Assign the appropriate string to global variable
    Reply_Name = RName;
} // if
else
{
    Reply_Name = RName;
} // else

// Do the same for the date data.

if (DateData.equals("NO DATA"))
{
    DateData = "";
    // Assign the appropriate string to global variable
    UserDate = DateData;
} // if
else
{
    UserDate = DateData;
} // else

// Assign the author’s index in the pop-up list to global
// variable
UName_Index = UName;

// The program uses the '*' char to represent a new line ('\n')
// when stored in the system files. Therefore, before we
// display the message entered by the user to the Create_Reply
// frame we have to replace the '* char with a new line.

if (Message.equals("NO DATA"))
{
    // Assign the appropriate string to global variable
    Message = "";
    UserMessage = Message;
} // if

else
{
    // Replacing the '*' char with a new line is done with
    // the following code.
    Modified_Reply = Message.replace('*','\n');
    // The final data that can be display on the screen
    UserMessage = Modified_Reply;
} // else

// Assign the Image Name to Global Variable
ImageName = Image_Name;

} // ArrangeData(String) method

PLICATE();

/*=============================================================*/
/* show() Method */
/*--------------------------------------------------------------*/
/*This method display the main interface for interaction with the */
/*user. Note that the move() command also set the position of the */
/*program on the screen. */
/*--------------------------------------------------------------*/

public synchronized void show()
{
    move(50, 50);
    super.show();
} // show() method

/*=============================================================*/
/* handleEvent() Method */
/*--------------------------------------------------------------*/
/*This method handles all the Button Events as well as when the */
/*window is destroyed. */
/*--------------------------------------------------------------*/

public boolean handleEvent(Event evt)
{
    // Check if it is a button event and handles the buttons
    // appropriately
    
    if (evt.target instanceof Button)
    
    return true;

    else
    
    return super.handleEvent(evt);
} // handleEvent() method
{
  if (evt.arg.equals("Cancel"))
  {
    clickedCancelButton();
    // Dispose the Create Reply frame
    this.hide();
    this.dispose();
    // Return the user to the NewsGrpIconVote frame.
    NewsGrpIconVote MyMenu = new NewsGrpIconVote();
    MyMenu.show();
    return true;
  } // if
  else
  {
    // Otherwise, if the 'Submit' button is clicked, then
    // check that the user has entered the Topic name, if it
    // is not entered, inform the user and prompt him/her to
    // enter again.
    if (evt.arg.equals("Submit"))
    {
      String Entered_RName = ReplyName.getText();
      // Checks if user enters Topic Name using most
      // likely possibilities.
      if (Entered_RName.equals("") ||
          (Entered_RName.equals("Must Edit Discussion Title") ||
           Entered_RName.equals("Must Edit Discussion") ||
           Entered_RName.equals("Must Edit Discussion ") ||
           Entered_RName.equals("Must Edit") ||
           Entered_RName.equals("Must Edit ") ||
           Entered_RName.equals("Must") ||
           Entered_RName.equals("Must ") ||
           Entered_RName.equals("Must_Edit_Discussion_Title") ||
           Entered_RName.equals("Must_Edit_Discussion") ||
           Entered_RName.equals("Must_Edit_Discussion ") ||
           Entered_RName.equals("Must_Edit") ||
           Entered_RName.equals("Must_Edit ") ||
           Entered_RName.equals("Must") ||
           Entered_RName.equals("Must ")))
      {
        ReplyMsg RplyMsg = new ReplyMsg(this);
        RplyMsg.show();
      } // if
      else
      {
        // If all data has been entered properly, connect
        // to the server and send the data
        CreateReply();
        // Dispose & Hide window once data is handled
        this.hide();
        this.dispose();
      }
    }
}
// Return the user to the NewsGrpIconVote frame.

NewsGrpIconVote MyMenu = new NewsGrpIconVote();
MyMenu.show();

} // else

return true;

} // if

else

// If the PickImage_Button is clicked, then load up the // ImgBox frame

if (evt.arg.equals("Select an Image Response"))
{
    // Dispose the Create Reply4 frame
    this.hide();
    this.dispose();

    // Load the ImgBox frame and send the data entered by // the user in the Create_Reply4 frame (if any) to // the ImgBox frame so that they can be displayed in // the Create_Reply4 frame with the Image selected.

    clickedPickImageButton();

    return true;

} // if

} // if

// If the user closes the Create_Reply4 frame, load up the // NewsGrpIconVote frame again.

if (evt.id == Event.WINDOW_DESTROY)
{
    clickedCancelButton();
    this.hide();
    this.dispose();

    // Return the user to the NewsGrpIconVote frame.

    NewsGrpIconVote MyMenu = new NewsGrpIconVote();
    MyMenu.show();

    return true;

} // if

// If the event received by handleEvent is not the appropriate // event, call the superclass handleEvent to send the event to // the appropriate event handling method.

return super.handleEvent(evt);
/**
 * Connect() Method
 * This method connects the client program to the server program using the port 8003 that was specified at the beginning of the program.
 */

public void Connect() throws IOException
{
    NewsGrpSocket = new Socket(Hostname,NewsGrpPort);
    FromServer = new DataInputStream(NewsGrpSocket.getInputStream());
    ToServer = new PrintStream(new DataOutputStream(NewsGrpSocket.getOutputStream()));
}

/**
 * CreateReply() Method
 * This method creates the discussion file name and combines all the discussion data into one string and sends it to the server.
 * The discussion made by the user has the '*' char attached to it.
 * This allows the program to detect a new line when it displays the information to the user.
 */

public void CreateReply()
{
    String String1 = "", // First Word entered by user
    String2 = "", // Second Word entered by user
    Final_Name = ", // Final copy of the modified discussion name entered by user
    Content = "", // Contains all the discussion data as ONE string
    Replies = "", // Contains the discussion contents
    Modified_reply = ""; // Contains the Modified discussion contents

    Content = CheckDataToServer();

    try
    {
        // Establish necessary connection between Server and Client
        Connect();
    }
}
ToServer.println(Content); // Send data to the server

// Signals the client that the data has been transmitted
ToServer.println("End Work");

} // try

// Clear the TextFields for new Entry
DateBox.setText(""), // Clear DateBox for next entry
Contents.setText(""), // Clear Contents for next entry

// Once the data has been properly sent, dispose of the
// Create_Replay4 frame.
this.hide();
this.dispose();

} // Create_Reply4 Method

/**------------------------------------------------------------------*/
/** CheckDataToServer() Method */
/**------------------------------------------------------------------*/
/*This method validate all discussion data entered by the user. */
/*If the data is not entered by user, the 'NO DATA' tag will be */
/*used to replace empty fields. */
/**------------------------------------------------------------------*/

public String CheckDataToServer()
{
    String String1 = "", // Temporary variable use to
                      // validate discussion name
    String2 = "", // Temporary variable use to
                   // validate discussion name
    Final_RName = "", // Final discussion name
                   // after modification
    Topic_Field = "", // Contains the Topic Name
    RName_Field = "", // Contains the discussion
                  // name
    Date_Field = "", // Contains the date of the
                  // discussion
    User_Field = "", // Contains the user name
    Reply_Field = "", // Contains the Initial
                 // discussion entered by
                 // user
    ModifiedReply = "", // Contains the Modified
                      // discussion entered by
                      // user
    Content = "", // Final contents to be
                  // displayed

    // To indicate that there is no data
    No_Data_Tag = "NO DATA",
// To indicate that user didn't select any image
No_Image = "NoPicture.gif";

// Retrieve the data entered by the user into local variables.
Topic_Field = TopicName.getText();
RName_Field = ReplyName.getText();
Date_Field = DateBox.getText();
User_Field = UserName.getSelectedItem();
Reply_Field = Contents.getText();

// Data are combined as ONE string. There are two parts to this
// string. The first contains the command to the server and the
// second is the actual discussion data. We start by writing
// the command to the Content string.
Content += Command + "^";

// Check the topic field
if (Topic_Field.equals("") )
{
    Topic_Field = No_Data_Tag;
    Content += Topic_Field + ";
} // if
else
{
    Content += Topic_Field + ";
} // else

// Check the discussion name field
if (RName_Field.equals("") )
{
    RName_Field = No_Data_Tag;
    Content += RName_Field + ";
} // if
else
{
    // The program have to store the discussion name in the
    // file as a single string. This can be achieved by
    // seperating the discussion name with the '_' char. E.g.
    // "This_is_a_Test". The system automatically detects
    // instances when the user fails to use the '_' char.
    StringTokenizer RName = new StringTokenizer ( RName_Field );
    while ( RName.hasMoreTokens() )
    {
        String1 = RName.nextToken();
        Final_RName += String1;
        if ( RName.hasMoreTokens() )
        {
```java
String2 = RName.nextToken();
Final_RName += "_" + String2;
if (RName.hasMoreTokens())
    Final_RName += "_";
} // if
else
    System.out.println("Final Reply Name = " + Final_RName);
} // else
} // while
Content += Final_RName + "#";
} // else

// Check the date field
if (Date_Field.equals(""))
    Date_Field = No_Data_Tag;
    Content += Date_Field + "#";
} // if
else
    Content += Date_Field + "#";
} // else

// Check the User name field
if (User_Field.equals(""))
    User_Field = No_Data_Tag;
    Content += User_Field + "#";
} // if
else
    Content += User_Field + "#";
} // else

// Check the discussion field
if (Reply_Field.equals(""))
    Reply_Field = No_Data_Tag;
    Content += Reply_Field + "#";
} // if
else
```
When the user enters the comments, it is likely that the RETURN key is pressed to go to the next line. To allow original data to be presented in its original form, the '*' char is used to replace the '\n' char.

```java
StringTokenizer st = new StringTokenizer
    (Reply_Field, "\n");

while (st.hasMoreTokens())
{
    Modified_Replay += st.nextToken();
    Modified_Replay += "*";
}

Content += Modified_Replay + "#";
```

// Check that the Image name is also included in the string. If the user did not select an image, the "NoPictT.gif" string is the default.

```java
if (ImageName.equals(""))
{
    ImageName = No_Image;
    Content += ImageName + ";";
}
else
{
    Content += ImageName + ";";
}
```

return Content;

// CheckDataToServer() Method
RName_Field = "", // Contains the discussion name

Date_Field = "", // Contains the date that the discussion was created

Reply_Field = "", // Contains the Initial discussion entered by user

Modified_Repl = "", // Contains the Modified discussion entered by user

Content = "", // Final contents to be displayed

// Used to indicate that there is no data.
No_Data_Tag = "NO DATA";

int User_Field = 0; // Contains the index of the user name pop-up list

// Retrieve the data entered by the user into local variables.
Topic_Field = TopicName.getText();
RName_Field = ReplyName.getText();
Date_Field = DateBox.getText();
User_Field = UserName.getSelectedIndex();
Reply_Field = Contents.getText();

// Data are combined as ONE string before they are sent to the Imgbox frame. The first item in the string is the client's request to the server.
Content += Command + ";"

// Check the Topic field
if (Topic_Field.equals(""))
{
    Topic_Field = No_Data_Tag;
    Content += Topic_Field + "#";
} // if
else
{
    Content += Topic_Field + "#";
} // else

// Check the discussion field.
if (RName_Field.equals(""))
{
    RName_Field = No_Data_Tag;
    Content += RName_Field + "#";
} // if
else
{
    // The program have to store the discussion name in the file as a single string. This can be achieved by
StringTokenizer RName = new StringTokenizer(RName_Field);
while (RName.hasMoreTokens())
{
    String1 = RName.nextToken();
    Final_RName += String1;
    if (RName.hasMoreTokens())
    {
        String2 = RName.nextToken();
        Final_RName += "_" + String2;
        if (RName.hasMoreTokens())
        {
            Final_RName += "_";
        } // if
    } // else
    else
    {
        System.out.println("Final Reply Name = "
                        + Final_RName);
    } // else
} // while

Content += Final_RName + ";"

// Check the date field
if (Date_Field.equals(""))
{
    Date_Field = No_Data_Tag;
    Content += Date_Field + ";"
} // if
else
{
    Content += Date_Field + ";"
} // else

// Do not have to check if UserName Field are
// Empty since they must contain an index anyway
Content += User_Field + ";"

// Check the discussion field
if (Reply_Field.equals(""))
{
    Reply_Field = No_Data_Tag;
    Content += Reply_Field + ";"
} // if
else
{

// When the user enters the comments, it is likely that
// the RETURN key is pressed to go to the next line. To
// allow original data to be presented in its original
// form, the '*' char is used to replace the '\n' char.

StringTokenizer st = new StringTokenizer
    (Reply_Field,  "\n");

while (st.hasMoreTokens())
{
    // Get Text on next line
    Modified_Reply  +=  st.nextToken();
    // Insert '*' char to it.
    Modified_Reply  +=  "*";
}  // while

Content  +=  Modified_Reply  +  "#";

}  // else

return Content;

}  // CheckDataToImgBox() Method

/*=================================================================* */
public void clickedPickImageButton()
{
    String Content  =  ":// Contains all Create_Reply4
    // frame data

    // Obtain current discussion data and send it to the ImgBox
    // frame so that data can be used to update the Create_Reply4
    // frame later including the the image name selected.
    Content  =  CheckDataToImgBox();

    // Load ImgBox frame and send current Create_Reply4 data to
    // it.
    ImgBox  ImgSelectionBox;
    ImgSelectionBox  =  new ImgBox(NewsGrpSocket,  Content);
    ImgSelectionBox.show();
}

}  // clickedPickImageButton() Method
public void clickedCancelButton() {
    Reply_Cancel RCancel;
    RCancel = new Reply_Cancel(this);
    RCancel.show();
}

public void addImage(String ImageLocation) {
    URL ImageURL; // Contains the Universal Resource
    // Locator address where the 'Smilies'
    // resides.

    // Retrieve the 'Smilies' from the server
    try {
        ImageURL = new URL(ImageLocation);
        MyImage = Toolkit.getDefaultToolkit().getImage(ImageURL);
        // Now display the image onto the frame
        imageViewer1.setImage(MyImage);
        imageViewer1.repaint();
    } // try
    catch(MalformedURLException e) {
        System.out.println(e.toString());
    } // catch()

    MediaTracker tracker = new MediaTracker(this);
    tracker.addImage(MyImage, 0);

    try {
        tracker.waitForAll();
    } // try
    catch(InterruptedException e) {
        System.out.println(e.toString());
    } // catch
}
Create_Retry4.java

/**
 * Program Name : Create_Topic4.java
 * Purpose : This class get the Topic name entered by the user and send the Topic name to the server so that the Topic?.txt can be created and the Topic_List.txt updated.
 * 
 * The Topic name entered by the user is checked to make sure that it is not empty. If it is empty, an Error message will be displayed, otherwise the Topic Name entered will be modified to include the _ char. E.g => This_is_a_Test
 */

import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.StringTokenizer;

public class Create_Topic4 extends Frame {

   Socket NewsGrpSocket;
   PrintStream ToServer = null;
   DataInputStream FromServer = null;
   int NewsGrpPort = 8003;

   String Command = "",
                   Hostname = "krakatoa.fste.ac.cowan.edu.au";

   TextField TopicName; // The Topic name under discussion
   Label TopicLabel; // Topic name Label
   Button Create_Topic; // Once clicked the Topic name
APPENDIX B : SOURCE CODE [CLIENT SIDE]

```java
// will be created Cancel;
// Cancel the current operation

/**
 * Create_Topic4(Socket, String) CONSTRUCTOR *
 */
/**
 * This constructor provides the interface by which the user can *
 * create the Topic name. *
 */
/**
 *****************************************/

public Create_Topic4(Socket MySocket,
                      String CommandString)
{
    // The title of the Create_Topic4 frame
    super("Create Topic window");

    // Assign values sent from NewsGrpIconVote (Parent class) to
    // Local variables
    // Contains the client's request to the server
    Command = CommandString;

    // Passing the NewsGrpIconVote frame socket to the current
    // frame so that socket connection with the server can be
    // established
    NewsGrpSocket = MySocket;

    // Organises the layout of the entire Create_Topic4 frame
    setLayout(new GridLayout(3,2));

    // Declare the panels to use for this frame
    Panel TLabel_Panel = new Panel(); // Topic Label
    Panel TName_Panel = new Panel(); // Topic name
    Panel TButtons_Panel = new Panel(); // Buttons

    // Arrange the TLabel_Panel
    TopicLabel = new Label("Topic Name");
    TLabel_Panel.add(TopicLabel); // Add the Topic Label to Frame
    add("North", TLabel_Panel);

    // Arrange the TName_Panel
    TopicName = new TextField(20);
    TName_Panel.add(TopicName); // Add the Topic Name to Frame
    add("Center", TName_Panel);

    // Add the components to the frame and arrange it on the
    // TButtons_Panel
    Create_Topic = new Button("Create Topic");
    Cancel = new Button("Cancel");

    // Add the Create_Topic Button to Frame
```
TButtons_Panel.add(Create_Topic);

// Add the Cancel Button to Frame
TButtons_Panel.add(Cancel);

add("South", TButtons_Panel);

// Set the size of the Create_Topic frame and make sure that
// the frame is not resizeable
resize(200, 150);
setResizable(false);

} // Create_Topic41(Socket, String) CONSTRUCTOR

/*==============================================*/
/*                  show() Method                  */
/*==============================================*/
/* This method display the main interface for interaction with the*/
/* user. Note that the move() command also set the position of the*/
/* program on the screen.                             */
/*==============================================*/

public synchronized void show()
{
    move(50, 50);
    super.show();
} // show() method
public void Connect() throws IOException
{
    NewsGrpSocket = new Socket(Hostname,NewsGrpPort);
    FromServer = new DataInputStream(NewsGrpSocket.getInputStream());
    ToServer = new PrintStream(new DataOutputStream(NewsGrpSocket.getOutputStream()));
}

public boolean handleEvent(Event evt)
{
    if (evt.target instanceof Button)
    {
        if (evt.arg.equals("Cancel"))
        {
            clickedCancelButton;
            // Dispose the Create_Topic 4 frame
            this.hide();
            this.dispose();
            // Return the user to the NewsGrpIconVote frame.
            NewsGrpIconVote MyMenu = new NewsGrpIconVote();
            MyMenu.show();
            return true;
        }
    }
    else
    {
        if (evt.arg.equals("Create Topic"))
        {
            String Entered_TName = TopicName.getText();
            // Checks if user enters the Topic name
            if (Entered_TName.equals(""))
            {

        184
APPENDIX B: SOURCE CODE [CLIENT SIDE]

```
TopicMsg = new TopicMsg(this);

} // if
else
{
    // If user has entered the Topic name, send
    // it to the server
    clickedCreateTopicButton();

    // Destroy the window once data is handled
    this.hide();
    this.dispose();

    // Return the user to the NewsGrpIconVote
    // frame.
    NewsGrpIconVote MyMenu;
    MyMenu = new NewsGrpIconVote();
    MyMenu.show();
} // else

return true;
} // if

if (evt.id == Event.WINDOW_DESTROY)
{
    // Load up the Cancel dialog box. Inform user that
    // the topic has not been created.
    clickedCancelButton();

    // Dispose the Create Topic Dialog Box
    this.hide();
    this.dispose();

    // If the user closes the Create Topic frame, load up the
    // NewsGrpIconVote frame again.
    NewsGrpIconVote MyMenu = new NewsGrpIconVote();
    MyMenu.show();

    return true;
} // if

// If the event received by handleEvent is not the
// appropriate event, call the superclass handleEvent to
// send the event to the appropriate event handling method.

return super.handleEvent(evt);
} // handleEvent() method
```

185
/* In order for the server to carry out its requested task, the */
/* client program needs to give it a command. This method sends */
/* the command and data to the server for updating. */
/**======================*/

public void clickedCreateTopicButton()
{
    // Contains the Topic name string

    StringTokenizer TName;

    // First Word and second word entered by user

    String String1 = "", String2 = "", Final_TName = "", // Final modified
    // discussion name
    // entered by user

    Topic_String = ""; // Command -
    // Topic Name

    try
    {
        // Established the connection between client and server

        Connect();

        // The program have to store the Topic name in the file
        // as a single string. This can be achieved by
        // separating the Topic name with the '_' char. E.g.
        // "This_is_a_Test". The system automatically detects
        // instances when the user fails to use the '_' char.

        TName = new StringTokenizer(TopicName.getText());

        while (TName.hasMoreTokens())
        {
            // Get the first token and add it to the Final_TName
            // string

            String1 = TName.nextToken();

            Final_TName += String1;

            if (TName.hasMoreTokens())
            {
                // If there are more tokens, add the '_' char
                // first before adding the next string.

                String2 = TName.nextToken();

                Final_TName += "_" + String2;

                if (!TName.hasMoreTokens())
                {
                    Final_TName += "_";
                }
            }
        }
    }
}
APPENDIX B: SOURCE CODE [CLIENTSIDE]

```java
} // while

// Combine the command and Topic name as one string
// separated by the '*' char
Topic_String = Command + '*' + Final_TName;

// Sending the command and Topic name to the server
ToServer.println(Topic_String);

// Signals the server that data has been sent
ToServer.println("End Work");
```

```java
} // try

catch (IOException e)
{
    System.err.println(e);
} // catch

// Clear the TextFields for next entry
TopicName.setText("");
```

```java
} // clickedCreateTopicButton() method
```
/**==============================================================*/
/** clickedCancelButton() Method */
/** This method is triggered when the user clicked on the 'Cancel' */
/** button */
/**==============================================================*/

public void clickedCancelButton()
{
    Topic_Cancel TCancel;
    TCancel = new Topic_Cancel(this);
    TCancel.show();
} // clickedCancelButton() method

} // Create Topic4 Class

ImgBox.java

/**************************************************************************/
---
Program Name : ImgBox.java
Purpose : This program allows the user to select different images to represent their reactions to the comments made by others within the Topic of discussion.

This allows other users to look at the reaction instantly and response to the discussion with their own 'Smileys'.

This program implements the classes from Visual Cafe.

NOTE : The Images retrieved are loaded from the server using the appropriate URLs.

import java.awt.*;
import java.net.*;
import java.io.*;

// Uses Visual Cafe Classes
import symantec.itools.awt.ImageListBox;
import symantec.itools.multimedia.imageViewer;

public class ImgBox extends Frame
{
    // Used to store the Socket Object sent by Create_Reply4 frame
    Socket NewsGrpSocket;

188
APPENDIX B : SOURCE CODE [CLIENT SIDE]

Label  Message_Label,           // Instruction for user
       Response_Label;          // Explains what each image
                                // represents

TextArea  Response_Field;       // Display the meaning of
                                // each of the Image

Button    OK_Button,            // Once clicked, the
       Cancel_Button;          // Create_Reply4 frame will
                                // be loaded
                                // Cancel current operation
                                // and return to main
                                // interface

// Inform the program the exact location of the image stored in
// the server

URL     ImageURL;

Image   MyImage;               // Contains the Image
       // Object to be displayed

ImageListBox  imageListBox1;   // Contains the Image in a
                                // form of a list

ImageViewer  imageViewer1;    // Allows the image to be
                                // displayed

// This char determines which Smilies the user has selected
char    Image_Index;

String   Create_ReplyData = "", // Data entered by
          // user in
          // Create_Reply4 Frame

Contents = ""; // All contents from
          // Create_Reply4
          // Frame and the
          // Image name

/*=================================================================*1
*I* ImgBox(Socket, String) CONSTRUCTOR
I*=================================================================*1
*/
public ImgBox(Socket MySocket,
              String Reply_Info)
{
    // The title of the ImgBox frame
    super("Image Selection Window");

    // Assign values sent from NewsGrpIconVote (Parent class) to
    // Local variables
    NewsGrpSocket = MySocket;

APPENDIX B : SOURCE CODE [CLIENT SIDE]

// Store the data from Create Reply4 frame and add the Image // name to the string as well.
Create_ReplyData = Reply_Info;

// Organises the layout of the entire ImgBox frame
setLayout(null);
addNotify();
resize(insets().left + insets().right + 382, insets().top + insets().bottom + 320);

// Initialise all AWT components
Message_Label = new Label("Click the ImageList to select the Image");
OK_Button = new Button("OK");
Cancel_Button = new Button("Cancel");
Response_Field = new TextArea(10,10);
Response_Label = new Label("Icon's Representation : ");

// Add the AWT components to the ImgBox frame and position // them accordingly
add(Message_Label);
Message_Label.reshape(insets().left + 28,insets().top + 7,259,19);
add(OK_Button);
OK_Button.reshape(insets().left + 42,insets().top + 285,91,26);
add(Cancel_Button);
Cancel_Button.reshape(insets().left + 189,insets().top + 285,91,26);
add(Response_Field);
Response_Field.setEditable(false);
Response_Field.reshape(insets().left + 30,insets().top + 180,350,85);
add(Response_Label);
Response_Label.reshape(insets().left + 30,insets().top + 152,196,13);
setBackground(new Color(12632256));

// This section implements the image list
imageListBox1 = new symantec.itools.awt.ImageListBox();
imageListBox1.reshape(48,52,220,120);
add(imageListBox1);

// The imageListBox displays an image in a list so that // users can select the smiley that best represent their // response at the time.
imageListBox1.select(0);
imageListBox1.setBackground(Color.white);
imageListBox1.setComboMode(false);
imageListBox1.setColumnsToShow(0);
imageListBox1.setShowHorizontalScroll(true);
imageListBox1.setShowVerticalScroll(true);
imageListBox1.setBorderType(ImageListBox.BORDER_REGULAR);
imageListBox1.setCellBorders(false);
imageListBox1.setDefaultEnabledTextColor(new Color(0));
imageListBox1.setDisabledTextColor(new Color(156));
imageListBox1.setListBoxFont(new java.awt.Font("TimesRoman", java.awt.Font.PLAIN, 11));

// The imageViewer displays an enlarged image of the Smiley
// selected by the user in the imageListBox
imageViewer1 = new ImageViewer();
imageViewer1.reshape(280, 16, 150, 113);
add(imageViewer1);
imageViewer1.setCenterMode(true);

// Load up the 'Smilies' from the server

// Set the size of the ImgBox frame and disallow user from
// resizing it.
resize(430, 410);
setResizable(false);
}

} // ImgBox() CONSTRUCTOR
public void imageListBox1_ListSelect(Event event)
{
  // Contains the name of the selected image
  String ImageName = "";

  ImageViewer1.setImage(imageListBox1.getImage(imageListBox1.getSelectedIndex()));
  ImageViewer1.repaint();

  // Determine what image is selected and explains the meaning
  // of the 'Smiley'.
  ImageName = imageListBox1.getItem(imageListBox1.getSelectedIndex());

  // Assign Image_Index for each image so that the program is
  // able to tell which image has been selected.
  if (ImageName.equals("OverJoyT.gif"))
  {
    Image_Index = '1';
    InformUser(Image_Index);
  } // if
  else
  
    if (ImageName.equals("SmileTN.gif"))
    {
      Image_Index = '2';
      InformUser(Image_Index);
    } // if
  else
    
      if (ImageName.equals("NeutralTN.gif"))
      {
        Image_Index = '3';
        InformUser(Image_Index);
      } // if
    else
    
      if (ImageName.equals("SadTN.gif"))
      {
        Image_Index = '4';
        InformUser(Image_Index);
      } // if
    else
      
        if (ImageName.equals("AngryTN.gif"))
        {
          Image_Index = '5';
          InformUser(Image_Index);
        } // if
      else
        
          if (ImageName.equals("BoringTN.gif"))
          {
            Image_Index = '6';
          }
InformUser(Image_Index);
} // if
else
  if (ImageName.equals("NoPictT.gif"))
  {
    Image_Index = '7';
    InformUser(Image_Index);
  } // if

} // imageListBox1_ListSelect(Event) method

/*============================================================*/
/* InformUser(int) Method */
/* This method allows the program to detect which image is */
/* selected by user to inform them about the image's meaning. */
/*============================================================*/

public void InformUser(char Image_Char)
{
  // Declare the meanings of all the Smilies
  String OverjoyString =
    "This icon represents an OVERJOYED response.\n" +
    "An example of this can be drawn from the programmer\n" +
    "who has finally allow this program to be tested.\n",
  SmileString =
    "This icon represents a HAPPY response.\n" +
    "It suggests that one is SATISFIED with the comment\n" +
    "or AGREES with the comment made by others",
  NeutralString =
    "This icon represents a NEUTRAL response \n" +
    "This suggests that one is sitting on the fence with\n" +
    "regards to the comment made by others.",
  SadString =
    "This icon represents a DISAPPOINTED or SAD Response \n" +
    "This suggests that one is upset or unhappy with the \n" +
    "comments made by others",
  AngryString =
    "This icon represents an ANGRY response. \n" +
    "This suggests that one is deeply UPSET with the \n" +
    "comments made by others",
  BoringString =
    "This icon suggests that one is DISINTERESTED with \n" +
    "the topic under discussion. The participants would like\n" +
    "to change the topic of discussion.",
  NoPictString =
"This icon is used when the participant does not wish to express their thoughts or comments made by others."

switch(Image_Char)
{
  case '1' : // Overjoy Icon
    Response_Field.setText(OverjoyString);
    break;
  case '2' : // Smile Icon
    Response_Field.setText(SmileString);
    break;
  case '3' : // Neutral Icon
    Response_Field.setText(NeutralString);
    break;
  case '4' : // Sad Icon
    Response_Field.setText(SadString);
    break;
  case '5' : // Angry Icon
    Response_Field.setText(AngryString);
    break;
  case '6' : // Boring Icon
    Response_Field.setText(BoringString);
    break;
  case '7' : // No Picture Icon
    Response_Field.setText(NoPictString);
    break;
}

// InformUser(char) method

/*============================================================================*/
/*
 * addImage(String, String) Method */
/*============================================================================*/
/* This method retrieved the required images from the URL */
/* provided. Since this program uses a frame, the */
/ **Toolkit.getDefaultToolkit()** method will be used.

```java
public void addImage(String ImageLocation, String name) {
    try {
        ImageURL = new URL(ImageLocation);
        MyImage = Toolkit.getDefaultToolkit().getImage(ImageURL);
    } // try
    catch(MalformedURLException e) {
        System.out.println(e.toString());
    } // catch()
    MediaTracker tracker = new MediaTracker(this);
    // Add the images to the MediaTracker one at a time
    tracker.addImage(MyImage, 0);
    try {
        // Wait for all the images to be loaded
        tracker.waitForAll();
    } // try
    catch(InterruptedException e) {
        System.out.println("Not all images are loaded "+ e);
    } // catch()
    finally, add the images to the imageListBOX1
    imageListBOX1.addItem(MyImage, name);
} // addImage(String, String) method
```

/*=================================================================·1
* show() Method
*/
/*-----------------------------------------------------------------•1
* This method displays the main interface for interaction with the
* user. Note that the move() command also sets the position of the
* program on the screen.
*=================================================================·1
*/

```java
public synchronized void show() {
    move(50, 50);
    super.show();
} // show() method
```

/*=================================================================·1
* handleEvent() Method
*/
/*-----------------------------------------------------------------•1
* This method handles all the Button Events as well as when the
* window is destroyed.
*=================================================================·1
*/
public boolean handleEvent(Event event) {
    // Check if it is a button event and handles the buttons appropriately
    if (event.id == Event.ACTION_EVENT &&
        event.target == OK_Button) {
        clickedOKButton();
        // Once the Create_Repl4 frame is loaded, hide the ImgBox frame
        this.hide();
        return true;
    } // if
    else
    if (event.id == Event.ACTION_EVENT &&
        event.target == Cancel_Button) {
        clickedCancelButton();
        return true;
    } // if
    else
    // If the imageListBox1 is clicked, display the appropriate images.
    if (event.target == imageListBox1 &&
        event.id == Event.LIST_SELECT) {
        imageListBox1_ListSelect(event);
        return true;
    } // if
    // If the user closes the Create_Repl4 frame, load up the Create_Repl4 frame. This probably means
    // that they did not select an image, thus we assign
    // the Blank smiley as the default.
    if(event.id == Event.WINDOW_DESTROY) {
        Contents = Create_ReplyData +
                  "NoPictT.gif" + ";"
        Create_Repl4 Reply_Box;
        Reply_Box = new Create_Repl4(NewsGrpSocket,
                                     Contents);
        Reply_Box.show();
        this.hide();
    } // if

    // If the event received by handleEvent is not the
APPENDIX B: SOURCE CODE [CLIENT SIDE]

// appropriate event, call the superclass handleEvent to
// send the event to the appropriate event handling
// method.

return super.handleEvent(event);
} // handleEvent(Event) method

/*=============================================================================*/
void clickedCancelButton()
/*=============================================================================*/
public void clickedCancelButton()
{
Contents Create_ReplyData = "NoPictT.gif" + ";
Create_Reply4 Reply_Box;
Reply_Box = new Create_Reply4(NewsGrpSocket, Contents);
Reply_Box.show();
this. hide();
} // clickedCancelButton() method

/*=============================================================================*/
void clickedOKButton()
/*=============================================================================*/
public void clickedOKButton()
{
String Reply_Name = "; // Contains the discussion name
ImageName = "; // Contains the image name

// Get the image name selected by the user
ImageName = imageListBox1.getItem(imageListBox1
.getSelectedIndex());

// Add the selected image name to previously entered
// discussion data so that it can be sent back to the
// Create_Reply4 frame for updating.
Contents = Create_ReplyData + ImageName + ";

// Load the Create_Reply4 frame and pass the updated data to
// it.
Create_Reply4 Reply_Box;
Reply_Box = new Create_Reply4(NewsGrpSocket, Contents);

198
APPENDIX B: SOURCE CODE [CLIENT SIDE]

    Reply_Box.show();

} // clickedOKButton() method

} // ImgBox Class
MyDate.java

/*===============================================================*/

Program Name : MyDate.java
Purpose : This class produce the date that I have formatted
for the NewsBrowse program.

#========================================================================================================*/
import java.util.Date;
import java.io.*;

class MyDate
{
    // Program declaration
    static Date date = new Date();

    /*-----------------------------------------------*/
    MyDate(Socket, String) CONSTRUCTOR
    /*-----------------------------------------------*/

    public void MyDate()
    {
        date = new Date();
    } // MyDate(Socket, String) Constructor

    /*-----------------------------------------------*/
    WhatTime() Method
    /*-----------------------------------------------*/

    public String WhatTime()
    {
        String MyHours = "", // Contains the hours of the day
        MySeconds = "", // Contains the seconds of the
                        // day
        FinalMinutes = "", // Contains the minutes of the
                            // day
        MyTime = "", // Contains today's time of the
                     // day
        SetMinutes = "0";

        // Check if the integer is single digit (E.g 0 - 7). If
        // single digit, add a 0 int value in front of it.
        int MyMinutes = 0;

        // Display the date and time
        System.out.println("Today's Date: "+MyTime); // Display date
        System.out.println("Today's Time: "+MyHours+MySeconds);
        System.out.println("Today's Date: "+MyTime);
        System.out.println("Today's Time: "+MyHours+MySeconds);
    }

    /*-----------------------------------------------*/
}
Appendix B: Source Code [Client side]

// Note. the String.valueOf() method takes an argument
// of various types, convert those arguments to type String
// and return them as String Objects. The objective here is
// to change all the data retrieved as a String object.

MyHours = String.valueOf(date.getHours());
MySeconds = String.valueOf(date.getSeconds());

// Since the getMinutes() retrieve a single digit (i.e 0 - 7)
// thus we want to add a 0 int value in front to make it
// look more presentable on the screen. E.g system retrieve 7
// change to 07 for minutes.

MyMinutes = date.getMinutes();
if (MyMinutes >= 0 && MyMinutes <= 9)
{
    FinalMinutes = SetMinutes +
          String.valueOf(MyMinutes);
}
else
{
    // If the value retrieve is double digit, leave it alone
    FinalMinutes = String.valueOf(MyMinutes);
}
// else

MyTime = MyHours + ":" + FinalMinutes + ":" + MySeconds;

return MyTime;
}
// WhatTime() method

/*==================================================================*/
/**
 * WhatDay() Method
 */
/*==================================================================*/

public String WhatDay(int Day)
{
    switch(Day)
    {
    case 0 : return "Sun";     // Sunday
    case 1 : return "Mon";     // Monday
    case 2 : return "Tues";    // Tuesday
    case 3 : return "Wed";     // Wednesday
    case 4 : return "Thurs";   // Thursday
    case 5 : return "Fri";     // Friday
    case 6 : return "Sat";     // Saturday
    default  : return "Wrong Day !";
    }
    // switch
}
// WhatDay() method

201
APPENDIX B: SOURCE CODE [CLIENT SIDE]

```java
public class javaDateExample {

    public int WhatMonth(int Month) {
        int My_Month = 0; // The final value to determine
                         // the current month
        My_Month = Month + 1; // Increase the month value to
                                // reflect the correct month.
        return My_Month;
    }

    public String TodayDate() {
        String TodayDate = "", // Date Format => day/month/year
                            Today = ""; // Today's Date & Day + Time
        // Retrieve the final date format of dd/mm/yy
        TodayDate = date.getDate() + "/
                    + WhatMonth(date.getMonth()) + "/19" +
                    date.getYear();

        // Now we are ready to display the date information to the
        // client.
        Today = WhatDay(date.getDay()) + " " + TodayDate
               + " " + WhatTime();
        return Today;
    }
}
```

The java DATE.getMonth() method returns an integer value. The method used value 0-11 to represent Jan-Dec with value 0 representing January & 1 as February thus June is the value 5 which is incorrect if you want to display the date format as dd/mm/yy (25/6/1997 for June). This method is used to convert the integer value to the correct format by incrementing the value by 1.
/**
  main(String) Method
  */
/**
  Contains command to retrieve the date information.
  */
/**===============================================================================================*/

public static void main(String args[])
{
    MyDate DateDemo = new MyDate();
    String Today = DateDemo.TodayDate();
    System.out.println("Thavee's Date = " + Today);
    try
    {
        System.out.println("Press <ENTER> to Quit");
        System.in.read();
    } // try
    catch(IOException e) {}
} // main

} // MyDate Class
**APPENDIX B: SOURCE CODE [CLIENT SIDE]**

**VoteWin2.java**

```java
/*================================================================================*/

Program Name    : VoteWin2.java
Purpose          : This system implements the voting system. The main purpose of this system is to enable the user to vote in favour or against the topic under discussion. This allows a more interactive discussion among the participants.

================================================================================*/

import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.StringTokenizer;

public class VoteWin2 extends Frame
{
    // Used to store the Socket Object sent by NewsGrpIconVote Main Frame
    Socket                 NewsGrpSocket;
    // Output the data to the server, through the socket, to update necessary files
    PrintStream            ToServer    = null;
    // Get the data from the server through the socket
    DataInputStream        FromServer = null;
    // Using port 8003 to create the discussion name.
    int                    NewsGrpPort = 8003;
    // The name of the server
    String                 Hostname   = "krakatoa.fste.ac.cowan.edu.au",
                           RFileName  = "; // Contains the discussion file name to write the data to

    Label                  TOD_Label,   // TOD refers to Topic Of Discussion
                           Discussion_Label; // Discussion details label
    TextField             TOD_Field;   // The Textfield to hold the topic of discussion
    TextArea              Discussion_Field; // Contains the comments made by the user
    CheckboxGroup         UserVote;    // CheckboxGroup contains the user's opinion on the discussion
    Checkbox              Agree,       // Vote in favour of the
```

204
APPENDIX B : SOURCE CODE [CLIENT SIDE]

// discussion
Disagree;  // Vote against the
// discussion

Button
Submit,  // Sends the user's data to
// the server

Cancel;  // Cancel the current
// operation

int Total = 0,  // Total number of people
// participating

Voted = 0,  // Number of people who
// have submitted their
// opinion

NotVoted = 0,  // Number of people who
// have not submit their
// opinion

InFavour = 0,  // Number of people in
// favour of the discussion

Against = 0;  // Number of people against
// the discussion

String Discussion = "",  // The discussion contents

Reply_Data = "",  // Other Discussion details
// needed to update the
// discussion (Reply?.txt)
// file

VoteStatus = "",  // Inform program if the
// user is voting the first
// or the 'n' time

InitialVote = "";  // Determine if user agree
// or disagree

/*=================================================================*1
*/

VoteWindow() CONSTRUCTOR
/*=================================================================*1
*/

/* This constructor provides the interface by which the user can
* enter their vote on a chosen Discussion.
*=================================================================*1*/

public VoteWin2(String Selected_TName,
String Reply_Info,
String Reply_Content,
String Reply_File,
int TotalNumber,
int Number_Voted,
int Number_NotVoted,
int Number_InFavour,
int Number_Against,
String Vote_Status,
String Initial_Vote)
{
    super("Polling Window");  // Title of the VoteWin2 frame

    // Assign the arguments received from ValidateVotersBox
    // class to Global variable. It receives the same
// information as the ValidateVotersBox class but it also
// carries the Voters Status (Vote FIRST/SECOND time) which
// is determined in the ValidateVotersBox class. The Voters
// status is needed to determine if the user has previously
// submit in their opinion.

// Contains the comments made by the user
Discussion = Reply_Content;

// Contains the Author name, Date data
Reply_Data = Reply_Info;

// Contains the discussion file to write the polling data to
RFileName = Reply_File;

// Total number of people participating
Total = TotalNumber;

// Number of people who have submitted their opinion
Voted = Number_Voted;

// Number of people who have not submit their opinion
NotVoted = Number_NotVoted;

// Number of people in favour of the discussion
InFavour = Number_InFavour;

// Number of people against the discussion
Against = Number_Against;

// Inform program if the user is voting the first or the 'n'
// time
VoteStatus = Vote_Status;

// Used as a flag to determine if the user is voting the
// first or the 'n' time
InitialVote = Initial_Vote;

// Initialise all AWT used
TOD_Label = new Label("Topic Of Discussion");
TOD_Field = new TextField(50);
Discussion_Label = new Label("Discussion");
Discussion_Field = new TextArea(15, 15);
UserVote = new CheckboxGroup();
Agree = new Checkbox("Agree", UserVote, false);
Disagree = new Checkbox("Disagree", UserVote, false);
Submit = new Button("Submit");
Cancel = new Button("Cancel");

// Layout Management Area
this.setLayout(new BorderLayout());

// Declare Panels to use
Panel Top_Info = new Panel();// Uses BorderLayout
Panel Topic_Info = new Panel();// Uses BorderLayout
Panel Discussion_Info = new Panel();// Uses FlowLayout
Panel UserVote_Info = new Panel();// Uses FlowLayout
Panel Button_Info = new Panel();// Uses FlowLayout

// Arrange the Voting information
/**===============================================*/
/** Top_Info Panel */
/** The Top_Info panel contains the topic of discussion, discussion details and the polling options */
/** (i.e Agree/Disagree) */
/**===============================================*/
Top_Info.setLayout(new BorderLayout());

// A sub-section of Top_Info
Topic_Info.setLayout(new BorderLayout());
Topic_Info.add("North", TOD_Label);
Topic_Info.add("Center", TOD_Field);
TOD_Field.setText(Selected_TName);
TOD_Field.setEditable(false);
TOD_Field.setBackground(Color.lightGray);

// A sub-section of Top_Info
Discussion_Info.setLayout(new BorderLayout());
Discussion_Info.add("North", Discussion_Label);
Discussion_Info.add("Center", Discussion_Field);
Discussion_Field.setBackground(Color.lightGray);

// Before we display the discussions, we have to modify the data to its original format.
String New_String = Modify_Discussion(Discussion);

// Now we can put the discussion on the frame. Set the field so that user cannot edit it, only reading is allowed.
Discussion_Field.setText(New_String);
Discussion_Field.setEditable(false);

// A sub-section of Top_Info
UserVote_Info.setLayout(new FlowLayout());
UserVote_Info.add(Agree);
UserVote_Info.add(Disagree);
Top_Info.add("North", Topic_Info);
Top_Info.add("Center", Discussion_Info);
Top_Info.add("South", UserVote_Info);

Button_Info.setLayout(new FlowLayout());
// Add the buttons to the frame
Button_Info.add(Submit);
Button_Info.add(Cancel);
this.add("North", Top_Info);
this.add("South", Button_Info);

// Set the size of the Create Reply frame and make sure the
// frame is not resizable
resize(450, 400);
setResizable(true);
this.setBackground(Color.cyan);

} // CONSTRUCTOR VoteWin2()

/**
 * insets() Method
 */
public Insets insets()
{
    return new Insets(25, 15, 15, 15);
}

/**
 * show() Method
 */
public synchronized void show()
{
    move(50, 50);
    super.show();
}

} // show() Method
public String Modify_Discussion(String Reply_Content) {
    // New string containing the original message format
    String Modified_Reply = "";
    // This program automatically placed the '*' char next to
    // the string when the user presses the 'RETURN' key. Thus we
    // have to convert the '*' char to the next line char
    // ('\n').
    Modified_Reply = Reply_Content.replace('*', '\n');
    return Modified_Reply;
} // Modify_Discussion() Method

public void CheckVote()
{
    int Total_Temp = 0, // Contains the updated value
        Voted_Temp = 0, // Contains the updated value
        NotVoted_Temp = 0, // Contains the updated value
        InFavour_Temp = 0, // Contains the updated value
        Against_Temp = 0; // Contains the updated value
    String Previous_Vote = "";
    // Assign the local variable to the value sent to by
    // the ValidateVotersBox class. Data is lost when the frame
    // is disposed. Thus we have to pass it from classes to
    // classes.
    Total_Temp = Total;
    Voted_Temp = Voted;
    NotVoted_Temp = NotVoted;
    InFavour_Temp = InFavour;
    Against_Temp = Against;
    // If the Agree radio button is chosen, we know that this is
    // the first time the user is submitting their opinion. We
    // updated the polling result based on this information.
    if (Agree.getState() == true)
```java
if (VoteStatus.equals("First_Time"))
{
    if (InitialVote.equals("Neutral"))
    {
        // The total number of people participating
        // stays the same.
        Total = Total_Temp;

        // Increase the number of people who voted by 1
        Voted = Voted + 1;

        // Decrease the number of people who has not
        // voted by 1
        NotVoted = Total - Voted;

        // Since user selected "Agree" button, number of
        // people in favour of the discussion should
        // increase by 1
        InFavour = InFavour + 1;

        // The number of people against the discussion
        // stays the same.
        Against = Against_Temp;

        // Informs the program if the user has voted. if
        // so, what is the original opinion.
        InitialVote = "Agree";
    } // if
} // if
else
{
    // Since the user has already voted ONCE, then the
    // number of people voted should stay the same,
    // since the user is only changing his/her vote
    if (VoteStatus.equals("Second_Time"))
    {
        if (InitialVote.equals("Agree"))
        {
            // The total number of people participating
            // stays the same.
            Total = Total_Temp;

            // Number of people voted stays the same
            Voted = Voted_Temp;

            // Number of people who has NOT VOTED stays
            // the same
            NotVoted = NotVoted_Temp;

            InFavour = InFavour_Temp;
        } // if
    } // if
}
```
APPENDIX B: SOURCE CODE [CLIENT SIDE]

```
Against = Against_Temp;
InitialVote = "Agree";
} // if
else
{
    // The total number of people participating stays the same.
    Total = Total_Temp;
    // Number of people voted stays the same
    Voted = Voted_Temp;
    // Number of people who has NOT VOTED stays the same
    NotVoted = NotVoted_Temp;
    // Increase the number of people in favour by 1
    InFavour = InFavour + 1;
    // Decrease the number of people against by 1
    Against = Against - 1;
    // Informs the program if the user has voted, if so, what is the original opinion.
    InitialVote = "Agree";
} // else
} // if
} // else
else
{
    // User chooses to DISAGREE
    if (VoteStatus.equals("First_Time"))
    {
        if (InitialVote.equals("Neutral"))
        {
            // The total number of people participating stays the same.
            Total = Total_Temp;
            // Increase the number of people voted by 1
            Voted = Voted + 1;
            // Decrease the number of people not voted by 1
            NotVoted = Total - Voted;
            InFavour = InFavour_Temp;
        }
    }
}
```
APPENDIX B : SOURCE CODE [CLIENT SIDE]

```java
// Since user selected "Disagree" button, number
// of people in favour of the argument should
// stay the same

// The number of people against the discussion
// should increase by 1
Against = Against + 1;

// Informs the program if the user has voted, if
// so, what is the original opinion.

InitialVote = "Disagree";
} // if
} // if
else
{
    // Since the user has already voted ONCE, then the
    // number of people voted should stay the same,
    // since the user is only changing his/her vote

    if (VoteStatus.equals("Second_Time"))
    {
        if (InitialVote.equals("Agree"))
        {
            // The total number of people participating
            // stays the same.
            Total = Total_Temp;

            // Number of people voted stays the same
            Voted = Voted_Temp;

            // Number of people who has NOT VOTED stays
            // the same
            NotVoted = NotVoted_Temp;

            // User voted the second time has change his
            // mind
            InFavour = InFavour - 1;

            //Thus number of people against increase by 1
            Against = Against + 1;

            // Informs the program if the user has
            // voted, if so, what is the original
            // opinion.
            InitialVote = "Disagree";
        } // if
    }
    else
    {
        // The total number of people participating
        // stays the same.
        Total = Total_Temp;
    }

    
```

// Number of people voted stays the same
Voted = Voted_Temp;

// Number of people who has NOT VOTED stays
// the same
NotVoted = NotVoted_Temp;

// No change of heart, so result stays the
// same
InFavour = InFavour_Temp;

// No change of heart, so result stays the
// same
Against = Against_Temp;

// Informs the program if the user has
// voted, if so, what is the original
// opinion.
InitialVote = "Disagree";

} // else
} // if
} //else
} // CheckVote() method

/*handleEvent() Method
This method handles all the Button Events as well as when the
window is destroyed.
*/

public boolean handleEvent(Event evt)
{
    String Content = "";
    
    // Handles the button event
    if (evt.target instanceof Button)
    {
        if (evt.arg.equals("Cancel"))
        {
            clickedCancelButton();
            // Dispose the Create_Reply4 frame
            this.hide();
            this.dispose();

            return true;
        } // if

        else
        {
            if (evt.arg.equals("Submit"))
            {
                if ((Agree.getState() == false) &&
                    (Disagree.getState() == false))
                {
                }
            }
        }
    }
} // handleEvent() method
if (ErrorMsg.Message) {
  NewErrorMsg(this);
  Message.show();
  return true;
} //
else {
  // Check and calculate all voting results
  CheckVote();
  // Update the Reply?.txt file in the server
  // to reflect the new opinion poll result
  System.out.println("Updating " + RFileName + " with Opinion Poll result.");
  // Before we send data to the server, combine
  // all the discussion data (Username, Date,
  // Discussion) including the polling data
  // together so that it can be used to update
  // the discussion (Reply?.txt) file.
  // NOTE: We must also include the
  // discussion name so that we know which
  // file to updated.
  Content = Reply_Data + "+" + Discussion + "+" + RFileName + "+" + Total + "+" + Voted + "+" + NotVoted + "+" + InFavour + "+" + Against;
  // Now write the data to the server
  Update_ReplyFile(Content);
  // Dispose & Hide window once data is handled
  this.hide();
  this.dispose();
  // Load the NewsGrpIconVote frame
  NewsGrpIconVote MyMenu;
  MyMenu = new NewsGrpIconVote();
  MyMenu.show();
  return true;
} // else
} // if

// Handles the windows event
if (evt.id == Event.WINDOW_DESTROY) {
  this.hide();
  this.dispose();
  return true;
}
APPENDIX B: SOURCE CODE [CLIENT SIDE]

If the event received by handleEvent is not the appropriate event, call the superclass handleEvent to send the event to the appropriate event handling method.

```java
return super.handleEvent(evt);
}
```
APPENDIX B: SOURCE CODE [CLIENT SIDE]

ToServer.println("End Work");

} // try

catch(IOException e)
{
    System.err.println(e);
} // catch

} // UpdateReplyFile() method

/**
* The clickedCancelButton() method
* 
* This method is triggered when the user clicked on the 'Cancel' button.
*/
public void clickedCancelButton()
{
    // Load the NewsGrpIconVote frame
    NewsGrpIconVote MyMenu = new NewsGrpIconVote();
    MyMenu.show();
    this.hide();
    this.dispose();
}

} // ClickedCancelButton() method

} // VoteWin2 class
APPENDIX B: SOURCE CODE [CLIENT SIDE]

NewsGrpIconVote.java

/*===================================================================
Program Name : NewsGrpIconVote.java
Purpose : Differences between this program and
NewsGrpIconVote.java is that this program include
the use of the Visual Cafe Classes which implementes
the use of ImageLists. The ImageLists allows the
user to select an image that represents their
reaction of their reply to a certain Topic of
discussions.

===================================================================*/
import java.applet.*;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.*;

// Uses Symantec Visual cafe Classes
import symantec.itools.awt.ImageListBox;
import symantec.itools.multimedia.ImageViewer;

public class NewsGrpIconVote extends Frame
{
  // Program Declaration
  /*=============================*/
  // Variables needed to retrieve the smilies location
  /*================================*/
  private String ImageLocation =
                             "http://krakatoa.fste.ac.cowan.edu.au/
                              Images/Thesis_Images/New_Color/";

  private URL ImageURL;
  // =========================================================================
  // Maximum number of Topic names allowed
  final int MaxNumbers = 200;
  // Using port 8003 to create the discussion name
  final int NewsGrpPort = 8003;
  // The name of the server
  String Hostname = "krakatoa.fste.ac.cowan.edu.au";

  // Inform the program the operation the user request for
  char User_Action;
  // Get the data from the server through the socket
}

217
DataInputStream FromServer = null;

// Output the data to the server, through the socket, to update
// necessary files

PrintStream ToServer = null;

// Instantiate the socket object to use in the program

Socket NewsGrpSocket;

// Visual Cafe Classes

// The discussion list that allows an image to be displayed

ImageListBox ReplyList;

// Declare all objects components that we are going to use

Menu FileMenu; // Creating the File Menu object
Menu HelpMenu; // Creating the Help Menu object

List TopicList; // The topic list is used to
display topic names created by

Label TopicName_Label; // Topic name label
Label Reply_Label; // Discussion name label
Label Author_Label; // Author label
Label Date_Label; // Date label

Label Total_Label; // Total number of participants

Label Voted_Label; // Total number who voted label
Label NotVoted_Label; // Total number not voted label
Label InFavour_Label; // Number in favour of discussion
Label Against_Label; // Number against the discussion
Label ReplyContent_Label; // The discussion content label

Label SystemMsg_Label; // The system error message label

TextField Author_Field; // Contains the name of the author
TextField Date_Field; // Contains the date of the author
TextField Total_Field; // Contains total number of
// participants

TextField Voted_Field; // Contains total number of people who
// voted
TextField NotVoted_Field; // Contains total number of
// people who have not voted

TextField InFavour_Field; // Contains number of people in
// favour of the discussion
TextField Against_Field; // Contains number of people
// against the discussion

TextField SystemMsg_Field; // Contains the system error
// messages

TextArea ReplyContent_Field; // Contains the comments made by
APPENDIX B : SOURCE CODE [CLIENT SIDE]

Button Create_Topic_Button; // Allow user to create a topic
// name

Button Create_Discussion_Button; // Allow user to create a
// discussion

Button Submit_Opinion_Button; // Submitting opinion for
// the first time

Button Change_Opinion_Button; // Submitting opinion for
// the 'n' time

// Use to locate the Image name associated with the discussion
// name selected by the user

Properties ImageTable = new Properties();

Image MyImage; // Contains the smiley image object

// The Reply_Name & Image_Name variable below is declared global
// because the result need to be retained for the program to be
// able to send the information to the server to search for the
// correct discussion name

String Reply_Name = "", // Contains discussion name
// selected by user in ReplyList
Image_Name = ""; // Contains the correct smiley
// image associated with the
// discussion name

// These data are needed for the opinion polling System
// We store them a string valuable since they have to be
// passed on to the "Vote_Win2" class for calculation purposes

String InitialVote = "Neutral", // Initial value
// when the
// discussion file
// is first created

VoteStatus = "", // Tell us if user
// is giving the
// opinion for the
// First time or
// Second time

Discussion = "", // Discussion contents

Reply_FName = "", // The FILENAME that
// stores this
// discussion

DataTo_OpinionPoll = ""; // Data necessary for
// opinion polling
// system to update
// the discussion File
// in the server

int Total_Number = 0, // Total number of people
// participating
APPENDIX B : SOURCE CODE [CLIENT SIDE]

Voted = 0, // Number of people who have submitted their opinion
Not_Voted = 0, // Number of people who have not submit their opinion
InFavour = 0, // Number of people in favour of the discussion
Not_InFavour = 0; // Number of people against the discussion

// an array holding all the Topic names retrieved from
// Topic_List.txt file
String TName[] = new String[MaxNumbers];

public NewsGrpIconVote() {
    // The title of the NewGrpIconVote frame
    super("User Conferencing System");
    // Create the Menu Bar to contain the File and Help menus
    MenuBar mbar = new MenuBar();
    // Create and add the File menu object to the menu bar
    FileMenu = new Menu("&File");
    FileMenu.add(new MenuItem("Create &Topic"));
    FileMenu.add(new MenuItem("Create &Discussion"));
    FileMenu.addSeparator();
    FileMenu.add(new MenuItem("E&xit"));
    mbar.add(FileMenu);
    // Create and add the Help menu to the menu bar
    HelpMenu = new Menu("&Help");
    HelpMenu.add(new MenuItem("&About"));
    mbar.add(HelpMenu);
    // Initialise the Menu bar
    setMenuBar(mbar);
    // Initialise all AWT (Abstract Window Toolkit) used
    TopicName_Label = new Label("Topic Name");
    TopicList = new List();
    Reply_Label = new Label("Discussion Title");
    Author_Label = new Label("Author");
    Author_Field = new TextField(26);
APPENDIX B : SOURCE CODE [CLIENT SIDE]

Date_Label = new Label("Response Date");
Date_Field = new TextField(26);

// Opinion poll objects
Total_Label = new Label("Total Participants");
Total_Field = new TextField(10);
Voted_Label = new Label("# Submitted Opinion");
Voted_Field = new TextField(9);
NotVoted_Label = new Label("# NOT Submitted Opinion");
NotVoted_Field = new TextField(9);
InFavour_Label = new Label("# In Favour");
InFavour_Field = new TextField(10);
Against_Label = new Label("# Against");
Against_Field = new TextField(9);

// Discussion & System Message objects
ReplyContent_Label = new Label("Discussion");
ReplyContent_Field = new TextArea(15,15);
SystemMsg_Label = new Label("System Messages");
SystemMsg_Field = new TextField(73);

// Buttons Objects
Create_Topic_Button = new Button("Create Topic");
Create_Discussion_Button = new Button("Create Discussion");
Submit_Opinion_Button = new Button("Submit Opinion");
Change_Opinion_Button = new Button("Change Opinion");

// Declare the NewsGroup Main Layout
setLayout(null);
addNotify();
resize(insets().left + insets().right + 644, insets().top + insets().bottom + 462);

// Initialise & create the TopicList, ReplyList & their respective Labels
Create_TopicList();
Create_ReplyList();

// Updates TopicList with Topic names from the server when this class is first initialised. Note that the topic list have to be created first before populating it.
UpdateTopicListCall();

// Add all necessary AWT objects to the Frame
add(TopicName_Label);
TopicName_Label.reshape(insets().left + 7, insets().top + 0, 133, 13);
add(TopicList);
TopicList.reshape(insets().left + 27, insets().top + 25, 282, 111);

add(Reply_Label);
Reply_Label.reshape(insets().left + 315, insets().top + 0, 133, 13);

add(Author_Label);
Author_Label.reshape(insets().left + 7, insets().top + 143, 119, 13);

add(Author_Field);
Author_Field.reshape(insets().left + 21, insets().top + 159, 217, 20);

Author_Field.setEditable(false);
add(Date_Label);
add(Date_Field);

Date_Field.setEditable(false);

// Add the opinion polling objects to the Frame
add(Total_Label);
Total_Label.reshape(insets().left + 7, insets().top + 184, 119, 13);

add(Total_Field);
Total_Field.reshape(insets().left + 21, insets().top + 198, 84, 26);

Total_Field.setEditable(false);
add(Voted_Label);
Voted_Label.reshape(insets().left + 149, insets().top + 184, 131, 11);

add(Voted_Field);
Voted_Field.reshape(insets().left + 161, insets().top + 198, 77, 26);

Voted_Field.setEditable(false);
add(NotVoted_Label);
NotVoted_Label.reshape(insets().left + 287, insets().top + 184, 175, 11);

add(NotVoted_Field);
NotVoted_Field.reshape(insets().left + 294, insets().top + 198, 77, 26);

NotVoted_Field.setEditable(false);

add(InFavour_Label);
InFavour_Label.reshape(insets().left + 7, insets().top + 224, 161, 13);
add(InFavour_Field);
InFavour_Field.reshape(insets().left + 21,insets().top + 241,84,19);
InFavour_Field.setEditable(false);

add(Against_Label);
Against_Label.reshape(insets().left + 287,insets().top + 224,140,13);

add(Against_Field);
Against_Field.reshape(insets().left + 294,insets().top + 241,77,19);
Against_Field.setEditable(false);

// Now add the discussion & system message Fields to the Frame
add(ReplyContent_Label);
ReplyContent_Label.reshape(insets().left + 7,insets().top + 263,112,13);

add(ReplyContent_Field);
ReplyContent_Field.reshape(insets().left + 21,insets().top + 280,609,117);
ReplyContent_Field.setEditable(false);

add(SystemMsg_Label);
SystemMsg_Label.reshape(insets().left + 7,insets().top + 398,168,13);

add(SystemMsg_Field);
SystemMsg_Field.reshape(insets().left + 21,insets().top + 414,609,20);
SystemMsg_Field.setEditable(false);
SystemMsg_Field.setForeground(Color.black);

// Add the buttons to the Frame
add(Create_Topic_Button);
Create_Topic_Button.reshape(insets().left + 516,insets().top + 146,112,26);

add(Create_Discussion_Button);
Create_Discussion_Button.reshape(insets().left + 516,insets().top + 177,112,26);

add(Submit_Opinion_Button);
Submit_Opinion_Button.reshape(insets().left + 516,insets().top + 206,112,26);

add(Change_Opinion_Button);
Change_Opinion_Button.reshape(insets().left + 516,insets().top + 237,112,26);

// Check the "Submit Opinion" & "Change Opinion" buttons,
// when this Frame is first loaded, the button should be
// disable.
    Check_Butons();

    // Set the size of the Create Reply frame
    resize(645, 560);
    show(); // Display the NewsGrpIconVote frame
    setResizable(false); // Disallow user to resize the
    // NewsGrpIconVote frame

} // NewsGrpIconVote() CONSTRUCTOR
public void SYSTEM_MESSAGE(String ErrorMessage)
{
    SystemMsg_Field.setText(ErrorMessage);
}

public void Create.TopicList()
{
    // Create topic name label
    TopicName_Label = new Label("Topic Of Discussion");
    add(TopicName_Label);
    TopicName_Label.reshape(insets().left + 17, insets().top + 5, 110, 13);

    // Now create the topic list
    TopicList = new List();
    add(TopicList);
    TopicList.select(0); // Select the 1st Item in the Topic
    // List as the default
    TopicList.reshape(insets().left + 24, insets().top + 24, 190, 250);  // 250
}

public void Create.ReplyList()
{
    // Create the reply name label
    Reply_Label = new Label("Discussions");
    add(Reply_Label);
    Reply_Label.reshape(insets().left + 285, insets().top + 590, 13);

    // Now create the discussion list
    ReplyList = new ImageListBox();
    ReplyList.reshape(insets().left + 328, insets().top + 25298, 111);  // 25298
add(ReplyList);

// Set the background of the discussion list to be white
// background

ReplyList.setBackground(Color.white);
ReplyList.setComboBoxMode(false);
ReplyList.setRowsToShow(0);
ReplyList.setShowHorizontalScroll(true);
ReplyList.setShowVerticalScroll(true);

ReplyList.setBorderType(syntec.tools.awt.ImageListBox.BORDE_REGULAR);
ReplyList.setCellBorders(false);
ReplyList.setDefaultEnabledTextColor(new Color(0));
ReplyList.setDisabledTextColor(new Color(0));
ReplyList.setTextBoxFont(new java.awt.Font("TimesRoman",java.awt.Font.PLAIN,15));

} // Create_ReplyList() method

/*===============================================================*/
/*
* Check_Buttons() Method
*---------------------------------------------------------------*/
/* This method disable the Submit Opinion & Change Opinion
* buttons when the frame is first loaded. To do this checking, we
* require one of the field on the main interface to test.
* Since all the opinion poll results have to be displayed
* together anyway. If the Total field is not empty, we know that
* the user has selected a discussion name to submit an opinion.
*---------------------------------------------------------------*/

public void CheckButtons()
{

// Contains the content of the Total number of participants
// field for testing purposes.
String TotalField = "";

// Retrieve data on the total number of participants.
TotalField = Total_Field.getText();
// If the field is empty, disable both the polling system 
// buttons since data has not been retrieved. Otherwise, 
// enable both the buttons.

if (TotalField.equals(""))
{
    // Disable the two buttons first
    Submit_Opinion_Button.disable();
    Change_Opinion_Button.disable();
} else
{
    // Otherwise enable the buttons
    Submit_Opinion_Button.enable();
    Change_Opinion_Button.enable();
} // else

public void Manage.Buttons()
{
    // Used to store the total number of participants
    String TotalField = "";

    // Retrieve data on the total number of participants.
    TotalField = Total_Field.getText();

    // Data retrieved from the NewsGrplconvote frame are string 
    // data thus we have to convert them to integer value before 
    // comparing them as numbers.
    int Total = Integer.parseInt(TotalField);
    int Voted = Integer.parseInt(Voted_Field.getText());
    int NotVoted = Integer.parseInt(NotVoted_Field.getText());
APPENDIX B: SOURCE CODE [CLIENT SIDE]

// Check if data are updated to the Total field. If the field is empty, disable the polling buttons.
if (TotalField.equals(""))
{
    // If the field is empty, disable both buttons
    Submit_Opinion_Button.disable();
    Change_Opinion_Button.disable();
}
else
{
    // Disable the "Submit Opinion" button if total number of people is equal to the number of people voted.
    // If it’s equal, then everyone has voted so display an error message.
    if (Total == Voted)
    {
        SYSTEM_MESSAGE("You have already voted. Click Change_Opinion to change your vote");

        // Disable the "Submit Opinion" button so that the user cannot clicked it.
        Submit_Opinion_Button.disable();
        Change_Opinion_Button.enable();
    } // if
else
{
    // Disable the "Change Opinion" button if the number of people who have not voted is equal to the total number of people taking part. In this case, we disallow anyone to change their opinion since no one has voted
    if (NotVoted == Total)
    {
        SYSTEM_MESSAGE("No one has voted yet, you cannot change your vote.");

        // Disable the "Change Opinion" button so that the user cannot click it
        Change_Opinion_Button.disable();
        Submit_Opinion_Button.enable();
    } // if
else
    // When the users are allowed to submit an
    // opinion or change their previous one.
    if (Voted < Total)
    {
        // Enable both buttons
        Submit_Opinion_Button.enable();
        Change_Opinion_Button.enable();
    } // if
} // else

} // Manage_Butons() method

/*==============================================*/
/* UpdateTopicListCall() Method*/
/*-----------------------------------------------------------------*/
/* This command sent the command to the server requesting it to */
/* send the topic names to the client so that the topic list can */
/* be updated.*/
/*=================================================================*/
public void UpdateTopicListCall()
{
    // User action code number '5' is the command to update the
    // topic list
    User_Action = '5'; // Set Current User Action
    ExecuteCommand(); // Connect to the server & retrieve
    // the topic names
} // UpdateTopicListCall() Method

/*=================================================================*/
/* CheckTopicList(String) Method*/
/*=================================================================*/
/* This method checks if the topic list is empty, if it is, the */
/* user is not allowed to create a discussion since there is no */
/* topic name to relate to. Therefore the "Create_Discussion" */
/* Button should be disabled. Otherwise, allow the user to create */
/* a discussion name.*/
/*=================================================================*/
public void CheckTopicList(String TopicOfDiscussion)
{
    // Contains the command to the server to create a discussion
    // name
    String Create_Reply_Command = "Create_Reply",
    TopicName = ""; // Contains the topic
    // name under which
    // the discussion
    // name is created

    // Assign the topic of discussion name to local variable
    TopicName = TopicOfDiscussion;
// If there is nothing in the topic list, then disable
// the "Create_Discussion" button.

if (TopicName == null)
{
    Create_Discussion_Button.disable();
    SystemMsg_Field.setText("Please create a Topic of Discussion First");
}
else
{
    SYSTEM_MESSAGE("Please wait. Loading Discussion Dialog Box ...");

    // If the topic list is not empty then allow the user to
    // create a discussion. Load the Create_Reply4 frame so
    // that the user can create a discussion name.
    Create_Reply4 Reply = new Create_Reply4
    (NewsGrpSocket,Create_Reply_Command,TopicName);
    Reply.show();
    // FIRST hide & dispose parent (NewsGrpIconVote) frame
    this.hide();
    this.dispose();
}

} // else

} // CheckTopicList(String) method

.showMessage("Message", SYSTEM_MESSAGE);

/***********************************************************/
/* UpdateTopicList() Method */
/**************************************************************/
/* This method will update the topic list with the topic names */
/* that is retrieved from the server. */
/**************************************************************/

public void UpdateTopicList(String Updated_TName)
{
    StringTokenizer Tokens;
    String Topic_Names = "";

    // First clear the previous data in the topic list
    TopicList.clear();
// Get updated discussion names that is separated by '#'
// char.
Tokens = new StringTokenizer(Updated_TName, "#");

while (Tokens.hasMoreTokens()) {
    // Get next topic names
    Topic_Names = Tokens.nextToken();
    // If the topic names are not retrieved correctly from
    // the client a null value will be displayed, we do not
    // want to display that on the topic list.
    if (Topic_Names.equals("null") ||
        Topic_Names = null)
    {
        System.out.println("NULL value will not be
                  added to TopicList");
    }
    else
    {
        // Add the topic names to topic list
        TopicList.addItem(Topic_Names);
    }
} // while

// Select the First item in the topic list
TopicList.select(0);

} // UpdateTopicList(String) method

/*============================================================================*/
/* UpdateReplyList(String) Method */
/*============================================================================*/
/* This method will update the discussion list with the discussion*/
/* names that is retrieved from the server. Since the discussion */
/* names retrieved are seperated by ' # ' char, we have to get each */
/* discussion names using Tokens and update the ReplyList with the*/
/* discussion names. The data retrieved from the server tells us */
/* the image name associated with each discussion names created,so*/
/* we must retrieve it too. */
/*============================================================================*/

public void UpdateReplyList(String Updated_RNames)
{
    StringTokenizer Tokens, MyTokens;

    // The Reply_Names string contains the following data string
    // "Discussion name"ImageName"
    String Reply_Names = "",

    // Contains the discussion name excluding the ImageName
    Final_RName = "",
    // Contains the image name necessary for display in the
// ImageList object

ImageName = "",

// Contains the absolute URL to the smiley image.

ModifiedImageName = "",

// The directory where the smiley image resides


// First clear the previous data in the reply list

ReplyList.clear();

// Inform user if the topic name does not contain any discussions.

if (Updated_RNames.equals(""))
{
    SYSTEM_MESSAGE("This Topic DOES NOT contain any discussions.");
}

// Get Updated discussion names that is seperated by '# ' char.

Tokens = new StringTokenizer(Updated_RNames, ",");

while (Tokens.hasMoreTokens())
{
    // Get next discussion names

    Reply_Names = Tokens.nextToken();
    System.out.println("ORIGINAL Reply Name = " + Reply_Names);

    // Seperate the discussion Names from image Name before adding to the discussion list. The discussion names is stored using the following format Discussion name-ImageName (E.g Good_Design-overjoyed.gif), whereby the ImageName contains the smiley that the user wants to portray together with his/her text response. Get the discussion names & image names seperated by "-" char

    MyTokens = new StringTokenizer(Reply_Names, "-");
APPENDIX B : SOURCE CODE [CLIENT SIDE]

// The code below separate the discussion name from the
// image name seperated by "_" char so that the image
// location can be obtain to retrieve the image to be
// display in the ReplyList (an ImageListBox object)

while (MyTokens.hasMoreTokens())
{
    Final_RName = MyTokens.nextToken();

    if (MyTokens.hasMoreTokens())
    {
        String Temp = MyTokens.nextToken();

        // Alert the program if the user did not select
        // any smiley

        if (Temp.equals("NO_DATA"))
        {
            ImageName = "";
            System.out.println("User did not select
Image !!!");
        } //if

        else
        {
            // Otherwise, assign the image name to
            // ImageName variable

            ImageName = Temp;
        } //else
    } //if

    // Put the value into the ImageTable Property Table
    // so that we can associate the correct discussion
    // name to the image name.

    ImageTable.put(Final_RName, ImageName);
} // while

// Now we can add the item & smiley image to discussion
// list. First, get the absolute URL path to the image,
// then add the image to the discussion list.

ModifiedImgName = ImageLocation + ImageName;
addImage(ModifiedImgName, Final_RName);
// Initialise the variables after it has been added to // the list.

Final_RName = "";
ImageName = "";

} // while

// Select the First Item in the discussion list
ReplyList.select(0);

} // UpdateReplyList(String) method

/**=================================================================* /
/** addImage(String, String) Method */
/**=================================================================* /
/** This method retrieved the required images from the URL given. */
/** Since this program uses a frame, the Toolkit.getDefaultToolkit() */
/** method is used. */
**************************************************************************/

class Peer {

public void updateReplyList(String name) {

ReplyList.select(0);
ReplyList.additem(MyImage, name);

} // updateReplyList(String) method

public void addImage(String ImageLocation, String name) {

try {

URL ImageURL;

try {

ImageURL = new URL(ImageLocation);

MyImage = Toolkit.getDefaultToolkit().getImage(ImageURL);

} // try

catch(MalformedURLException e) {

System.out.println(e.toString());
} // catch()

MediaTracker tracker = new MediaTracker(this);

// Add the images to the MediaTracker one at a time
tracker.addImage(MyImage, 0);

try {

// Wait for all the images to be loaded
tracker.waitForAll();
} // try

catch(InterruptedException e) {

System.out.println("Not All images are loaded " + e);
} // catch

// Finally, add the images to the discussion list
ReplyList.additem(MyImage, name);

} // addImage(String, String) method
/**
 * action() Method
 */
/**
 * Handles all the menu events
 */
public boolean action(Event event, Object arg) {
    // Handles the Menu event
    if (event.target instanceof MenuItem)
    {
        String label = (String)arg;
        if (label.equalsIgnoreCase("Create &Topic"))
        {
            // User action code number '1' is the command to
            // create the topic name
            User_Action = '1'; // Set Current User Action
            // Inform the user that the system is loading up the
            // Topic name frame.
            SYSTEM_MESSAGE("Loading Topic Dialog Box. Please
            be patient ..... ");

            ExecuteCommand(); // Connect to the server
            // and send the data
            returnCommand();
        } // if
        else
        if (label.equalsIgnoreCase("Create &Discussion"))
        {
            // User action code number '2' is the command to
            // create the discussion name
            User_Action = '2'; // Set Current User Action
            System.out.println("Create Reply option
            selected");

            // Inform the user that the system is loading up
            // the discussion frame.
            SYSTEM_MESSAGE("Loading Discussion Dialog Box.
            Please be patient ..... ");

            // Connect To Server and send data
            ExecuteCommand();
            return true;
        } // if
        else
        if (label.equalsIgnoreCase("&About"))
        {
            // Inform the user that the system is loading up
            // the Help frame.
            SYSTEM_MESSAGE("Loading Help Dialog Box.
            Please be patient ..... ");

            // Connect To Server and send data
            ExecuteCommand();
            return true;
        } // if
        else
        if (label.equalsIgnoreCase("&Options"))
        {
            // Inform the user that the system is loading up
            // the Options frame.
            SYSTEM_MESSAGE("Loading Options Dialog Box.
            Please be patient ..... ");

            // Connect To Server and send data
            ExecuteCommand();
            return true;
        } // if
        else
        if (label.equalsIgnoreCase("&Help"))
        {
            // Inform the user that the system is loading up
            // the Help frame.
            SYSTEM_MESSAGE("Loading Help Dialog Box.
            Please be patient ..... ");

            // Connect To Server and send data
            ExecuteCommand();
            return true;
        } // if
        else
        if (label.equalsIgnoreCase("&About"))
        {
            // Inform the user that the system is loading up
            // the About frame.
            SYSTEM_MESSAGE("Loading About Dialog Box.
            Please be patient ..... ");

            // Connect To Server and send data
            ExecuteCommand();
            return true;
        } // if
    }

    return false;
}
APPENDIX B: SOURCE CODE [CLIENT SIDE]

```java
{ // Load the About dialog box and inform the // user of current action.

    SYSTEM_MESSAGE("Loading About Dialog Box. Please be patient .....");
    selectedAbout();
    // Inform the user that the About dialog box // is loaded.
    SYSTEM_MESSAGE("About Dialog Box has been loaded.");

    return true;
} // if

else

    if (label.equalsIgnoreCase("E&xit"))
    {
        // Inform the user that the system is // loading up the Exit dialog box.
        SYSTEM_MESSAGE("Loading Exit Dialog Box. Please be patient .....");
        selectedExit();
        // Inform the user that the Exit dialog // box is loaded.
        SYSTEM_MESSAGE("Exit Dialog Box has been loaded");

        return true;
    } // if
} // Handles MenuItem event
```
// Handling the List event
if (event.target instanceof List) {
  // Handles topic list
  if (event.target == TopicList) {
    // User action code number '3' is the command to
    // update the topic list
    User_Action = '3'; // Set Current User Action
    // Clear all the fields to display next data.
    Author_Field.setText("");
    Date_Field.setText("");
    Total_Field.setText("");
    Voted_Field.setText("");
    NotVoted_Field.setText("");
    InFavour_Field.setText("");
    Against_Field.setText("");
    ReplyContent_Field.setText("");
    System_Msg_Field.setText("");
    // Connect to the server and send data
    ExecuteCommand();
  } // if instanceof List

  // Handling the Discussion list
  if (event.target instanceof ImageListBox) {
    if (event.target == ReplyList) {
      // Get the discussion name when the user clicked on
      // the ReplyList (which is the discussion list) Match
      // the selected discussion name in ImageTable
      // property table to find out the related image
      // name. This is necessary for the program to locate
      // the matching discussion name in the server.
      Reply_Name = ReplyList.getItem(ReplyList
                               .getSelectedIndex());

      // Once user has selected a discussion name, use the
      // name to locate the associated image Name in
      // ImageTable
      Image_Name = FindImgName(Reply_Name);

      // User action code number '4' is the command to
      // update the discussion data on the main frame.
      User_Action = '4'; // Set Current User Action
      // Connect to the server and send data
      ExecuteCommand();
    } // if (event.target == ReplyList)
  } // if (event.target instanceof ImageListBox)
return true;
// if
} // if instanceof ImageListBox

// Handling Button: Objects
if (event.target instanceof Button)
{
    if (event.arg.equals("Submit Opinion"))
    {
        // Inform the user that the system is loading up the
        // Opinion polling system.
        SYSTEM_MESSAGE("Loading OPINION POLL Dialog Box.
                        Please be patient .....!");

        // Since user clicked the "Submit Opinion" button,
        // this indicates that this is the FIRST TIME user
        // is giving opinion
        VoteStatus = "First_Time";

        // Get the topic name to be displayed in the Opinion
        // Poll frame
        String Selected_TName = TopicList.getSelectedItem();

        // When sending data to VoteWin2 class, send the
        // discussion data as well so that the
        // discussion file can be updated to reflect new
        // changes.
        VoteWin2 OpinionPoll;
        OpinionPoll = new VoteWin2(Selected_TName,
                                   DataTo_OpinionPoll, Discussion,
                                   Reply_FName, Total_Number,
                                   Voted, Not_Voted, InFavour,
                                   Not_InFavour, VoteStatus, InitialVote);

        OpinionPoll.show();

        // Dispose and hide the window once data is handled
        this.hide();
        this.dispose();

        return true;
    } // if "Submit Opinion"

else
{
    if (event.arg.equals("Change Opinion"))
    {
        // Inform the user that the system is loading up
        // the change Opinion frame.
        SYSTEM_MESSAGE("Loading OPINION POLL Dialog Box.
                        Please be patient .....!");

        // Dispose and hide the window once data is handled
        this.hide();
        this.dispose();

        return true;
    } // if "Change Opinion"

} // if instanceof ImageListBox

// Handling Button: Objects
if (event.target instanceof Button)
{
// Since user clicked the "Change Opinion"
// button, this indicates that this is NOT the
// first time the user is submitting an opinion

VoteStatus = "Second_Time";

// Get the Topic name to be displayed in the
// Opinion Poll box

String Selected_TName = TopicList.getSelectedItem();

Change_OpinionBox ValidateBox;

ValidateBox = new Change_OpinionBox
    (Selected_TName,
    DataTo_OpinionPoll, Discussion,
    Reply_FName, Total_Number,
    Voted, Not_Voted, InFavour,
    Not_InFavour, VoteStatus);

ValidateBox.show();

// Dispose & hide the window once data is handled

this.hide();
this.dispose();
return true;

} // if "Change Opinion"

else

if (event.arg.equals("Create Topic"))
{

    // User action code number '1' is the
    // command to create the topic name

    // Set current User Action

    User_Action = '1';

    System.out.println("Create Topic option
    selected");
// Inform the user that the system is
// loading up the create topic frame.
SYSTEM_MESSAGE("Loading Topic Dialog Box.
   Please be patient ..... ");
// Connect to the server and send data
ExecuteCommand();
return true;
} // if Create Topic
else
    if (event.arg.equals("Create Discussion"))
        {
            // User action code number '2' is the
            // command to create the discussion name
            // Set Current User Action
            User_Action = '2';
            System.out.println("Create Reply option
selected");
            // Inform the user that the system is
            // loading up the Discussion frame.
            SYSTEM_MESSAGE("Loading Discussion Dialog
Box. Please be patient ..... ");
            // Connect to the server and send data
            ExecuteCommand();
            return true;
        } // if "Create Discussion"
} // if

// If the event received by handleEvent is not the
// appropriate event, call the superclass handleEvent to send
// the event to the appropriate event handling method.
return super.action(event, arg);
} // action
public boolean handleEvent(Event event)
{
    if (event.id == Event.WINDOW_DESTROY)
    {
        // Inform the user that the system is loading up the
        // Exit dialog box.
        SYSTEM_MESSAGE("Loading Exit Dialog Box. Please be
        patient ......");

        // Prompt user if they wish to quit the program.
        selectedExit();
        return true;
    } // if

    // If the event received by handleEvent is not the
    // appropriate event, call the superclass handleEvent to
    // send the event to the appropriate event handling method.
    return super.handleEvent(event);
}

/*=======================================================================
* FindImgName(String) method
/*=======================================================================
/* This method will locate the Image name associated with the */
/* discussion name selected by the user in the discussion list. */
/* The image name is necessary because it will be combined into */
/* the string format "ReplyName-ImageName" */
/* (E.g Excellent_Design-Smiley.gif) so that they can be sent to */
/* the server to locate the correct discussion name. */
/* NOTE: Reason for this is because the discussion name in both */
/* the Reply_List.txt & Topic?.txt files have been saved */
/* in the "ReplyName-ImageName" format */
/*=======================================================================

public String FindImgName(String RNameToSearch)
{
    String ImageFound = "";

    // Check if discussion name given is empty
    if (RNameToSearch.equals("")
    {
        System.out.println("ERROR !!! Discussion name selected
        by user is not available !");
    } // if

    else
    {
        ImageFound = ImageTable.getProperty(RNameToSearch,
        "ERROR IMAGE NAME CANNOT BE FOUND !!");
    } // else

241
return ImageFound;
    } // FindImageName(String) method

    /*===========================================================*/
    /* selectedAbout() Method */
    /* This method is triggered when the user selected the About menu */
    /* option. */
    /*===========================================================*/
    public void selectedAbout()
    {
        AboutBox CallAboutBox;

        CallAboutBox = new AboutBox(this);

        CallAboutBox.show();
    } // selectedAbout() method

    /*===========================================================*/
    /* selectedExit() Method */
    /* This method is triggered when the user select the Exit menu */
    /* option. Note that the program will prompt the user before it */
    /* attempts to terminate the program. */
    /*===========================================================*/
    public void selectedExit()
    {
        QuitBox CallQuitBox;

        CallQuitBox = new QuitBox(this);

        CallQuitBox.show();
    } // selectedExit() method
/**
 * Connect() Method
 */
// This method establish the connection between the client and the
// server using the port that is specified at the beginning of the
// program.
/**
 * public void Connect() throws IOException
 * {
 *     NewsGrpSocket = new Socket(Hostname,NewsGrpPort);
 *     FromServer = new DataInputStream(NewsGrpSocket.getInputStream());
 *     ToServer = new PrintStream(new DataOutputStream(NewsGrpSocket.getOutputStream()));
 * } // connect() method
 */

/**
 * Disconnect() Method
 */
// This method disconnect the socket connection between the client
// and the server.
/**
 * public void Disconnect()
 * {
 *     try
 *     {
 *         NewsGrpSocket.close();
 *     } // try
 *     catch(IOException e)
 *     {
 *         System.out.println("Error Closing the Socket" + e.toString());
 *     }
 * } // Disconnect() method
 */

/**
 * show() Method
 */
// This method display the main interface for interaction with the
// user. Note that the move() command also set the position of the
// program on the screen.
/**
 * public synchronized void show()
 * {
 *     move(50, 10);
 *     super.show();
 * } // show() Method
 */

/**
 * ExecuteCommand() Method
 */
// This method is one of the most important in the program. It
// sends the client request (Command) to the server. Thus the

243
/** server knows what task to carry out. This method does not. */
/** retrieve the topic names or discussion names to update the */
/** respective list. This is done using the UpdateTopicList() method. */

public void ExecuteCommand() {
    String DataFromSvr = "", // Data sent to server from
                    // client
    TName_Selected = "", // Topic name selected by
                    // user in the topic list
    TList_String = "", // Command to server + Topic
                    // name selected in topic list
    RName_Selected = "", // Discussion name selected
                    // by user in the discussion
                    // list

    // Command to server + discussion name selected in
    // the discussion list
    RList_String = "",

    // Discussion names retrieved from the server
    Retrieved_RNames = "",

    // Discussion data and its contents
    Reply_Data = "",

    // Topic Names to update topic list
    TopicList_Data = "",

    // Command String to inform server of requested
    // operation
    Create_Topic = "Create_Topic",
    Create_Sent = "Create_Sent",
    TList_Selected = "TList_Selected",
    RList_Selected = "RList_Selected",
    Update_TopicList = "Update_TopicList";
try {
    Connect();  // Request for a connection
    while ((DataFromSvr = FromServer.readLine()) != null) {
        // If the 'Welcome' message is received, the server
        // is ready
        if (DataFromSvr.startsWith("Welcome")) {
            // Determine current User Action
            System.out.println("User Action = "
                                + User_Action);
            switch(User_Action) {
                case '1':  // Create the topic name
                    SYSTEM_MESSAGE("Please wait. Loading
                                    Topic Dialog Box .......");
                    // Call Create_Topic class. Sending
                    // Command + Topic Name entered by
                    // user to Server
                    Create_Topic4 Topic = new Create_Topic4
                                    (NewsGrpSocket,Create_Topic);
                    Topic.show();
                    // FIRST hide & dispose parent frame
                    this.hide();
                    this.dispose();
                    break;
                case '2':  // Create the discussion name
                    SYSTEM_MESSAGE("Please wait. Loading
                                    Discussion Dialog Box .......");
                    // Get discussion name from the User
                    String Topic_Name = TopicList
                                        .getSelecteditem();
                    System.out.println("Item Selected in
                                         TopicList = " + Topic_Name);
                    // After the List has been updated,
                    // check if the Topic list is empty.
                    // If it is, then the user are not
                    // allowed to create a discussion. The
                    // reason why the checking is done
                    // here is because the
                    // "Create_Discussion_Button" has to
                    // be created first for the
                    // CheckTopicList() method to access it.
                    CheckTopicList(Topic_Name);
                    break;
            }
        }
    }
    // Update the discussion list of the topic
// name selected

case '3':

    SYSTEM_MESSAGE("Please wait, loading
        Discussions Titles ..... ");

    // Reset the INFAVOUR & AGAINST fields to
    // neutral when the TopicList is selected
    // again. This code implements the color
    // scheme used in the polling system.

    Infavour_Field.setBackground
        (Color.lightGray);
    Against_Field.setBackground
        (Color.lightGray);

    TName_Selected = TopicList
        .getSelectedItem();

    TList_String = TList_Selected + "" +
        TName_Selected;

    // Send the data to the server.
    ToServer.println(TList_String);

    // Assigned the data from server to
    // local variable
    Retrieved_RNames = FromServer.readLine();

    // Update the ReplyList with discussion
    // names.

    // Clear the ReplyList first
    ReplyList.clear();
    UpdateReplyList(Retrieved_RNames);

    // Inform server that work is done.
    ToServer.println("End Work");

    // Lastly, check that if the ReplyList
    // has items in it. Change the remarks
    // made earlier in SystemMsg_Field.

    if (ReplyList.getSelectedItem() == null)
        {
            SYSTEM_MESSAGE("Loading Discussion
                Titles ..... ");
        } // if

    else
        {
            SYSTEM_MESSAGE(" ");
            SYSTEM_MESSAGE("Discussion Titles
                Loaded.");
        } // else

    break;

    // Load the discussion data of the
// discussion name selected.
case '4':
    SYSTEM_MESSAGE("Please wait, loading
discussions ..... ");

    // Assign discussion name selected by //
    // user from Global to Local variable //
    // to be sent to server
    RName_Selected = Reply_Name + "-
                    * Image_Name;

    RList_String = RList_Selected + ""
                   + RName_Selected;

    // Send the data to the server
    ToServer.println(RList_String);

    // Check that FromServer.readLine()
    // IS!=NULL
    Reply_Data = FromServer.readLine();

    // Since discussion Data comes in one
    // string, we have to divide them to
    // their respective position for
    // display. Organize the data and
    // inform user that the work is done.
    Organize_RData.(Reply_Data);
    ToServer.println("End Work");

    // If one of the fields contains data,
    // that means that the data has been
    // loaded, so remove the remarks from
    // the System_Msg_Field.
    SYSTEM_MESSAGE("Discussions loaded.");
    break;
APPENDIX B: SOURCE CODE [CLIENT SIDE]

// Update the topic list

case '5':
    // Send the command to the server.
    ToServer.println(Update_TopicList);
    // Check that FromServer.readLine()
    // is != NULL
    TopicList_Data = FromServer.readLine();
    // Since discussion data comes in one
    // string, we have to divide them to
    // their respective position for
    // display. Update the topic list and
    // inform the server that work is done.
    UpdateTopicList(TopicList_Data);
    ToServer.println("End Work");
    break;

default:
    System.out.println("No Action made by
    User");
    break;

} // if // switch

} // try

catch(IOException e)
{
    System.out.println("Error connecting to the Server
    \n" + e.toString());
}
Appendix B: Source Code [Client Side]

```java
} // catch
finally
{
    Disconnect();
    // Initialise the User action for next entry
    User_Action = "";
} // finally

} // ExecuteCommand() method

/**************************************************************************/
/** Organize_RData() Method *
/**************************************************************************/
 /** The discussion data sent by the client comes as ONE string. They are divided by the '#' char e.g. "TName#RName#Date". This method divides the string into "logical" chunks so that the discussion data can be displayed to its relevant position for the user to understand. */
/** NOTE : The User's Reply is further modified to replace the '*' char with the '\n' char. */
/**************************************************************************/

public void Organize_RData(String Reply_Data) {
    // To Divide discussion data String into logical chunks
    StringTokenizer Tokens;
    // Maximum number of discussion data to store
    int MaxNumbers = 9;
    // Used to detect fields with no data
    String No_Data_Tag = "NO DATA",
    // Contains the Author's Name
    User_Field = "",
    // Contains the discussion date
    RDate_Field = "",
    // Contains the actual discussion edited by user
    Reply_Field = "",
    // Contains the program's Modified discussion for display purposes. These data is needed for the opinion polling system. It is stored as a single string since they have to be passed on to the Vote_Win2 class for calculation purposes
    Modified_Repl...```
Reply_FileName = ":",
// Total number of people participating
Total = ":",
// Number of people who have submitted their
// opinion
Number_Voted = ":",
// Number of people who have not submitted
// their opinion
Number_Not_Voted = ":",
// Number of people in favour of the
// discussion
Number_InFavour = ":",
// Number of people against the discussion
Number_Not_InFavour = ":";
String ReplyInfo[] = new String[MaxNumbers];
if (Reply_Data != null)
{
    try
    {
        Tokens = new StringTokenizer(Reply_Data, ":")
        for (int i=0; i<MaxNumbers; i++)
        {
            if (Tokens.hasMoreTokens())
            {
                ReplyInfo[i] = Tokens.nextToken();
                System.out.println("ReplyInfo = " + ReplyInfo[i]);
            }
        }  // if
    }  // for

    // Updates all TextFields with Respective Data
// Set the Fields to be displayed

User_Field = ReplyInfo[0];
RDate_Field = ReplyInfo[1];
Reply_Field = ReplyInfo[2];
Reply_FileName = ReplyInfo[3];
Total = ReplyInfo[4];
Number_Voted = ReplyInfo[5];
Number_Not_Voted = ReplyInfo[6];
Number_InFavour = ReplyInfo[7];
Number_Not_InFavour = ReplyInfo[8];

// Assign data for the opinion poll to global
// variables so that they can be passed to the
// VoteWin2 class for calculation. The
// Integer.parseInt(String) is used to convert a
// string object to an integer object.

Discussion = Reply_Field;
Reply_FName = Reply_FileName;
Total_Number= Integer.parseInt(Total);
Voted = Integer.parseInt(Number_Voted);
Not_Voted = Integer.parseInt(Number_Not_Voted);
InFavour = Integer.parseInt(Number_InFavour);
Not_InFavour= Integer.parseInt
    (Number_Not_InFavour);

// Arrange all discussion data into a single string
// so that they can be updated to the discussion
// file to reflect the Opinion Poll result

DataTo_OpinionPoll = User_Field + "#"
    + RDate_Field;

// Checks the Author's field

if (User_Field.equals(No_Data_Tag))
    {
        Author_Field.setText(" ");
    } // if
else
    {
        Author_Field.setText(User_Field);
    } // else

// Checks the Date's field

if (RDate_Field.equals(No_Data_Tag))
    {
        Date_Field.setText(" ");
    } // if
else
    {
        Date_Field.setText(RDate_Field);
    } // else
APPENDIX B: SOURCE CODE [CLIENT SIDE]

// Set the opinion poll result. No checking is needed since it will definitely contain data
Total_Field.setText(Total);
Voted_Field.setText(Number_Voted);
NotVoted_Field.setText(Number_Not_Voted);
InFavour_Field.setText(Number_InFavour);
Against_Field.setText(Number_Not_InFavour);

// NOTE that the system will set the field to the appropriate color depending on whether the number of people INFAVOUR is greater than people AGAINST. Before we set the appropriate colors for each field, make sure we set the field to the neutral colors first.

InFavour_Field.setBackground(Color.lightGray);
Against_Field.setBackground(Color.lightGray);

if (InFavour > Not_InFavour)
{
    // Set the InFavour field to Green color to indicate that more people are in favour of the discussion
    InFavour_Field.setBackground(Color.green);
} // if
else
{
    // Set the Against field to Red color to indicate that more people are against the discussion
    Against_Field.setBackground(Color.red);
} // else

if (InFavour == Not_InFavour)
{
    // Set the InFavour Field & Against Field to their respective colours to indicate a neutral stand.
    InFavour_Field.setBackground(Color.green);
    Against_Field.setBackground(Color.red);
} // if

// After the result is displayed on the screen, we have to check the buttons again to see if we want to enable it or disable it.
Manage.Buttons();
// Checks the discussion field
if (Reply_Field.equals(No_Data_Tag))
{
    ReplyContent_Field.setText("");
} // if
else
{
    // This program automatically placed the '*'
    // char next to the string when the user presses
    // the 'RETURN' key. Thus we have to convert the
    // '*' to the NEXT LINE system char '\n'
    Modified_Reply = Reply_Field.replace('*','\n');
    ReplyContent_Field.setText(Modified_Reply);
} // else
} // try
catch (NullPointerException e)
{
    System.err.println(e.toString());
} // catch
} // if
)

} // Organize_RData() method

} // NewsGrpiconVote Class
APPENDIX C : SOURCE CODE [SERVER SIDE]

SERVER PROGRAMS

SvrIconVote1.java

/**
 * Program Name : SvrIconVote1.java
 * Purpose
 * 
 * This is the MAIN area used to get the server side of the Client/Server program running.
 * 
 * This class is programmed so that the server can process multiple incoming client sockets at any one time. To allow this to happen, we have to implement a producer/consumer relationship model.
 * 
 * The 3 classes to make this possible are:
 * 
 * [1] StoreClientData Class
 * 
 * This is the PRODUCER class. When the SvrNoIconNoVote1 class detects an incoming socket from the client, it sends the socket to the StoreClientData class so that it can write the socket object to the shared data variable to be accessed by other methods in the HoldData class.
 * 
 * [2] ExecuteIconVote Class
 * 
 * This is the CONSUMER class. The purpose of this class is to consumed the data that is set to the shared data variable in the HoldData class.
 * 
 * This class holds onto the Socket object so that it can sent the required data to the client. After all the data has been sent to the client, the socket will then be closed and allows the shared data in HoldData class to be accessed again.
 * 
 * [3] HoldData Class
 * 
 * This is the MOST IMPORTANT ingredient that will allow the producer/consumer relationship to work.
 * 
 * The key thing about this class is that all the methods in this class are synchronized methods. Furthermore, all methods accessed the same shared data variable and the same conditional flag.
 * 
 * This means that when any one method is accessing the shared
object, the rest of the methods cannot access the shared data. Instead all the threads will go into the "wait" state until they are "notified" by the method who has just released the shared object.

Besides implementing the producer/consumer relationship, this class also checks that all system files are available for use.

```
import java.net.*;
import java.io.*;
import java.util.*;

public class SvrIconvotel extends Thread {
    public static final int MyPort = 8003;
    static ServerSocket Listen_Socket;

    public static void main(String args[]) {
        System.out.println("Check for System Files ....");
        InitProgramFiles Initialise = new InitProgramFiles();
        Initialise.createFiles();
        System.out.println("Server is now running .....");
        try {
            System.out.println("Setting server Port ...");
            while (true) {
                System.out.println("Waiting for connection from Client ...");
                // Wait for a connection. Listen indefinitely for an attempt by a client to connect
                Socket Client_Socket = Listen_Socket.accept();
            }
        } catch (Exception e) {
            System.out.println("Error: ");
        }
    }
}
```
// If a connection is established, send the socket // to the methods that handle the client/server // relationship.

System.out.println("Setting Producer/Consumer relationship ...");

HoldData h = new HoldData();

StoreClientData Client_Connect = new StoreClientData(h, Client_Socket);

ExecuteIconVote Execute = new ExecuteIconVote(h);

Client_Connect.start();
Execute.start();
}

} // while

} // try

catch(IOException e1)
{
    System.out.println("Error Creating Socket ");
    System.err.println(e1.toString());
} // catch

} // main

} // SvrIconVote1 Class
StoreClientData.java

import java.net.*;

class StoreClientData extends Thread {
    Socket Client; // Used to store the Socket Object
    private HoldData pHold;

    public StoreClientData(HoldData h, Socket Client_Socket) {
        Client = Client_Socket;
        pHold = h;
    } // StoreClientData Constructor
/**
 * run() Method
 * This method automatically execute after the socket object has
 * has been assigned to the pHold variable.
 */

public void run()
{
    System.out.println("PRODUCER entered RUNNING state");
    // Set data from client to global variable
    pHold.setSharedData(Client);
    System.out.println("Producer has set shared data to HoldData Object");

    try
    {
        // Sleep for a period of time
        sleep((int) (Math.random() * 3000));
        System.out.println("Producer taking a nap ...");
    } // try

    catch(InterruptedException e3)
    {
        System.err.println("ERROR MSG = " + e3.toString());
    } // catch

} // run() method

} // StoreClientData Class
InitProgramFiles.java

/*==============================================*/
Program Name : InitProgramFiles.java
Purpose : This class initialised all the necessary system files that are used by this program.
/*==============================================*/
import java.net.*;
import java.io.*;

/*==============================================*/
/* InitProgramFiles Class */
/*==============================================*/
/* This program basically creates all the necessary files that are going to be used by the User conferencing program. */
/* They include : Topic_List.txt */
/* Topic_Count.txt */
/* Reply_List.txt */
/* Reply_Count.txt */
/*==============================================*/

class InitProgramFiles
{
    // Used to check if all four system files exists
    File
    Topic_List,
    Reply_List,
    Topic_Count,
    Reply_Count;

    // Output data to the respective system files.
    DataOutputStream TListOutput,
    RListOutput,
    TRReplyOutput;

    FileOutputStream TCountOutput,
    RCountOutput;

    PrintStream
    PrintInput = null;

    // CHANGE THE PATH BELOW TO ALLOCATE FOLDER FOR NEW USERS
    String
    TList_FName =
    "c:/Program files/Netscape/Server/docs/NewsGrpIconVote/
    UserData/Screen Shots/Topic_List.txt",

    RList_FName =
    "c:/Program files/Netscape/Server/docs/NewsGrpIconVote/
    UserData/Screen Shots/Reply_List.txt",

    TCount_FName =
    "c:/Program files/Netscape/Server/docs/NewsGrpIconVote/
    UserData/Screen Shots/Topic_Count.txt", 

    259
public void createFiles()
{
    // Create all the four system files.
    Topic_List = new File(TList_FName);
    Reply_List = new File(RList_FName);
    Topic_Count = new File(TCount_FName);
    Reply_Count = new File(RCount_FName);
    // Create the Topic_List.txt file if it does not exist
    if (Topic_List.exists() == false)
    {  
        create_TListFile();
    } // if
    // Create the Reply_List.txt file if it does not exist
    if (Reply_List.exists() == false)
    {  
        create_RListFile();
    } // if
    // Create the Topic_Count.txt file if it does not exist
    if (Topic_Count.exists() == false)
    {  
        create_TCountFile();
    } // if
    // Create the Reply_Count.txt file if it does not exist
    if (Reply_Count.exists() == false)
    {  
        create_RCountFile();
    } // if

} // createFiles() method

public void create_TListFile()
{
    String Initialise = "";

}
/ Output the empty string to the file

try
|
 TLCListOutput = new DataOutputStream
(new FileOutputStream(TList_FName));

System.out.println("<<Topic_List.txt>> is created !!!");
TLTListOutput.writeUTF(Initialise);
TLTListOutput.close();

} // try

catch(FileNotFoundException el)
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
public void create_RListFile()
{
    String Initialise = "";

    // Output the data to file
    try
    {
        RListOutput = new DataOutputStream(new FileOutputStream(RList_FName));

        System.out.println("<Reply_List.txt> is created !!!");
        RListOutput.writeUTF(Initialise);
        RListOutput.close();
    }
    catch(FileNotFoundException e1)
    {
        System.out.println(RList_FName + " File NOT Found");
        System.exit(1);
    } // catch FileNotFoundException

    catch(IOException e2)
    {
        System.out.println("Error writing to file !!" + e2.toString());
    } // catch

} // create_RListFile() method
public void create_TCountFile()
{
    int File_Value = 0;
    // Output an initial integer value to the file
    try
    {
        TCountOutput = new FileOutputStream(TCount_FName);
        System.out.println("<<Topic_Count.txt>> is created !!!");
        TCountOutput.write(File_Value);
        TCountOutput.close();
    } // try
    catch(FileNotFoundException e1)
    {
        System.out.println(TCount_FName + " File NOT Found");
        System.exit(1);
    } // catch FileNotFoundException
    catch(IOException e2)
    {
        System.out.println("Error writing to file !!!
                        + e2.toString()");
    } // catch
} // create_TCountFile() method
public void create_RCountFile()
{
    int File_Value = 0;
    // Output the initial integer value to the file
    try
    {
        RCountOutput = new FileOutputStream(RCount_FName);
        System.out.println("<<Reply_Count.txt>> is created !!!");
        RCountOutput.write(File_Value);
        RCountOutput.close();
    } // try
    catch (FileNotFoundException e1)
    {
        System.out.println(RCount_FName + " File NOT Found");
        System.exit(1);
    } // catch FileNotFoundException
    catch (IOException e2)
    {
        System.out.println("Error writing to file ::
        + e2.toString());
    } // catch
} // create_RCountFile() method

// InitProgramFiles Class

APPENDIX C: SOURCE CODE [SERVER SIDE]
HoldData.java

/**
  * Program Name : HoldData.java
  * Purpose : This class implements a producer/consumer relationship that allows 2 or more users to send their requests to the server for processing.
  *
  * All the methods in this class are synchronized. This means that when the socket object is being held by the setSharedData() method, the object cannot be accessed by any other methods in the class until the shared "writeable" variable has been set to false, which allows other objects to access it.
  *
  * Bottom line is, when one method is using the object, other methods cannot access it. When any of the method is finished with the object, the "notifyAll()" method is called to tell all other methods that the object is now available to be used.
  *
  * NOTE : The notify() method can also be used but it is a lot tricky to use it so I'm using notifyAll() method for now.
  */

import java.net.*;

/**
  * HoldData Class
  * The purpose of this class is to allow 2 threads to synchronize the access of the same data. This is to enable the client to be able to logged into the NewsGroup program at the same time and send requests to the server for processing without any lost of data.
  */

class HoldData
{
    private Socket ClientSocket; // Used to store the Socket Object
    private boolean Writeable = true;
    private boolean ClientClosed = true;
public synchronized void setSharedData(Socket Client) {
    // If the buffer still holds the object, go to "wait"
    // state until the buffer is empty again.
    while (!Writeable) {
        try {
            wait();
        } // try
        catch (InterruptedException e) {
            System.err.println("Exception " + e.toString());
        } // catch
    } // while

    // Otherwise assign the global data variable the value
    // passed by the server program.
    // Set "Writable" to false to indicate that the buffer
    // contains some data and cannot be overwritten.
    ClientSocket = Client;
    Writeable = false;
    notifyAll();
} // setSharedData(Socket)

public synchronized Socket getSharedData() {
    // Since the buffer is empty, the setSharedData(Socket)
    // has to write to the buffer first. Therefore, we have
    // to go to the "wait" state until buffer is full.
while (Writeable)
{
    try
    {
        wait();
    } // try
    catch (InterruptedException e)
    {
        System.err.println("Exception "+e.toString());
    } // catch
} // while

// We want to just return the socket so that the client
// can use first. We want to disallow the
// setSharedData(Socket) method to set the shared
// "ClientSocket" variable until the PREVIOUS socket has
// been CLOSED.
// Notify all methods that the objects cannot be accessed
// so that all methods can go to the "wait" state.
Writeable = false;
notifyAll();
return ClientSocket;
} // getSharedData() Method

/**************************************************************************/
/**                  CloseSocket(boolean) Method
**************************************************************************/
/**
* This method plays an important role. The purpose of this method*
* is to inform the rest of the methods if the current object can *
* be accessed again. *
* *
* If the Socket is closed, we will allow the setSharedData(Socket) *
* method to set a new socket object to the shared data variable. *
* Otherwise, notify all other methods that the "ClientSocket" *
* variable cannot be accessed. *
**************************************************************************/

public synchronized void CloseSocket(boolean SocketState)
{
    // Assigned the current state of the socket to local
    // variable.
    ClientClosed = SocketState;
    System.out.println("Client Socket terminated ? = "
    + ClientClosed);

    // Once the client socket is terminated, we have to make
    // sure that we allow the setSharedData(Socket) method
    // to write to the shared data again.
    if (ClientClosed == true)
    {
        Writeable = true
        notifyAll();
    } // if
else
{  
    Writeable = false;  
    notifyAll();  
} // else

} // Class HoldData
ExecuteIconVote.java

/***************************************************************************/
Program Name    : ExecuteIconVote.java
Purpose         : This is the CONSUMER part of the Producer/Consumer relationship. The main purpose of this class is to allow multiple clients to send their request simultaneously.

To complete this function, this class makes use of the HoldData class to enable synchronization so that the shared data cannot be modified when another method is accessing it. This allows all the client requests to be processed.

To enable this class to be used, make sure that the following classes are also available. They include:

[1] SvrIconVote1 Class
-------------------------
This is the SERVER program that listens to the client port to check if there are any incoming client requests. If there are, then the server program will pass the socket object to the StoreClientData class so that the shared data variable can be set.

[2] StoreClientData Class
-------------------------
This class is the PRODUCER. It creates threads that pass all the incoming client sockets to the HoldData class.

[3] HoldData Class
-------------------------
This is the main class that initiate the Producer/Consumer relationship. This class uses a shared data variable as well as a shared conditional flags to allow all the methods to be synchronized. This permits all the methods to turn to the 'wait' state if the shared data is used by another method.

stract public class ExecuteIconVote extends Thread
{    // Store the Socket object so that the connection can be made to
    // process data and send back to client.

import java.io.*;
import java.net.*;
import java.util.*;

class ExecuteIconVote extends Thread
{
protected Socket Client;
// Used to retrieve the data from the server through the socket
protected DataInputStream FromClient;
// Output the data to the server, through the socket, to update
// necessary files
protected PrintStream ToClient;
// Declaring a HoldData object so that the methods in HoldData
// can be used
private HoldData cHold;
// Informs the HoldData class that the socket has been
// terminated

boolean SocketClosed = true;
int MaxFiles = 200, // Maximum number of topic
               MaxNumbers = 6, // Number of data items
               // allowed for storing

// The TotalNumber variable is used to calculate the opinion
// poll results. Modify this value depending on how many
// people are taking part in the polling activities.
TotalNumber = 10;

File Topic_List, // Check if Topic List.txt exists
      Topic_Count, // Check if Topic_Count.txt exists
      Reply_Count; // Check if Reply_Count.txt exists

// The program has been designed for the testing data to be
// easily stored and examined. If a new group is required for
// testing then the following paths should be modified to
// reflect the absolute path to the data folder.

// Topic List.txt file name
String TList_FName =
"c:/Program Files/Netscape/Server/docs/NewsGrpIconVote/
 UserData/Screen Shots/Topic_List.txt",

// Reply List.txt File Name
RList_FName =
"c:/Program Files/Netscape/Server/docs/NewsGrpIconVote/
 UserData/Screen Shots/Reply_List.txt",

// Topic Count.txt File Name
TCount_FName =
"c:/Program Files/Netscape/Server/docs/NewsGrpIconVote/
 UserData/Screen Shots/Topic_Count.txt",

// Reply Count.txt File Name
RCount_FName =
"c:/Program Files/Netscape/Server/docs/NewsGrpIconVote/UserData/Screen Shots/Reply_Count.txt",

// Change the FileLocation variable to create/save the system files (Topic?.txt & Reply?.txt) in the correct location.

FileLocation =

"c:/Program Files/Netscape/Server/docs/NewsGrpIconVote/UserData/Screen Shots/";

// Contains all the available topic names
Properties TopicTable = new Properties();

// Updates discussion names to the discussion file
// file (Reply_List.txt)
Properties ReplyListTable = new Properties();

// Updates RListTable with discussion names from the discussion file (Reply_List.txt)
Properties RListTable = new Properties();

// Contains the discussion names in the Topic?.txt file
Properties ReplyTable = new Properties();

// Also contains the discussion names from the Topic?.txt file but searching purposes
Properties TFileReplyTable = new Properties();

// Contains data in this format TFNAME-TNAME where TFNAME is topic filename and TNAME is topic name
// TFNAME = (E.g TopicB.txt) & TNAME = (E.g Interface_Design)
Properties TopicSearchTable = new Properties();

// Contains data in this format RFNAME-RNAME where RFNAME is discussion filename and RNAME is discussion name
// RFNAME = (E.g Reply8.txt) & RNAME = (E.g Good_Design)
Properties ReplySearchTable = new Properties();

/*===================================================================*/
/* This constructor holds the HoldData object and process the */
/* task requested by the client. */
/*===================================================================*/

public ExecuteIconVote(HoldData h)
{
    // Assign a HoldData Object to local variable
    cHold = h;
} // ExecuteIconVote(HoldData) Constructor
This method automatically execute after the HoldData object has been assigned to the chHold variable.

This class is very important because it communicate with the client program to inform it that it is ready and awaits the server's program command. It sends out the "Welcome to NewsGroup Service" message to indicate that it is ready, whereby the server will sends its request for processing.

public void run()
{
    StringTokenizer Tokens;
    String Command = "", // contains the request from the client
    Data = ""; // contains the data sent by the client

    // since the server program serves multiple clients, it is necessary to make sure that only one method get hold of a socket object to process the client request. If the socket is available to use, it will be assigned to the local socket object.
    Client = chHold.getSharedData();

    // if socket is received, try sending the "Welcome ..." message to the client.
    try {
        FromClient = new DataInputStream (Client.getInputStream());
        ToClient = new PrintStream (Client.getOutputStream());
        ToClient.println("Welcome to NewsGroup Service");
        System.out.println("Sending Welcoming Message");
    } // try
    catch (IOException e)
    {
        System.err.println("ERROR MSG = " + e.toString());
    } // try
    catch (IOException e2)
    {
        System.err.println("ERROR MSG = " + e2.toString());
    } // try
} // run
If the client receives the "Welcome ..." message, the client will start transmitting information, the code below is now ready to process the client request accordingly.

```java
try {
    String DataFromClient = "";//Contain data from Client
    while((DataFromClient = FromClient.readLine()) != null) {
        System.out.println("DataFromClient = " + DataFromClient);

        System.out.println("Receiving Message From CLIENT");
        // Remembering that the format of the data string sent from the client is as follows:
        // (Client_Request~Data) This code below separate the command string from the data sent by the client separated by '~' symbol.
        StringTokenizer Tokens = new StringTokenizer(DataFromClient, "~");
        while (Tokens.hasMoreTokens()) {
            Command = Tokens.nextToken();
            System.out.println("The command to Server = " + Command);

            if (Tokens.hasMoreTokens()) {
                Data = Tokens.nextToken();
                System.out.println("Data To Server = " + Data);
            } //if
        } //while
        // Tell client if the request task is done
        if (Command.startsWith("End Work")) {
            ToClient.println("Client work is completed");
            break;
        } // if "End Work"
        else {
            // The client request server to create a topic name
            if (Command.startsWith("Create_Topic")) {
                Create_Topic(Data);//Method to create topic file
                ToClient.println("Topic File is Created");
                break;
            } //if "Create_Topic"
            else {
                // The client request server to create a discussion name
                //...
if (Command.startsWith("Create_Reply")) {
    // Call the method to create the discussion name
    ManageReply_Data(Data);
    System.out.println("Data received from Client = " + Data);
    ToClient.println("Reply Name is Sent");
    break;
} // if "Create_Reply"
else {
    // The client request server to retrieve the discussion name
    if (Command.startsWith("TList_Selected")) {
        // Call the method to retrieve the discussion name/s
        Retrieve_Reply(Data);
        ToClient.println("Reply Name is Sent");
        break;
    } // if "TList_Selected"
    else {
        // Client request server to retrieve the discussion data
        if (Command.startsWith("RList_Selected")) {
            // Call the method to retrieve the discussion data
            Retrieve_ReplyData(Data);
            ToClient.println("Reply Data is Sent");
            break;
        } // if "RList_Selected"
        else {
            // Client request server to update the topic list in the NewsGrpIconVote frame.
            if (Command.startsWith("Update_TopicList")) {
                // Call the method to retrieve the topic names and send them to the server
                Retrieve_TopicListData();
            } // if "Update_TopicList"
        } // else
    } // else
} // else
break;
}

} // if "Update_TopicList"

else

// Basically, we want to update the
// discussion file (Reply?.txt) so that
// the newest result can be retrieved
// by the user the next time they double
// clicked the discussion list to
// retrieve the discussion data.

if (Command.startsWith("Update_OpinionPoll"))
{
    // Updates the discussion (Reply?.txt)
    // file so that the most updated data is
    // retrieved.

    Update_OpinionPoll(Data);

    break;
}

} // if "Update_OpinionPoll"

} // while

// Tell the client that the task is completed
ToClient.println("Bye");

// Close the PrintStream object
ToClient.close();

// Close the DataInputStream object
Client.close();

// Let the HoldData class knows that the socket has been
// terminated so that other methods get a chance to get
// hold of the HoldData object.

CHold.CloseSocket(SocketClosed);

// Sleep for a period of time so that other methods get
// a chance to use the socket.

sleep((int) (Math.random() * 3000));

System.out.println("Socket is closed. CONSUMER taking a
nap...");

} // try

catch (IOException e1)
{
    System.out.println("Communication Error between Client &
Server");
}

// catch

catch (InterruptedException e2)
{
    System.err.println(e2.toString());
}

} // run() method

275
APPENDIX C: SOURCE CODE [SERVER SIDE]

/*================================================================================================*/
/*
NEWSGROUP METHODS
/*
/* The code below contains all the methods that will be used by the server program to complete the
request from the client. Most of these methods are used to update the client data to the system
files for later retrieval.
/*
/*================================================================================================*/

/* Retrieve TOPICLIST data and its methods */
/* Retrieve TOPICLIST data and its methods */
public void Retrieve_TopicListData()
{
    // This method retrieves the topic names from the Topic_List.txt file and sends the data to the client to update the topic list.
    // Used as argument to Bubblesort() method so that it knows if it is sorting Topic OR Reply Names

    String File_Type = "";

    // Initialising the Vector to 3 and increment by 1 everytime the vector grows in size.
    // For sorting purposes

    Vector TNumberVector = new Vector(3,1);

    // Read the data from the Topic_List.txt file and updates the TopicTable property

    Update_TopicTable();

    // Creates a Topic Search Table that revert the TopicTable format from TNAME=TFNAME to TFNAME=TNAME format for easy searching.

    Create_Topic_SearchTable();

    // Get all the topic file names to be sorted from the TopicTable using the GetTopicFileName() method. We then pass the Vector to the GetTopicIntegerValue() method to extract all the integer values out for sorting purposes.

    TNumberVector = GetTopicIntegerValue(GetTopicFileName());

    // Now Sort those topic number using the simplest sorting algorithm BUBBLE sorting. If faster algorithm is needed, just insert the new algorithm here.

    File_Type = "Topic"; // Let Bubblesort method knows that it is sorting the topic file numbers.

    // Sort the topic file numbers

    BubbleSort(TNumberVector, File_Type);
}

public void Create_Topic_SearchTable()
{
    String TName = "", // Topic name
    TFileName = ""; // Topic file name

    Enumeration TopicNames;
    
    // First get all the topic names from the TopicTable. Using
    // the key() method will retrieve all the topic names since
    // it is the key.
    TopicNames = TopicTable.keys();
    
    // Now get the keys (Topic FILENAME [Topic?.txt]) for each
    // of the Topic names mentioned. Add the retrieved topic
    // file names to the vector
    while (TopicNames.hasMoreElements())
    {
        TName = (String) TopicNames.nextElement();
        // Check if the element is an empty string, if it is,
        // Do not add it to the TFileName VECTOR
        TFileName = new String
        (TopicTable.getProperty(TName));
        // Insert data into SearchTable in the following format
        // TFILENAME=TNAME for easy searching using the
        // BubbleSort() method
        TopicSearchTable.put(TFileName, TName);
    }
}

// Create_Topic_SearchTable method
public Vector GetTopicFileName()
{
    // Initialise the vector to 3 and increment by 1 everytime
    // the vector starts to grow.
    Vector TFName = new Vector(3,1);
    String TFileName = ""; // Contains the topic
    // file name
    Enumeration TopicNames;

    // First get all the topic file names from TopicSearchTable
    // properties
    TopicNames = TopicSearchTable.keys();
    while (TopicNames.hasMoreElements())
    {
        TFileName = TopicNames.nextElement().toString();
        if (TFileName.equals(""))
        {
            System.out.println("Null String will not be added to
            the Vector");
        } // if
        else
        {
            // Add the element to the TFName Vector
            TFName.addElement(TFileName);
        } // else
    } // while

    // Return a Vector object of the Topic file name
    System.out.println("TFNAME IS " + TFName);
    return TFName;
} // GetTopicFileName() method
public Vector GetTopicIntegerValue(Vector Topic_Names)
{
    // Initial size is 3, increment is 1 when the vector grows
    Vector TNumber = new Vector(1,1);
    Vector TName = new Vector(1,1);
    Enumeration TNameVector;

    String TopicName = "", // Contains the topic name
    SearchC = "c", // The character needed for
    SearchDot = ".", // The character needed for
    MyString = "; // The string that is
    // extracted

    int Start_Index = 0, // Index position of the
    // character 'c', so that
    // the program knows where
    // it is.
    End_INDEX = 0, // Index position of the
    // character ','
    Topic_Number = 0; // Initialise the topic
    // number when first started

    // Assign Local Vector variable to point to the Vector
    // argument
    TName = Topic_Names;

    System.out.println("Contents are = " + TName);

    // Use the element() method to return the Enumeration
    // interface for the Vector TNamexVector
    TNameVector = TName.elements();

    while (TNameVector.hasMoreElements())
    {
        // Get each of the elements, convert them to string
        // before assigning it to a string variable.
        TopicName = TNameVector.nextElement().toString();

        System.out.println("Currently processing = "
                        + TopicName);

        // We increment the Start_Index by one since we want
        // to start extracting the integer value after the value
Start_Index = TopicName.indexOf(SearchC) + 1;
End_Index = TopicName.indexOf(SearchDot);

We are now ready to extract the integer number from the topic file name for sorting purposes.

try {
    // Extract the character between the Start_Index and End_Index
    char[] Buf = new char[End_Index - Start_Index];
    // Now Get the Characters
    TopicName.getChars(Start_Index, End_Index, Buf, 0);
    // Now we have to convert the char in the buffer to a string object We do this using the code below
    MyString = new String(Buf);
    System.out.println("MyString value = " + MyString);
    // Convert the extracted string object to an integer value
    try {
        Topic_Number = Integer.parseInt(MyString);
        // Add the extracted number to the TNumber vector
        TNumber.addElement(new Integer(Topic_Number));
        System.out.println("TNumber contents = " + TNumber);
    } // try
    catch(NumberFormatException e) {
        System.err.println(e);
    } // catch(NumberFormatException)
    catch(ArrayIndexOutOfBoundsException e) {
        System.err.println(e);
    } // catch(ArrayIndexOutOfBoundsException)
} // try
catch(StringIndexOutOfBoundsException e) {
    System.err.println(e);
} // catch(StringIndexOutOfBoundsException)

} // while
return TNumber;
} // GetTopicIntegerValue(Vector) method
APPENDIX C : SOURCE CODE [SERVER SIDE]

/* Retrieve REPLY DATA & Its Method */
/*
Retrieve_ReplayData(String) Methods */
/* This method receives the topic name from the client and retrieve*/
/* the appropriate discussion names that match the topic name that*/
/* is sent by the Client. */
/*==================================================================*/

public void Retrieve_ReplayData(String Reply_Name) {
    // Read data from the Reply_List.txt file first
    // Open the Reply_List.txt file to update the ReplyTable
    // property
    Update_RListTable();
    // Check if the discussion file name exists, if it does.
    // send data to the client
    FindRFileName(Reply_Name);
} // Retrieve_ReplayData() method
public void Update_RListTable()
{
    FileInputStream RListInput;
    try
    {
        // Clear the Property table first
        RListTable.clear();

        // Opens the Reply_List.txt file and populate the
        // RListTable property table
        RListInput = new FileInputStream(RList_FName);
        RListTable.load(RListInput);
        RListInput.close();
    } // try
    catch(FileNotFoundException e1)
    {
        System.out.println("File NOT Found");
        System.exit(1);
    } // catch FileNotFound
    catch(IOException e2)
    {
        System.err.println(e2);
        System.exit(1);
    } // catch
} // Update_RListTable() method
public void FindRFilename(String PName) {
    String Reply_Filename = "",  // Contains the full
    FullPath = "";  // path to retrieve
    // the discussion file
    File RFilename;

    // Get the discussion file name that matches the discussion
    // name given by Client. if it doesn't, display an error
    // message.
    try {
        Reply_Filename = RListTable.getProperty
                        (PName,"REPLY NAMES DOES NOT EXIST");
        // Get the full path to the Reply_list.txt system file
        // so that we can make sure that the file has been
        // created.
        FullPath = FileLocation + Reply_Filename;
        System.out.println("Full path to Reply File = 
                           + FullPath);
        // Check if requested discussion filename exists
        RFilename = new File(FullPath);
        // If the file exists then send the data to the client,
        // otherwise display an error message
        if (RFilename.exists())
            { // Open the discussion file & sent data to the
              // client
                SentRData_ToClient(FullPath);
            }
        else
            {
                System.out.println("Requested Reply Filename
                                    NOT FOUND");
            }
    } // try

    catch(NullPointerException e)
    {
        System.out.println("Reply Name corrupted OR Reply FILE
                                    NOT FOUND \nSystem Message => " + e.toString());
    }
}
```java
public void SendRData_ToClient(String FileName)
{
    DataInputStream ReplyInput = null;
    String ReplyData = "";
    try
    {
        // Read discussion data from the discussion (Reply?.txt) file
        ReplyInput = new DataInputStream((new FileInputStream(FileName)));
        ReplyData = ReplyInput.readLine();
        System.out.println("ReplyData = " + ReplyData);
        ReplyInput.close();
        // Declare a PrintStream object and sends data to the client
        ToClient = new PrintStream(
            Client.getOutputStream());
        ToClient.println(ReplyData);
    }
    catch(FileNotFoundException e1)
    {
        System.out.println("File NOT Found");
        System.exit(1);
    } // catch FileNotFoundException
    catch(IOException e2)
    {
        System.err.println(e2);
    } // catch
    catch(NullPointerException e3)
    {
        System.out.println("Reply Name not found in 
            Reply_List.txt" + e3.toString());
    }
} // SendRData_ToClient(String) method
```
/**
 * RETRIEVE REPLY NAME & ITS METHOD
 */

/*@Retrieve_reply(String) Method*/
/*/ This method receives the topic name from the client and /*
/*/ retrieve the appropriate discussion names that matched the /*
/*/ topic name sent by the Client. */

public void Retrieve_reply(String Topic_Name) {
    // First Updates the TopicTable property table with data // retrieved from the Topic_List.txt system file
    Update_TopicTable();
    // Now search for the topic file name and send discussion // names to the client.
    FindRName(Topic_Name);
} // Retrieve_reply(String) Method

/*@FindRName(String) Method*/
/*/ This method search for the discussion names that matches the /*
/*/ topic name sent by the client. Once the discussion names are /*
/*/ retrieved, they are sent to the client to update the discussion List. /*
/*/ NOTE : Before sending the discussion names to the client, we /*
/*/ have sort it so that the lastest discussion names are /*
/*/ displayed first. To allow easy sorting, a discussion Name /*
/*/ SearchTable is created. */

public void FindRName(String Tname) {
    // Initialise vector to 3, increment by 1 when vector grows
    Vector RNumberVector = new Vector(3, 1);
    String Filetype = "-", // Used as argument to // BubbleSort() method so // that it knows if it is // sorting topic or // discussion names
    Topic_Filename = "",
    FullPath = ""; // Contains the full path to // the Topic?.txt file

    try
    {
        // Get topic file name matches the topic name given by // client
        Topic_Filename = TopicTable.getProperty
            (Tname, "TOPIC NAMES DOES NOT EXIST");
    }
} // FindRName(String) Method
Check if the requested topic file name exists. To check that the file exist, we have to include the FULL path so that the file can be found.

```java
FullPath = FileLocation + Topic_Filename;
System.out.println("FullPath = " +FullPath);
TFilename = new File(FullPath);
if (TFilename.exists())
{
    // Open the topic file and updates the TFileReplyTable property table so that the discussion names can be retrieved to the client.
    Update_TFileTable(FullPath);
    // This method is used for Bubble Sorting purposes
    // Creates a REPLY SearchTable that revert the TFileReplyTable format from RNAME=RFNAME to RFNAME=RNAME format for easy searching.
    Create_Reply_SearchTable();
    // Get all the discussion file names to be sorted from the TfileReplyTable table using the GetReplyFileName() method. We then pass the Vector to the GetReplyIntegerValue() method to extract all the integer values for sorting.
    RNumberVector = GetReplyIntegerValue(GetReplyFileName());
    // Now Sort those topic file number using the simplest sorting algorithm BUBBLE sorting. If a faster algorithm is needed, just insert the new algorithm here.
    BubbleSort(RNumberVector, "Reply");
}
else
{
    System.out.println("Requested Topic Filename NOT FOUND");
}
try
    catch(NullPointerException e)
    {
        System.out.println("Topic Name not found in Topic_List.txt. ERROR MESSAGE = " + e.toString());
    }
// FindRName(String) method
```

APPENDIX C : SOURCE CODE [SERVER SIDE]

II 1*=================================================================*1
I* Update_TFileTable (String) Method*
II 1*---------------------------------------------------------------*
II 2* This method open the topic file that contains the discussion *
II 3* names and update the TFileReplyTable property table with these *

287
public void Update_TFileTable(String TFName)
{
    FileInputStream TReplyInput;
    try
    {
        TReplyInput = new FileInputStream(TFName);
        // Clear the property table first
        TFileReplyTable.clear();
        // Now load it with new data from the Topic?.txt file
        TFileReplyTable.load(TReplyInput);
        // Close the file after reading from it.
        TReplyInput.close();
    } // try
    catch(FileNotFoundException e1)
    {
        System.out.println("File NOT Found");
        System.exit(1);
    } // catch FileNotFoundException
    catch(IOException e2)
    {
        System.err.println(e2.toString());
        System.exit(1);
    } // catch
    catch(NullPointerException e3)
    {
        System.out.println("Topic Filename might not be null value" + e3.toString());
        System.exit(1);
    } // try
}

/*--------------------------------------*/
/* Create Reply SearchTable() Method */
/*--------------------------------------*/
/* To sort the data, the BubbleSort() method needs to refer to the */
/* the RFNAME (E.g Reply8.txt) to get the integers values for */
/* sorting purposes. However, Data in TFileReplyTable is stored in*/
/* the RNAME=RFNAME format (E.g Good_Design=Reply8.txt). */
/* */
/* Since using properties method to get each of the RFNAME is */
/* cumbersome, instead let us create another TABLE that stores */
/* the results as RFNAME=RNAME for us to search the result. This */
/* way we can use the keys() method to retrieve all the RFNAME */
/* */
/* Thus this method will revert the TFileReplyTable format from */
/* RNAME=RFNAME to RFNAME=RNAME format for easy searching. */
/*--------------------------------------*/

288
public void Create_Reply_SearchTable()
{
    String RName = ""; // Discussion name
    String RFileName = ""; // Discussion file name

    Enumeration ReplyNames;

    // First get all the topic names from the TopicTable // property table
    ReplyNames = TFileReplyTable.keys();

    // Now get the keys (discussion file names - [Reply?.txt]) // for each of the discussion names mentioned. Add the // retrieved discussion file names to the Vector
    while (ReplyNames.hasMoreElements())
    {
        RName = (String) ReplyNames.nextElement();

        // Check if the element is an empty string, if it is, // Do not add it to the RFName vector
        if (RName.equals(""))
        {
            RFileName = new String
                (TFileReplyTable.getProperty(RName));

            // First clear the previous contents in // ReplySearchTable. Insert data into SearchTable in the // following format TFILENAME-TNAME for easy searching by // BubbleSort() method
            ReplySearchTable.put(RFileName, RName);
        }
    }
}

// Create_Reply_SearchTable method
// "===================================================================
// * GetReplyFileName(): Method
// "===================================================================
// This method is used to retrieve the discussion names in the // TFileReplyTable property table. However, the existing format in // TFileReplyTable is in the [ReplyFilename-ReplyFileName] format. Thus // we have to retrieve all the ReplyFilename i.e Reply?.txt for // sorting purposes.
// "===================================================================

public Vector GetReplyFileName()
{
    // Initialise to 3, increment by 1 when vector grows
    Vector RFName = new Vector(3, 1);
    String RFileName = "";

    Enumeration ReplyNames;

    // First get all the discussion file name from // ReplySearchTable
    ReplyNames = ReplySearchTable.keys();

    while (ReplyNames.hasMoreElements()){
        RFileName = (ReplyNames.nextElement()).toString();

        if (RFileName.equals(""))
        {
            // Add the discussion file names to the RFName vector
            RFName.addElement(RFileName);
        }
    }
}

// GetReplyFileName method
// "===================================================================
// * GetReplyFileName(): Method
// "===================================================================
// This method is used to retrieve the discussion names in the // TFileReplyTable property table. However, the existing format in // TFileReplyTable is in the [ReplyFilename-ReplyFileName] format. Thus // we have to retrieve all the ReplyFilename i.e Reply?.txt for // sorting purposes.
// "===================================================================
System.out.println("Null String will not be added to the Vector");

// if
else
{
    // Add the element to the TPName Vector
    TPname.addElement(FileName);
} // else

} // while

// Return a Vector object of the discussion file names
return TPName;

} // GetTopicFileName() Method
public Vector GetReplyIntegerValue(Vector Reply_Names)
{
    // Initial size is 3, increment to 1 when the vector grows
    // Use to hold the extracted integer values from Reply?.txt string
    Vector RNumber = new Vector(3,1);
    Vector RName = new Vector(3,1);
    Enumeration RNameVector;
    String ReplyName = "", // Contains the discussion name
    SearchY = "y", // The character needed for sorting purposes
    SearchDot = ".", // The character needed for sorting purposes
    MyString = ""; // The string that is extracted
    int Start_Index = 0, // Index position of the character 'c', so that the program knows where it is.
    End_Index = 0, // Index position of the character '.
    Reply_Number = 0; // Initialise the topic number when first started

    // Assign Local vector variable to point to the Vector argument
    RName = Reply_Names;

    System.out.println("Contents are = " + RName);
APPENDIX C: SOURCE CODE [SERVER SIDE]

// Use the element() method to return the Enumeration // interface for the Vector RNameVector
RNameVector = RName.elements();

while (RNameVector.hasMoreElements())
{
    // Get each of the Elements
    ReplyName = RNameVector.nextElement().toString();
    System.out.println("Currently processing = " + ReplyName);

    // We increment the Start_Index by one since we want to // start extracting the integer value after the value // "c", otherwise the result will include the 'c' char
    Start_Index = ReplyName.indexOf(SearchY) + 1;
    End_Index = ReplyName.indexOf(SearchDot);

    try
    {
        // Extract the character between the Start_Index and // End_Index
        char Buf[] = new char[End_Index - Start_Index];

        // Now Get the Characters
        ReplyName.getChars(Start_Index, End_Index, Buf, 0);

        // Now we have to convert the char in the buffer // (Buf) to a string object. We do this using the // code below
        MyString = new String(Buf);
        System.out.println("MyString Value = " + MyString);

        // We then assign the extracted string object to an // integer value
        try
        {
            Reply_Number = Integer.parseInt(MyString);
            RNumber.addElement(new Integer(Reply_Number));
            System.out.println("RNumber contents = " + RNumber);
        }
    }
    catch(NumberFormatException e)
    {
        System.err.println(e);
    } // catch(NumberFormatException)

    catch(ArrayIndexOutOfBoundsException e)
    {
        System.err.println(e);
    } // catch(ArrayIndexOutOfBoundsException)
public void Bubblesort(Vector Number_To_Sort, String Type_Of_File)
{
    Vector RName_Number = new Vector(3,1);
    // Assign Vector argument to Local variable.
    RName_Number = Number_To_Sort;
    Object ObjectArray[] = new Object[RName_Number.size()];
    int IntArray[] = new int[RName_Number.size()];
    int ArraySize = 0,
    First_Number = 0, // Contains the first i value to check
    Second_Number = 0, // Contains the next i value for checking
    Hold = 0; // Variable to temporary hold values for swapping
    String File_Type = "," // The file name TAG
    Extention = ".txt", // The extention for all files
    NameToClient = "," // Topic or discussion name to be sent to
    // the client for display.
    Final_FNames = ";" // Contains the concatenated string
    // which include the Strings "File_Type" +

    // This method can be used to sort both the topic and discussion names. One problem with using the Vector class is that the items are stored as object thus the Object "8" cannot be compared with Object "5" for instance to see if one is lesser than the other.
    // This method will retrieved the objects in the Vector, convert it to a string object then to an integer value for comparison. The result is then bubble sorted and the string object will be concatenated as "Topic?.txt" or "Reply3.txt" for example. Note that the result will be sent as a single string separted by the '#' character.

    try
    {
        catch(StringIndexOutOfBoundsException e)
        {
            System.err.println(e);
        }
        // catch(StringIndexOutOfBoundsException)
    }
    // while
    return RNumber;
} // GetTopicIntegerValue(Vector) method
To allow this method to be able to sort both topic & discussion, this method need to know what file type it is sorting so that the correct name can be sent to the client.

```java
File_Type = Type_Of_File;
```

Since items in a Vector are objects, they cannot be compared if one is lesser than another, thus we copy all the objects in RName_Number Vector into the ObjectArray which holds objects in array then convert all the objects into an integer value for sorting.

```java
try {
    RName_Number.copyInto(ObjectArray);
} // try
```

```java
catch(ArrayIndexOutOfBoundsException e) {
    System.out.println(e.toString());
} // catch
```

Determine the size of the Array

```java
ArraySize = RName_Number.size();
```

Convert all objects into string then to their integer values and copy that to the IntArray

```java
for (int i=0; i < ArraySize; i++) {
    System.out.println("Object Array = " + ObjectArray[i]);
    for (int j=0; j < IntArray.length; j++) {
        // Object item in Vector is converted to a string
        // object first then to an integer value for sorting
        // purposes
        IntArray[i] = Integer.parseInt(ObjectArray[i].toString());
    } // for
} // for
```

Having all the integer at hand, we can now sort the array from the largest number to the smallest.

```java
for(int pass = 1; pass < ArraySize; pass++) {
    for(int i = 0; i < IntArray.length - 1; i++)
        if(IntArray[i] < IntArray[i + 1]) {
            // Swap the values
            Hold = IntArray[i];
            // Assign current value to the value of next value
```

294
IntArray[i] = IntArray[i + 1];
System.out.println("Swap " + Hold + "with"
 + IntArray[i]);

// Assign value of next value to current value;
IntArray[i + 1] = Hold;
} // if
} // for

// Once all the data has been sorted, concatenate the sorted
// integer values with the string "File_Type" to form the
// final file name (E.g "Reply" + 8 = "Reply8.txt") Then
// send all the file names to the client applet.
for(int k=0; k<IntArray.length; k++)
{
    Final_FNames = File_Type + IntArray[k] + Extention;

    System.out.println("File names to search = 
    + Final_FNames);

    // Search the discussion name that matches this file
    // name. Get all the result and sends everything to the
    // client as a single string seperated by '!' char.
    if (File_Type.equals("Topic"))
    {
        NameToClient += TopicSearchTable.getProperty(Final_FNames) + "#";
    } // if
    else
    {
        NameToClient += ReplySearchTable.getProperty(Final_FNames) + "#";
    } // else
} // for
try
{
    // Sent the SORTED discussion name to the client
    ToClient = new PrintStream
    (Client.getOutputStream());
    ToClient.println(NameToClient);
    System.out.println("SENDING TOPIC/REPLY NAMES TO CLIENT
    = " + NameToClient);
}

try
catch(IOException e1)
{
    System.err.println(e1);
} // catch

catch(NullPointerException e2)
{
    System.out.println("Reply Name not found in
    Reply_List.txt" + e2.toString());
}

} // BubbleSort(Vector,String) method

/*=================================================================**/
/* Create Topic(String) Methods */
/*=================================================================**/

public void Create_Topic(String Topic_Name)
{
    String TFile = ","; // Topic file name to create
    TName = ","; // Topic name send by the client
    FullPath = ","; // Contains the full path to the file
    FileTag = "Topic", // Topic file name
    ExtentionTag = ".txt"; // Topic file extention

    int TName_Counter = 0, // Topic counter value
    Updated_Counter = 0; // Contains updated counter

    Topic_Count = new File(TCount_FName);
    TName = Topic_Name; // Topic name sent by the client
    // Check that the file exist, if it does then we can create
    // a topic file name for storing discussion information.
    if (Topic_Count.exists() == true)
    {
TName_Counter = read_TCounter(); // Get the counter
    // value from file.

System.out.println("Initial Counter value = "
        + TName_Counter);

    // Check that the next topic file number is not more
    // than our maximum level set for the program. If
    // everything is fine, create the topic file.
if (TName_Counter != MaxFiles)
{
    TFile = FileTag + TName_Counter + ExtentionTag;
        // Increment counter to produce next file name
        TName_Counter++;
} // if

    // Get counter to create NEW file name
Updated_Counter = TName_Counter;

    // Write Counter value to the Topic_Count.txt file. This
    // way we know what the next topic name will be.
    // Write updated counter value to the Topic_Count.txt
    // file
write_TCountFile(Updated_Counter);

    // Create NEW Topic file.
    // To do so, we include the FULL path so that the file
    // can create/save to the same location
FullPath = FileLocation + TFile;
create_TopicFile(FullPath);

    // Use to update the contents in the Topic List.txt
    // When updating Topic_List.txt file, we do not need
    // to include the full path since we have done so for
    // the the Topic_List.txt file
update_TopicFile(TFile, TName);
}
else
{
    System.out.println("Topic_Count.txt File does not
        exist");
} // else
} // // Create_Topic() method

/************************************************************
 */
/* read_TCounter() Method */
/************************************************************
/* This method is used to READ the counter value from the */
/* "Topic_Count.txt" file */
/************************************************************

public int read_TCounter()
{
    int Int_Value = 0; // Initialise variable
FileInputStream TCountInput;

// Open the file Topic_Count.txt file and READ the contents
// from it.
try{
    TCountInput = new FileInputStream(TCount_PName);
    // Read counter value from file
    Int_Value = TCountInput.read();
    // Close the Topic_Count.txt file
    TCountInput.close();
} // try

catch(FileNotFoundException e1)
{    System.out.println("File NOT Found");
    System.exit(1);
} // catch FileNotFoundException

catch(IOException  e2)
{    System.err.println("File not opened properly\n" + e2.toString());
    System.exit(1);
} // catch

return Int_Value;

} // read_TCounter method
import \* as fs from 'fs';

// This method will WRITE the updated counter value to the Topic_Count.txt file for new file names to be created.

public void write_TCountFile(int TCounter) {
    FileOutputStream TCountOutput;
    // Opens the file Topic_Count.txt file and WRITE the contents to the file.
    try {
        TCountOutput = new FileOutputStream(TCount_FName);
        // Writing updated counter value to the Topic_Count.txt file
        TCountOutput.write(TCounter);
        System.out.println(TCounter + " value has been written to Topic_Count.txt");
    }
    catch(FileNotFoundException e1) {
        System.out.println(TCount_FName + " File NOT Found" + e1.toString());
        System.exit(1);
    } // catch FileNotFoundException
    catch(IOException e2) {
        System.out.println("write_TCountFile() Method");
        System.err.println(e2);
    } // catch
} // write_TCountFile() method
public void create_TopicFile(String TFName)
{
    DataOutputStream TFileOutput;
    String Content = "";
    try
    {
        TFileOutput = new DataOutputStream(new FileOutputStream(TFName));
        System.out.println(TFName + " is created !!!");
        TFileOutput.writeUTF(Content);
        TFileOutput.close();
    } // try
    catch(IOException e)
    {
        System.out.println("Error writing to file !! "
                + toString());
    } // catch
} // create_TopicFile() method

public void update_TopicFile(String TFile, String TName)
{
    FileInputStream TListInput;
    FileOutputStream TListOutput;
    try
    {
        // FIRST READ the topic names from the Topic_List.txt
        // Load them to the TopicTable Property Table
        TListInput = new FileInputStream(TList_FName);
        TopicTable.load(TListInput); // Updates TopicTable
        TListInput.close(); // Close Topic_List.txt file
        // PUT new topic names to the TopicTable Property Table
        // NOW WRITE topic names to the Topic_List.txt
        TListOutput = new FileOutputStream(TList_FName);
    } // try
} // update_TopicFile(String, String)
// Put data to TopicTable property
TopicTable.put(TName, TFile);
TopicTable.save(TListOutput, "Topic FileNames & its Contents");

// Close Topic_List.txt file
TListOutput.close();
}  // try

catch (FileNotFoundException e1)
{
    System.out.println("File NOT Found");
    System.exit(1);
}  // catch FileNotFoundException

catch (IOException e2)
{
    System.err.println(e2);
}  // catch

}  // update_TopicFile(String, String) Method

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%**/
/* MANAGE REPLY DATA & ITS METHOD */
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%**/

/*============================================================================*/
/* MANAGE REPLY DATA(String) methods */
/*============================================================================*/

/* This method receives the discussion name from the client and */
/* updates the Topic_List.txt. It creates the discussion file and */
/* writes the discussion data to newly created discussion file. */
/*============================================================================*/

public void ManageReply_Data(String Reply_Info)
{
    int MaxNumbers = 6;
    String TName = "", // Topic name
    UName = "", // User/ Author's name
    RName = "", // Discussion name
    DateData = "", // Discussion date
    Message = "", // Discussion contents
    ImageName = "", // Image Name selected by user
        // as their response
    RContent = "", // FINAL discussion data string
    // to be kept in the Reply.txt
    NewRFName = "", // Newly Created discussion file
        // name

    String ReplyData[] = new String[MaxNumbers];

    try
    {
        StringTokenizer Tokens = new StringTokenizer(Reply_Info, ",");

        for (int i=0; i<MaxNumbers; i++)
        {
            if (Tokens.hasMoreTokens())
            {
                StringTokenizer SubTokens = new StringTokenizer(Tokens.nextToken(), "#");
                for (int j=0; j<SubTokens.countTokens(); j++)
                {
                    ReplyData[i] += SubTokens.nextToken();
                }
            }
        }
    }
    catch (Exception e)
    {
        System.out.println("Error: "+e);
    }

    // Save Reply Data to Reply.txt file
    ListOutput = new ListOutput();
    ListOutput.put(TName, new String[MaxNumbers]);
    ListOutput.save(TListOutput, "Reply FileNames & its Contents");

    // Close Reply_List.txt file
    TListOutput.close();
}

}  // ManageReply_Data(String)
APPENDIX C : SOURCE CODE [SERVER SIDE]

```java
{
    ReplyData[i] = Tokens.nextToken();
    System.out.println("ReplyData = "
        + ReplyData[i]);
} // if
} // for

// Start reading from array[1] because array[0]
// contains the command string.
try {
    // Contains the TOPIC name
    TName = ReplyData[0];
    System.out.println("TName = " + TName);

    // Contains the DISCUSSION Name
    RName = ReplyData[1];
    System.out.println("RName = " + RName);

    // Contains the discussion date
    DateData = ReplyData[2];
    // Contains the User name
    UNarne = ReplyData[3];
    // Contains the discussion message
    Message = ReplyData[4];

    // Contains the Image name to be display on the
    // ReplyList
    ImageName = ReplyData[5];
    RContent = UNarne + "]" + DateData + "]" +
        Message;
    System.out.println("RContent = " + RContent);
} // try

catch (ArrayIndexOutOfBoundsException e) {
    System.err.println("Array out of Bounds = " + e);
} // catch

catch (NullPointerException e) {
    System.out.println("NullException Caught" + e.toString());
} // catch (NullPointerException)

catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Array out of Bounds = " + e);
} // catch (ArrayIndexOutOfBoundsException)

// Create A NEW discussion file name
// (Only the Filename NOT the FILE itself)
```

302
APPENDIX C: SOURCE CODE [SERVER SIDE]

NewRFName = Create_Replay();
System.out.println("Created Following File = " + NewRFName);

// Create NEW discussion file and write discussion data to
// it. Also updates the discussion (Reply_List.txt) file
Update_ReplyFile(NewRFName, RName, RContent, ImageName);

// Update the Topic?.txt file to include the discussion name
// and the Image name as well
Update_TopicFile(TName, NewRFName, RName, ImageName);
}

} // ManageReply_Data(String) method

/*-----------------------------------------------*/
/* Update_ReplyFile(String,String,String,String) Method */
/*-----------------------------------------------*/
/* This method is used to create a discussion file and writes */
/* data to the file. The key to using only one file to update is */
/* the use of TopicTable property. It also updates the */
/* Reply List.txt file. */
/*-----------------------------------------------*/

public void Update_ReplyFile(String RFName,
String RName,
String RData,
String Image_Name)
{
    // Create Reply?.txt with contents
    Create_RFile(RFName, RData);

    // Update the Reply_List.txt system file
    Update_RListFile(RFName, RName, Image_Name);
}

} // Update_ReplyFile(String,String,String,String) method
/** Create Reply & Its Method */

public String Create_REPLY()
{
    int RNameCounter = 0;
    String RFile = "", FileTag = "Reply", ExtentionTag = ".txt";

    // Read counter and create discussion file
    // Get Counter Value from the Reply_Count.txt file
    RNameCounter = read_RCounter();

    System.out.println("Counter Retrieved in Client = " + RNameCounter);

    // Creates the discussion (Reply?.txt) file
    if (RNameCounter != MaxFiles)
    {
        RFile = FileTag + RNameCounter + ExtentionTag;
        System.out.println("Reply File Created = " + RFile);
        // Increment counter to produce next file name
        RNameCounter++;
    } 

    // Write the updated counter value to the file
    System.out.println("Updated counter = " + RName_Counter);

    // Writing updated counter value to the server
    write_RCountFile(RName_Counter);
    return RFile;
} // Create_REPLY() method
public void Create_RFile(String RFName, 
                         String Content)
{
    // For creating discussion File
    FileOutputStream RFileOutput;
    // For printing data as a string to the Reply?.txt file
    PrintStream PrintToFile;
    // When creating a new discussion file, we have to give it
    // the full path
    String FullPath = "";
    int Voted = 0, // Number of people who has
                // submitted their opinion
    Not_Voted = 0, // Number of people who has not
                   // submitted their opinion
    InFavour = 0, // Number of people who are
                  // in favour of the discussion
    Not_InFavour = 0; // Number of people who are
                        // against the discussion
    // Before we create the discussion file, add the following
    // values to the discussion file so that it can be used by
    // the "Opinion Poll" system to calculate the necessary
    // values.
    // [1] Reply Filename (Reply?.txt)
    // [2] Total no.of People Participating
    // [3] Number of people GIVEN OPINION
    // [4] Number of people NOT GIVEN OPINION
    // [5] Number of people IN FAVOUR of discussion
    // [6] Number of people AGAINST the discussion
    // NOTE : The number of people who has NOT voted is equal
            to the TOTAL number of people initially NOT zero
    Not_Voted = TotalNumber;
    Content += "" + RFName + "" + TotalNumber + "" +
                Voted + "" + Not_Voted + "" + InFavour +
                "" + Not_InFavour;
    // When we are creating a new discussion file, we have to
    // supply it the full name. BUT if we just want to retrieve
    // the discussion file names to a Property table, just the
    // discussion file name will do.
    FullPath = FileLocation + RFName;
    try
    {
        RFileOutput = new FileOutputStream(FullPath);
        // Print data to the file specified
PrintToFile = new PrintStream(RFileOutput);
PrintToFile.println(Content); // Print data to file
RFileOutput.close(); // Close the file
}
}

try
// Print data to file
II
Close the file

catch(FileNotFoundException e1)
{
    System.out.println(RName + " File NOT Found" + e1.toString());
}
// catch FileNotFoundException

catch(IOException e2)
{
    System.err.println(e2.toString());
    System.exit(1);
}
// catch
}

II
try
II
// Create_RFile(String, String) method

/*================================================================*************/
/*================================================================*************/

public void Update_RListFile(String RFName, String RName, String Image_Name)
{
    FileInputStream RListInput;
    FileOutputStream RListOutput;
    // First Load ReplyListTable with previous discussion names
    // and values
    try
    {
        // Clear the ReplyListTable first
        ReplyListTable.clear();
        RListInput = new FileInputStream(RList_FName);
        // Load the ReplyListTable property
        ReplyListTable.load(RListInput);
        // Close the file
        RListInput.close();
    }
    // try
catch(FileNotFoundException e1)
{
    System.out.println("File NOT Found");
    System.exit(1);
} // catch FileNotFoundException

catch(IOException e2)
{
    System.err.println(e2);
} // catch

// Update the ReplyListTable with new discussion names and value

try
{
    RListOutput = new FileOutputStream(RList_FName);
    // Combine discussion name + Image File b4 we put the key into the ReplyListTable property table.
    String ModifiedKey = FName + "-" + Image_Name;
    // Use discussion names as key and discussion file name as value
    ReplyListTable.put(ModifiedKey,RFName);
    // Saves the newly updated value
    ReplyListTable.save(RListOutput,"Reply FileNames & its Contents");
    // Close the file
    RListOutput.close();
} // try

catch(FileNotFoundException e3)
{
    System.out.println("File NOT Found");
    System.exit(1);
} // catch

catch(IOException e4)
{
    System.err.println(e4);
} // catch

} // Update_RListFile(String, String, String) method

public void Update_TopicFile(String TName,
                            String Reply_FName,
                            String FName,
                            String Image_Name)
{
    // First Update the TopicTable property from data
APPENDIX C : SOURCE CODE [SERVER SIDE]

```java
// retrieved from Topic_List.txt
Update_TopicTable();

// Now find the Topic File name & updates the Topic File.txt
// file with the discussion name and image name as well
// Search for Topic FILE name
FindTFileName(TName,Reply_iName,FName,Image_Name);
}

} // Update_TopicFile(String, String, String, String, String) method

/**
  * Update_TopicTable() Method
  *---------------------------------------------------
  * This method opens the Topic_List.txt and updates the TopicTable property table. The TopicTable property table is used to retrieve the Topic File Name that is associated with the Topic name in Topic_List.txt file.
  *---------------------------------------------------
  */

public void Update_TopicTable()
{
    FileInputStream TListInput;
    try
    {
        // First Clear the TopicTable property then load the
        // Topic_List.txt file to the TopicTable property list.
        TopicTable.clear();
        TListInput = new FileInputStream(TList_FName);
        TopicTable.load(TListInput);
        TListInput.close();
    } // try
    catch(FileNotFoundException e1)
    {
        System.out.println("File NOT Found");
        System.exit(1);
    } // catch FileNotFoundException
    catch(IOException e2)
    {
        System.err.println(e2.toString());
        System.exit(1);
    } // catch
}

} // Update_TopicTable() method

/**
  * FindTFileName(String,String,String,String) Method
  *--------------------------------------------------
  * This method is used to retrieve discussion name in TFileTable property using the topic name sent by the client. The retrieved discussion names is then output to the discussion file.
  *--------------------------------------------------
  */

public void FindTFileName(String TopicName, String RFName, String ReplyName, String Image_Name,
{
```
String Image_Name;
{
    // Contain the FULL PATH to the Topic?.txt file
    String FullPath = "";
    File TFilename;
    String Topic_Filename = "";
    try {
        // Get topic file name associated with topic name given
        // by client
        Topic_Filename = TopicTable
            .getProperty(TopicName,"TOPIC NAMES DOES NOT EXIST");
        System.out.println("Topic Filename = " + Topic_Filename);

        // Check if the requested topic file name exists
        // Before checking, give it the full path to the
        // file. To change any path for new users, just change
        // the variable "FileLocation" at top of program
        FullPath = FileLocation + Topic_Filename;
        System.out.println("Topic file FULL PATH = " + FullPath);
        TFilename = new File(FullPath);
    }
}
// Check if the file exists.
if (TFilename.exists())
{
    Update_ReplyTable(FullPath);
    // Update Topic?.txt with discussion name and image name
    open_TFile_ToWrite(FullPath, RFilename, ReplyName, Image_Name);
}
else
{
    System.out.println("Requested Topic Filename NOT FOUND");
}
} // try

catch(NullPointerException e)
{
    System.out.println("ERROR LOCATING TOPIC NAME IN Topic_List.txt ! " + "System Err Msg => " + e.toString());
}
} // catch

} // FindTFileName(String, String, String, String) method

public void Update_ReplyTable(String Topic_FName)
{
    FileInputStream TFileInput;
    try
    {
        // First Clear the ReplyTable property, then load the Topic file name to the ReplyTable property table.
        ReplyTable.clear();
        TFileInput = new FileInputStream(Topic_FName);
        ReplyTable.load(TFileInput);
        TFileInput.close();
    } // try

    catch(FileNotFoundException e1)
    {
        System.out.println("File NOT Found");
        System.exit(1);
    } // catch FileNotFoundException

    catch(IOException e2)
    {
        System.err.println(e2.toString());
        System.exit(1);
    } // catch

} // Update_ReplyTable method
APPENDIX C: SOURCE CODE [SERVER SIDE]

} // Update_RetryTable() method

/*==============================================*/
/* open_TFile_Write(String,String,String) Method */
/*==============================================*/
/* Used to open the Topic?.txt file. By opening it, it allows the */
/* Topic file name to be written sequentially to the */
/* Topic_list.txt file. If the file is open % close before the */
/* new data can be written to it, data will be overwritten. */
/*==============================================*/

public void open_TFile_Write(String TFile,
        String Reply_FName,
        String FName,
        String Image_Name)
{
    FileOutputStream TFileOutput;
    // Open the topic file and writes the discussion names to it
    String Topic_FName = TFile;
    try
    {
        System.out.println("Opening " + "Topic_FName");
        TFileOutput = new FileOutputStream(Topic_FName);
        System.out.println("Writing Reply data to file ... ");
        // Combine discussion name and Image name b4 inserting
        // into the ReplyTable property table
        String ModifiedKey = FName + "-" + Image_Name;
        // Place new data to ReplyTable
        ReplyTable.put(ModifiedKey,Reply_FName);
        // Save the new data to ReplyTable
        ReplyTable.save(TFileOutput,"Reply Names Related to
This Topic");
        TFileOutput.close();
        System.out.println("Closing " + "Topic_FName");
    } // try

    catch(IOException e)
    {
        System.err.println(e.toString());
        System.exit(1);
    } // catch

} // open_TListFile_Write(String,String,String) method

/*==============================================*/
public int read_RCounter() {
    int Int_Value = 0; // Initialise variable
    FileInputStream RCountInput;
    try {
        RCountInput = new FileInputStream(RCount_FName);
        // Read counter value from file
        Int_Value = RCountInput.read();
        // Close Topic_Count.txt file
        RCountInput.close();
    } // try
    catch (FileNotFoundException e1) {
        System.out.println(RCount_FName + " File NOT Found");
        System.exit(1);
    } // catch FileNotFoundException
    catch (IOException e2) {
        System.err.println(e2.toString());
        System.exit(1);
    } // catch
    return Int_Value;
} // read_RCounter method
public void write_RCountFile(int RCounter)
{
    FileOutputStream RCountOutput;
    try
    {
        RCountOutput = new FileOutputStream(RCount_FName);
        // Writing updated counter value to Topic.Count.txt
        RCountOutput.write(RCounter);
        System.out.println(RCounter + " value has been written to Topic.Count.txt");
        // Close the Topic_Count.txt file
        RCountOutput.close();
    } // try
    catch(FileNotFoundException e1)
    {
        System.out.println(RCount_FName + " File NOT Found" + e1.toString());
        System.exit(1); // catch FileNotFoundException
    }
    catch(IOException e2)
    {
        System.out.println("write_RCountFile() Method");
        System.err.println(e2);
    } // catch
} // write_RCountFile() method
public void Update_OpinionPoll(String Reply_Info) {
    int MaxNumbers = 9;
    String RFileName = "", // Discussion file name to
    // update the data
    UName = "", // User/ Author's name
    DateData = "", // Discussion date
    Message = "", // Discussion contents
    // Opinion Poll results
    Total = "", // Total number of voters voting
    Voted = "", // Number of people who have
    // voted
    NotVoted = "", // Number of people who have NOT
    // voted
    InFavour = "", // Number of people who are in
    // favoured of the discussion
    Against = "", // Number of people who are NOT
    // in favour of the discussion
    RContent = ""; // FINAL Reply Data string to be
    // kept in the Reply?.txt

    String ReplyData[] = new String[MaxNumbers];
    try {
        StringTokenizer Tokens = new StringTokenizer(Reply_Info, "#");
        for (int i=0; i<MaxNumbers; i++)
            if (Tokens.hasMoreTokens())
                ReplyData[i] = Tokens.nextToken();
        System.out.println("ReplyData = "+ReplyData[i]);
    } // if
    // Start reading from array[1] because array[0]
    // contains the command String.
    try {
        }
UName = ReplyData[0]; // The User name
DateData = ReplyData[1]; // Discussion date
Message = ReplyData[2]; // Discussion contents

// Contains the discussion file name to update the info
RFileName = ReplyData[3];
System.out.println("Reply File Name = " + RFileName);

// Total number of people participating in the opinion poll
Total = ReplyData[4];

// Number of people who has submitted their opinion
Voted = ReplyData[5];

// Number of people who has not submitted their opinion
NotVoted = ReplyData[6];

// Number of people who are in favour of the discussion
InFavour = ReplyData[7];

// Number of people who are against the discussion
Against = ReplyData[8];

RContent = UName + "#" + DateData + "#" + Message + "#" + RFileName + "#" + Total + "#" + Voted + "#" + NotVoted + "#" + InFavour + "#" + Against;

System.out.println("Data to update " + RFileName + " = " + RContent);
}

} // try

catch(ArrayIndexOutOfBoundsException e)
{
    System.err.println("Array out of Bounds = " + e);
} // catch

} // try

catch(NullPointerException e)
{
    System.out.println("NullPointerException Caught" + e.toString());
} // catch (NullPointerException)

catch(ArrayIndexOutOfBoundsException e)
{
    System.out.println("Array out of Bounds = " + e);
} // catch (ArrayIndexOutOfBoundsException)

// Opens the discussion (Reply?.txt) file and write the NEW discussion data to the file.
Update_ReplyFile(RFileName, RContent);
public void Update_ReplyFile(String Reply_Filename, 
String Reply_Content) 
{ 
File Reply_File; // The discussion file name object 
String RFileName = "", // The discussion file name to update 
RContent = "", // The contents to be updated to discussion file 
Fullpath = ""; // Contains the absolute path to locate the file 

// Assign global variable to local variables 
RFileName = Reply_Filename; 
RContent = Reply_Content; 

// Before we write to the file, we have to supply the FULL path name so that the file can be found. 

FullPath = FileLocation + RFileName; 
Reply_File = new File(FullPath); 

// Check that the file exist, if it does, update data to it. 
if (Reply_File.exists() == true) 
{ 
// Writes the NEW discussion data with the updated Opinion poll results to the discussion File 
// (Reply?.txt) 
WriteTo_ReplyFile(FullPath, RContent); 
} 
else 
{ 
System.out.println(RFileName + " File DOES NOT EXIST for updating!!!"); 
} 
} // else 

/** Update_ReplyFile(String, String) method 
 *************************************************************/ 
/** WriteTo_ReplyFile(String, String) Methods */ 
/** This method writes the NEW Reply data and the updated opinion poll results to the existing discussion file. */ 
passport void WriteTo_ReplyFile(String Reply_Filename, 
String Reply_Content) 
{ 
FileOutputStream RFileOutput; // For creating discussion File 
PrintStream PrintToFile; // For printing data as string to the Reply?.txt
try
{
    RFileOutput = new FileOutputStream(Reply_Filename);
    // Instantiate the PrintStream Object
    PrintToFile = new PrintStream(RFileOutput);
    // Print data to the file
    PrintToFile.println(Reply_Content);
    // Close the file
    RFileOutput.close();
} // try

catch(FileNotFoundException e1)
{  
    System.out.println(Reply_Filename + " File NOT Found");
    System.exit(1);
} // catch FileNotFoundException

catch(Exception e2)
{
    System.err.println(e2.toString());
    System.exit(1);
} // catch

} // WriteTo_ReplyFile(String, String) method

} // ExecuteIconVote Class
REFERENCES


REFERENCES


REFERENCES


  USA: Sams.net Publishing.


• Lotus Corp. (1995a). *Collaboration* [on-line].

• Lotus Corp. (1996b) *The Message Store.* [on-line]
  Available WWW: http://www.lotus.com/bible/21EA.htm

• Lotus Corp. (1996c) *Coordination.* [on-line]

• Lotus Corp. (1996d) *Defining Groupware.* [on-line]


  UK: Biddles, Inc


  USA: McGraw-Hill, Inc.


References

USA: Morgan Kaufmann Publishers


REFERENCES


