



Report on Industrial Attachment with Hewlett Packard Labs, Bristol, England

Ivan Low Swee Tieng
Publishing Systems Solutions Laboratory
HP laboratories Bristol
HPL-2000-68
7th June, 2000*

Baskerville
project,
E-services,
E-speak,
XML user
interface

The Hewlett-Packard Lab's goal of engaging in world-class research in the interrelated technologies has resulted in a need to invent new solutions to address the challenges of a web economy - from secure e-commerce, high availability systems to electronic content publishing. The Baskerville Project, a commercial Printing eServices project done by the Publishing Systems and Solutions Unit in Hewlett-Packard Laboratory, Bristol, sets to create and develop an infrastructure to enable desktop access to commercial printing capabilities.

The students' project, which contributed to part of the Baskerville Project, requires the students to produce components of a workflow concerning outsourced printing. The two main components will involve web front ends to a prototype printing service under development at HP labs. The technical competencies that will be learned by the students include Object Oriented and Java skills, and Internet and e-service technologies.

* Internal Accession Date Only

© Copyright Hewlett-Packard Company 2000

NANYANG TECHNOLOGICAL UNIVERSITY



Report on Industrial Attachment

with

**Hewlett-Packard Laboratories
Bristol, England**

**Prepared By:
Low Swee Tieng, Ivan
983330G06
17 May 2000**

**SCHOOL OF APPLIED
COMPUTER ENGINEERING**

Overview

This report is organized in the following order:

Chapter One: Introduction

This chapter explains the Purpose, Scope and Limitation. It also gives an introduction to the Company, Hewlett-Packard Lab, including its background, research goals and university partnerships.

Chapter Two: Project Background

This chapter summarizes the background information of the ‘Baskerville Project’ as well as the students’ responsibilities.

Chapter Three: Training, Research and Implementation

This chapter discusses the training and research done by the students, as well as the results of the implementation.

Chapter Four: The Attachment Conclusion

This chapter gives the overall conclusion of the industrial attachment and further recommendations for improvement.

Chapter Five: Bibliography and References

This includes the bibliography and references used by the student.

Table Of Contents

Overview	i
Table of Contents	ii
Table of Figures	v
Abstract	vi
Acknowledgements	vii
Chapter One : Introduction	1
<hr/>	
1.1 Purpose, Scope and Limitation	2
1.2 Company Background	3
1.2.1 Attached Company - Hewlett-Packard	3
1.2.2 Hewlett-Packard Laboratories in Europe	4
1.2.3 Research Goals	5
1.2.4 E-service and Internet Technologies	5
1.2.5 University Partnerships	6
Chapter Two : Project Background	7
<hr/>	
2.1 The Baskerville Project	8
2.2 The Students' Responsibilities	9
2.3 The E-services Problem Set	10
Chapter Three : Training, Research and Implementation	13
<hr/>	
3.1 The Text Editing Application	14
3.1.1 The Java Programming Language	15
3.1.2 Object Oriented Programming	15
3.1.3 Graphical User Interface	16
3.1.4 Implementation of the Menu Items	18
3.1.5 Problems and Difficulties Encountered	24



3.1.6 Recommendations	25
3.1.7 Conclusion	26
3.2 Text Editing Applet	27
3.2.1 Overview of Applets	28
3.2.2 Standalone and Downloadable Applet Program	28
3.2.3 What Applets Can and Cannot Do	29
3.2.3.1 Security Restrictions	29
3.2.3.2 Applet Capabilities	30
3.2.4 Client/Server Computing with Applets and Servlet	31
3.2.4.1 Servlet Overview	31
3.2.4.2 Sending User-defined Input to a Web Server	33
3.2.4.3 Implementation of the Client Interface	34
3.2.4.4 Implementation of the Web Servlet	35
3.2.5 Java Documentation Comments	38
3.2.6 Unified Modeling Language	40
3.2.7 Problems and Difficulties Encountered	42
3.2.8 Conclusion and Recommendation	43
3.3 Text Editing Service	44
3.3.1 E-speak and E-services	45
3.3.1.1 What is a service?	45
3.3.1.2 E-Speak Overview	46
3.3.2 The Design Overview	50
3.3.3 The Flow of Events	52
3.3.3.1 The Step-by-Step Explanation	52
3.3.3.2 The Flow Charts	57
3.3.4 The Package Structure	59
3.3.4.1 Implementation of the Class Attributes And Abstract Methods	60
3.3.4.2 The Subclasses	62



3.3.5 The New Package Structure	66
3.3.6 Implementation of the Servlet Code	67
3.3.7 Problems and Difficulties Encountered	69
3.3.8 Conclusion and Recommendation	70
3.4 Introduction to the Extensible Markup Language (XML)	71
3.4.1 What is XML?	71
3.4.2 What's Wrong with HTML?	72
3.4.3 Replacing HTML with XML	73
3.4.4 XSL: Converting XML to HTML	74
3.4.5 The Implementation of XML	76
3.4.6 Problems Encountered and Conclusion	76
Chapter Four : The Attachment Conclusion	77
<hr/>	
4.1 Conclusion	78
4.2 Other Experiences Gained	80
Chapter Five : Bibliography and References	81
<hr/>	
5.1 Java and Object Oriented Programming	81
5.2 HTML, XML and XSL	82
5.3 E-services and E-speak	82
5.4 Unified Modeling Language	82



Table Of Figures

Figure 1.1	Hewlett-Packard's First Product	3
Figure 1.2	The Hewlett-Packard Lab in Bristol	4
Figure 3.1a	The Text Editor Application	18
Figure 3.1b	The File Menu	18
Figure 3.1c	The Print Setup Option	20
Figure 3.1d	Flow of Events (Print)	21
Figure 3.1e	The About-about Menu Item	22
Figure 3.1f	The Flow of Events for the Editor Application	23
Figure 3.2	Overview of Applets	28
Figure 3.3	Servlet Overview	31
Figure 3.4	Editor Interface	34
Figure 3.5	Implementation of Web Servlet	35
Figure 3.6	Java Document Comments	38
Figure 3.7	UML Representation of Class Structure	41
Figure 3.8	The e-speak Infrastructure	48
Figure 3.9	The Design Overview	50
Figure 3.10	Step-by-Step Explanation	52
Figure 3.11	The Flow of Events for User	57
Figure 3.12	The Flow of Events for Servlet	58
Figure 3.13	The Package Structure	59
Figure 3.14	Class Structure	60
Figure 3.15	The New Package Structure	66
Figure 3.16	The XSL Processor	74
Figure 3.17	A Sample of XML Document	75
Figure 3.18	A Sample of XSL Document	75



Abstract

The Hewlett-Packard Lab's goal of engaging in world-class research in the interrelated technologies has resulted in a need to invent new solutions to address the challenges of a web economy – from secure e-commerce, high availability systems to electronic content publishing. The Baskerville Project, a commercial Printing e-Services project done by the Publishing Systems and Solutions Unit in Hewlett-Packard Laboratory, Bristol, sets to create and develop an infrastructure to enable desktop access to commercial printing capabilities.

The students' project, which contributed to part of the Baskerville Project, allowed the students to acquire the knowledge of Object Oriented Programming and Software Engineering.

The Text Editor Application, the initial part of the E-service problem set, served as an introduction to object oriented programming in Java. The Text Editor Applet faces an applet security that severely restricts functionality, particularly in file access and printing. The project introduced the use of servlet to handle these functions remotely instead.

The Text Editing Service then introduced a new electronic service, which goes beyond the simple client/server model. The former employs many small service components and make sure of the e-speak framework to define the text editor service and interface.

From the results of the successful project, it is recommended that the Extensible Markup Language, XML, be used in place of the HTML, which has limited internal structure. XSL, a transformation language for transforming a XML document into an HTML document, can be used to convert the Text Editing applet to talk to the e-speak service.



Acknowledgements

This attachment was possible thanks to the cooperation and support of a number of people, who have enabled the student to gain much more than what the scholastic or industrial aspects of the program could have given. The student is grateful to them all, and would like to express his appreciation to the following people:

- Mr. Tom Gardner, his IA supervisor in Hewlett-Packard Lab, for sharing enthusiastically with him his experiences in programming. The student is sincerely indebted to him for taking great pains to keep him on the right track. His support and assistance contributed to the success of the project.
- Dr. Steve Battle (his partner's supervisor in HP Lab) for being extremely patient and for providing advice to the student on his final year project.
- Dr Ian McLoughlin, the student's supervisor from NTU, who has helped in coordinating with the administration stuffs and provide valuable assistance in his logbook and report writing.

The student would also like to express his appreciation to all the staff and colleagues in the PSP departments for their full support and assistance during the attachment, particularly Dr. Anthony Wiley, the project manager.

He would also like to thank his partner Mr Lee Keng Hock for his help and encouragement throughout the entire industrial attachment.



Chapter One

Introduction

This chapter begins with explaining the purpose of the report; follow by its scope and limitations. It then goes on to give a brief introduction on the history of Hewlett-Packard and the attached company, Hewlett-Packard Laboratories, Bristol. This chapter also covers the research goals of Hewlett Packard, as well as a brief explanation on the E-services and Internet Technologies. This chapter ends by providing an overview of the Hewlett-Packard's partnership with the universities.



1.1 Purpose, Scope and Limitation

Purpose

The Industrial Attachment program fulfils part of the requirement in pursuing the degree of Bachelor of Applied Science (Computer Engineering) in Nanyang Technological University. This report serves to summaries the activities and experiences gained with Hewlett-Packard Laboratories, Bristol England.

Scope and Limitations

During the attachment, the student was involved in a commercial printing e-Services group project, known as the ‘Baskerville Project’. This report will only cover parts of the project, which were done by the student. The experience gained during the attachment had helped the students fulfill the objectives of the attachment. However, due to unforeseen circumstances, the commencement of the attachment was delayed. This resulted in a shortage of time for the completion of the project. Therefore, this report will be limited to the stages of implementation prior to the date of reporting. Further work and research done after the date of reporting are not described fully in this report.

The emphasis of this report will be on the research done by the student, in addition to the experience that he gained during the attachment. Certain assumptions need to be made on the reader’s level of familiarity with computing concepts such as object-oriented programming and software engineering etc. Frequent references will be made with regards to these concepts.



1.2 Company Background

1.2.1 Attached Company - HP

HP was founded in 1939 by Bill Hewlett and Dave Packard. The company's first product, built in a Palo Alto garage, was an audio oscillator – an electronic test instrument used by sound engineers. One of HP's first customers was Walt Disney Studios, which purchased eight oscillators to develop and test an innovative sound system for the classic movie "Fantasia".

Today, Hewlett-Packard Company is a leading global provider of computing and imaging solutions and services for business and home. One of HP's major strategies focuses on capitalizing on the opportunities of the Internet and the proliferation of electronic services (e-services).

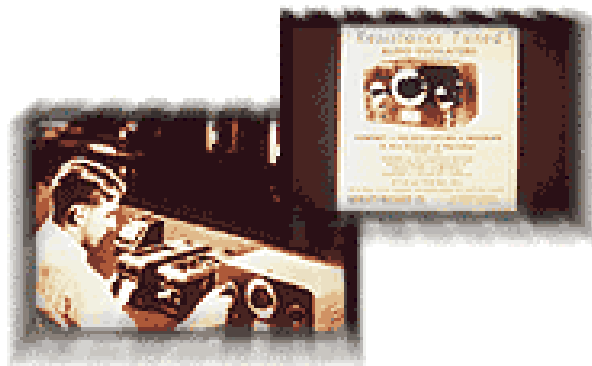


Figure 1.1 HP's First Product



1.2.2 Hewlett-Packard Laboratories in Europe

In September 1984, Hewlett-Packard took the innovative step of starting up a corporate research laboratory in Europe. Hewlett-Packard Laboratories (HPL) is directed by Dick Lampman, with HPL Bristol being HP's largest research center outside its headquarters in Palo Alto, California, and employs around 200 researchers – one third of HPL's total.

Some of HPL's research focuses on e-services, e-publishing and technologies needed for next generation digital imaging. Researchers are coming up with new solutions to address the challenges of a web economy – from secure e-commerce, high availability systems to electronic content publishing; from new ways of browsing the web to devices such as mobile phone, to the display, capture and communications technologies that will be used to build future mobile devices.



Figure 1.2 The Hewlett-Packard Lab in Bristol



1.2.3 Research Goals

HPLB's charter is to engage in world-class research in the interrelated technologies that will enable people to create, manipulate and share electronic information with others wherever they happen to be, for professional and social purposes.

1.2.4 E-services and Internet Technologies

HPLB is working to deliver the technology requirements for the next chapter of the Internet and creating the architecture and tools for the development and deployment of e-services. The program of research in e-services and Internet technologies encompasses a broad range of activities.

HP's Internet strategy revolves around creating and delivering services over the Net. It is built on the vision of e-services and pervasive computing. An e-service is defined as any asset that is made available via the Internet to drive new revenue streams or create new efficiencies. These can be applications, computing resources or services, processes or information. It's about the Internet working for you, rather than you working the Net. At HPLabs, they are inventing the technologies to make this vision a reality.



1.2.5 University Partnerships

HPLB has an extensive network of relationships with selected departments in leading academic institutions in over 20 countries, and invests considerable resources to support them. They are important to HPLB because it shares with the academic community a concern to foster innovative research, and to develop the skills and experience of the people involved in its generation and transfer. Over the past 10 years HPLB has built up a portfolio of programmes to encourage collaboration with universities across a broad spectrum of mutually beneficial activities.

Every year HPLB hosts about 50 multinational students who work as members of project teams in Bristol to gain industrial experience for their academic qualifications. Some of them are jointly funded by their government and HP on ‘industrial PhDs’ designed to forge long-term links between academia and industry. The student population makes a lively contribution to the quality of life at HPLB, and to its international outlook.

HPLB’s university relation’s portfolio is structured with independent but complementary programmes to help universities get to know HPLB through the route most appropriate to their needs. Peer-to-peer contacts can give rise naturally to opportunities for sustained external research activities where there is a good match of technologies and skills.



Chapter **Two**

Project BackGround

This chapter gives the reader a brief idea of the project which the student was involved in. It explains the purpose and direction of the project, as well as the responsibilities of the students in the project. Finally, this chapter gives a simple E-services problem set, which aims to develop core competencies required to build new electronic services.



2.1 The Baskerville Project

BackGround Information

The 'Baskerville Project' is a commercial Printing e-Services project done by the Publishing Systems and Solutions Laboratory in Hewlett-Packard Laboratory, Bristol. The project group consisted of a project manager, 4 other project members, 2 students from Nanyang Technological University and 1 student from Imperial College.

Purpose and Direction

The purpose of the project is to create and develop an infrastructure to enable desktop access to commercial printing capabilities.



2.2 The Students' Responsibilities

The two students would produce components of a workflow concerning outsourced printing. The two main components would involve web front ends to a prototype printing service under development at HP labs. The technical competencies required of the students included object-oriented & Java skills, and the understanding of Internet & e-Service technologies.

Individual Responsibilities

Mr Ivan Low Swee Tieng (Student from Computer Engineering)

To develop a user interface to a print service that allows the user to choose the contents they wish to print, to select from a number of print options, and to accept a printing proposal.

Mr Lee Keng Hock (Student from Electrical and Electronics Engineering)

To develop an interface that allows the user to monitor and manage the progress of a print job. This includes interrogation of the job history and configuration of an email notification system.



2.3 The E-services Problem Sets

The aim of these problem sets is to develop core competencies required to build new electronic services. The students were required to try the problem set so as to gain sufficient experience for implementing the 'Baskerville Project'.

Part One: Text Editing Application

This exercise provided an introduction to object oriented programming using Java. The objective was to build a simple text editing application. The editor was to be able to save and load a document to/from file and print the document. This exercise would require the student to

- install Java Development Kit (JDK)
- develop Project Organization Skill
- use of the AWT and IO packages



Part Two: Text Editing Applet

This exercise provided an introduction to distributed computing. The objective was to convert the text editing application to run as a downloadable applet. This raised issues regarding the applet security, which severely restricts functionality, particularly in file access and printing. These functions might be remotely handled by a servlet. The students were required to

- set up a Web Server (install Linux/Apache)
- learn Client/Server computing with Applets and Servlets (install JDSK)

Part Three: Text Editing Service

New electronic services go beyond the simple client/server model, employing many small service components. The e-speak framework would be used to define a text editor service and its interface. The objective was to convert the existing text editing applet & servlet to work in the e-speak framework. This would require

- installing e-speak
- thinking about Services, Interfaces and Vocabularies



Part Four: Text Editing

The e-speak client cannot be downloaded by a browser for access via the Internet. But a Java applet can speak to an e-speak service using the eXtensible Markup Language, XML. The objective was to convert the text-editing applet to talk to the existing e-speak service. This would require

- installing e-speak Web-Access
- using XML



Chapter **Three**

Training, Research and Implementation

This chapter covers all the training and experiences that the student has gained during the attachment. The student and his partner have done the implementation of the project, however only that carried out by the student himself will be discussed in detail. The research and implementation done by the student are described in different phrases, according to the E-services problem set. The reader should note that most of the time spent during the attachment was for training and research, followed by the implementation. Therefore, this chapter will cover the training and exploratory research carried out by the student first, followed by the results of the implementation.



3.1 The Text Editing Application

The Objective

The objective of this problem is to build a simple text editing application, which aims to provide an introduction to object oriented programming using Java. The editor should be able to save and load a document to/from file and print the document.

Overview

The first part of the project, which required the student to create a Text Editor Application, introduced the concept of object oriented programming in Java. The student made use of the **Abstract Windowing Toolkit (AWT)** to design the graphical user interface (GUI) for the editor. The editor comprised a frame, menu bar and menu items, and also command buttons, if required. The student also decided on the possible menu items to be included in the program, as well as their individual implementation.

Basic Requirements

- Java Development Kit (JDK)
- Basic Programming Skill
- Text Editing software



3.1.1 The Java Programming Language

The Java programming language is a state-of-the-art, object-oriented language that has a syntax similar to that of C. The language designers strove to make the Java language powerful, but at the same time, they tried to avoid the excessively complex features that have bogged down other object-oriented languages, such as C++. By keeping the language simple, the designers also made it easier for programmers to write robust, bug-free code.

3.1.2 Object Oriented Programming

Based on the knowledge of function-oriented C Programming, it is not difficult to write the Editing Application. The challenge, however, is to use a totally new language, Java, and to program using Object Oriented Programming.

The main difficulty is how to think in an Object Oriented way. In Function Oriented Programming, all the user has to do is to write a function to solve a problem. However, in Object Oriented Programming, it is a totally different approach. In Object Oriented Programming, the user will have to think of the problem as a set of objects. Then, the user will have to define the class and the methods, which the object may require, as well as the different attributes and their values.

To have a better understanding of object oriented programming in Java, visit the web site at <http://web2.java.sun.com>, which gives an introduction of Java programming.



3.1.3 Graphical User Interfaces

The first step to any interface program is to create the interface of the program. The interface of the text-editor requires only a frame, a menu bar with some menu items and command buttons. This in turn, requires the **Abstract Windowing Toolkit (AWT)**, which defines all the GUI components in Java. Note that these components comprise the `java.awt` package.

There are basically four steps to creating a GUI. The steps are listed below with an example given for each of the individual step.

Step 1: Create the component

A GUI component can be easily created just like any other object in Java, simply call the constructor.

E.g. `Button open = new Button (“Open”);`

Note that the components are typically created in the *init ()* method of an applet or in the constructor of a standalone application (or in a private method invoked by one of those methods).



Step 2: Add the component to a Container

All components must be placed within a container. Containers in Java are all subclasses of **java.awt.Container**.

E.g. `this.add (open);`

Step 3: Layout the components within their containers

A **LayoutManager** object can be used to automatically lay out the components of a container according to certain layout rules defined by the particular **LayoutManager** chosen.

E.g. `this.setLayout (new BorderLayout (10, 10));`

Step 4: Handle the events generated by the components

When working with GUI, it is more common to handle the higher-level semantic events that are generated by the components themselves. Java 1.1 provides different event-handling models, which can be used.

E.g. `public void mouseDragged (MouseEvent e) {`



3.1.4 Implementation of the Menu Items

The menu bar and its menu items formed the major part of the program. It is the menu items that tell the program what the user wants to do. It is also the main graphical interface for the user. *Figure 3.1a* below shows the Text Editor interface implemented by the student.

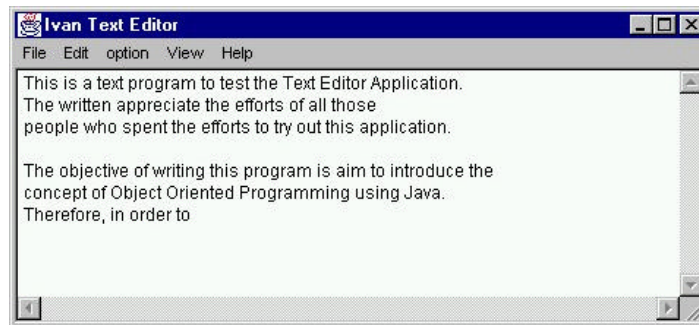


Figure 3.1a The Text Editor Application

The following sections explain the individual menu items and their functions. Refer to *Figure 3.1b* below for the graphical representation of the menu items.

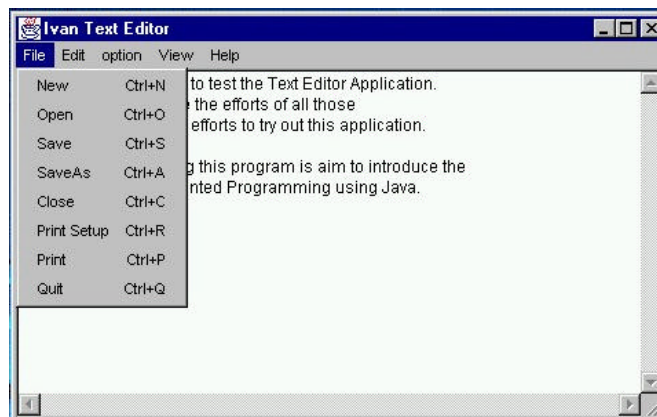


Figure 3.1b The File Menu



File-New

This will clear the contents in the editor text area. A new file with no initial filename will be created.

File-Open

This menu item allows the user to open a file from anywhere within the system. The File-Open menu item displays an Open Dialog Box for the user to choose the file they want. The user may choose an existing filename or create a new file.

File-Save

This menu item allows the user to save the file without having to specify the filename. This means that the contents of the editor will be saved into the same file, or rather, the default filename.

File-SaveAs

This will be different from Save as this will require the user to enter a filename. There will be no default filename available. The user normally uses this function instead of the Save function when they want to specify a filename.



File-Close

This will close the existing file. It will clear all the contents in the text area.

File-PrintSetup

This will allow the user to define how they want their printer layout to be like. The editor allows the user to specify the type of paper size most suitable for the document, the printing orientation, the number of pages etc. Refer to *Figure 3.1c* for the diagram.

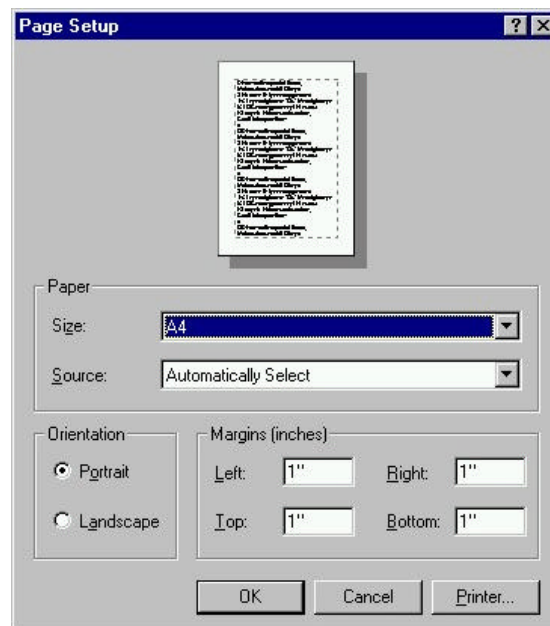


Figure 3.1c The Print Setup Option



File-Print

This allows the user to print a hardcopy of the contents of the editor. The student has to ensure that the program is able to print according to the user specification. For example, the user may want a margin of 2 inches, a landscape orientation, start printing from page 3 to page 6, and print 4 copies each etc. A flowchart of how the program takes care of all these is shown in *Figure 3.1d* below.

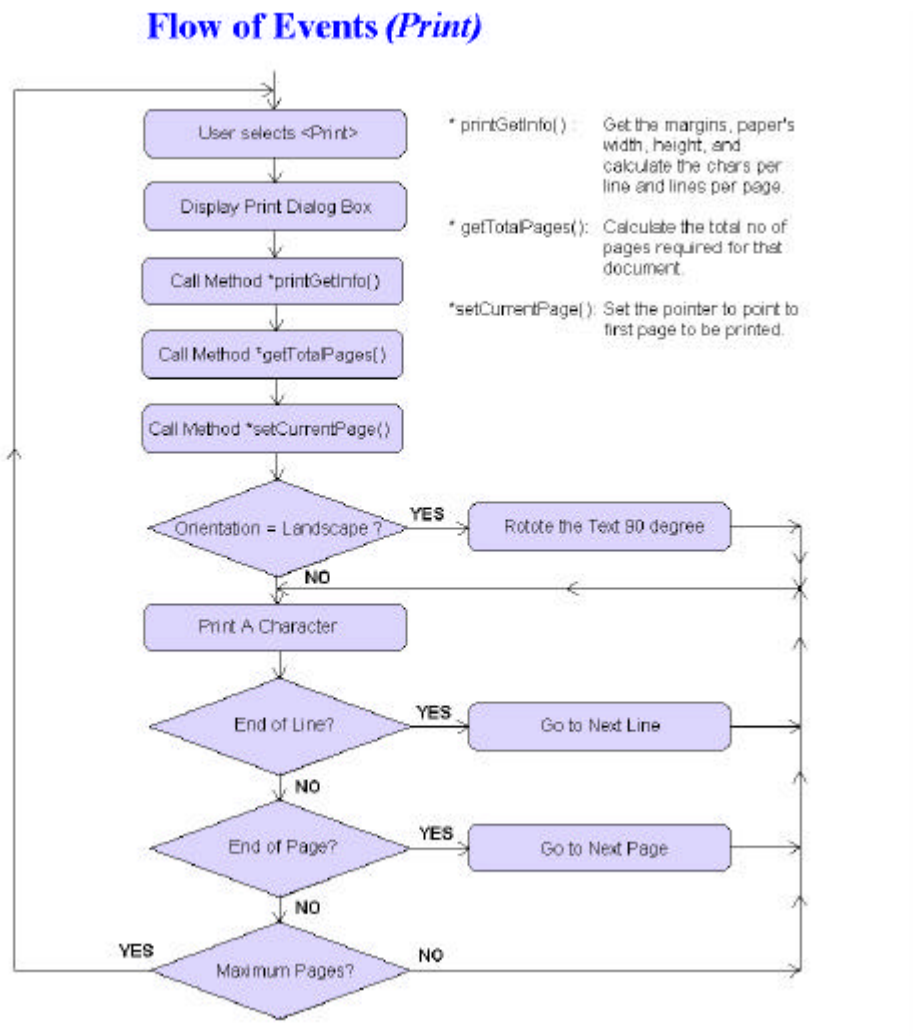


Figure 3.1d Flow of Events (Print)



File-Quit

This will exit from the editor.

About-about

This will display a new window that shows the date and author of the editor. Dialog boxes in Java are implemented using the **Dialog** class. An information dialog is a dialog box that lets the writer specify the message to be displayed. This dialog box makes use of a **MultiLineLabel** class that displays multiple static lines of text, which is something that the **AWT Label** class is incapable of doing.

Much of the code is taken up with the mechanics of breaking the label into separate lines and measuring the size of each of those lines. It made use of the `paint ()` method, `getPreferredSize ()` and `getMinimumSize ()` method. Refer to *Figure 3.1e* for the graphical representation of the frame.

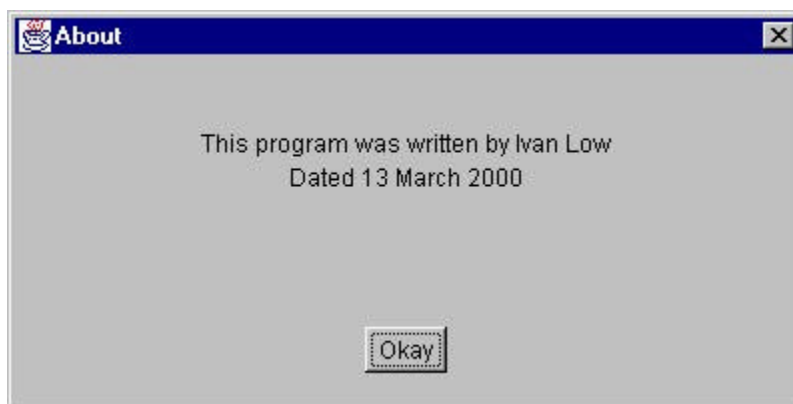


Figure 3.1e The About-about Menu item



Flow of Events

Figure 3.1f below shows all the possible flow of events that may occur. The user is allowed to select any of the following commands from the menu. Note that this program only serves as an introduction to the actual project, and at the same time illustrating the concepts of object oriented programming in Java, which is an essential element for the progress of the actual project.

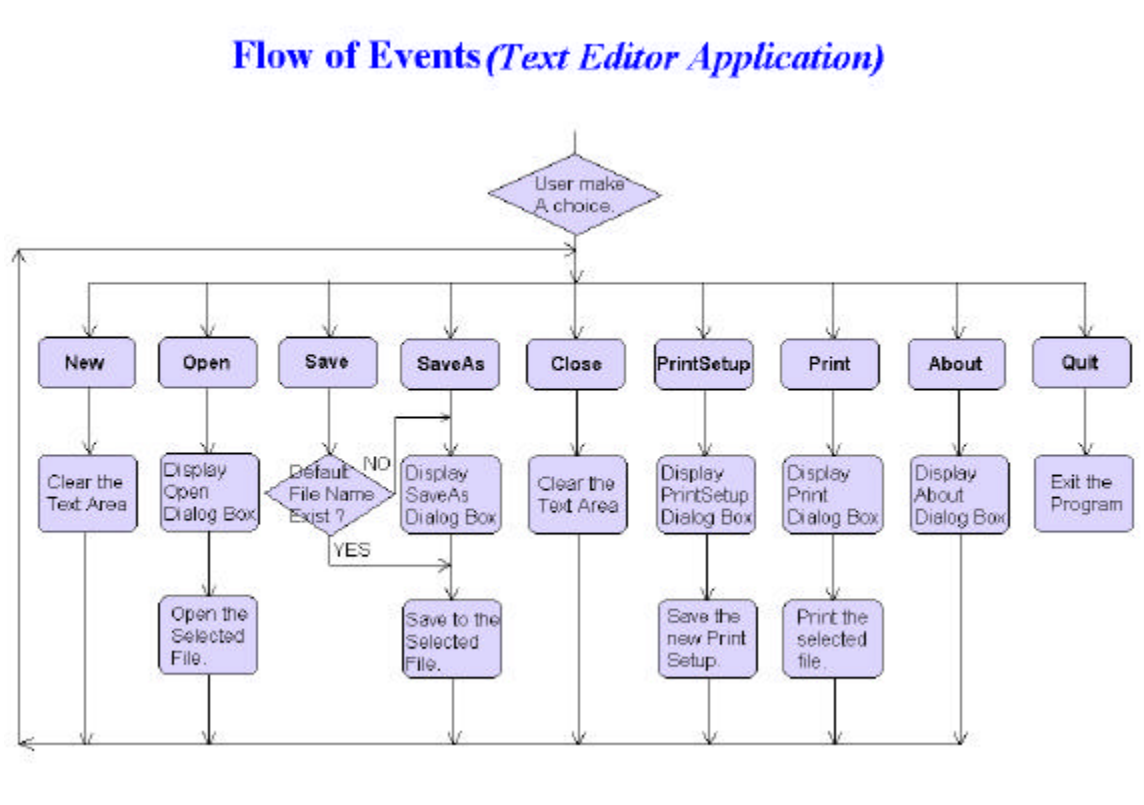


Figure 3.1f The Flow of Events for the Editor Application



3.1.5 Problems and Difficulties Encountered

The main difficulty that the student encountered was the implementation of the program using the concept of Object Oriented. The student, with 4 years of programming experiences in C using Function Oriented concept, found it hard to adapt to using a totally different concept. Although the student was using objects and classes to write a program, there still existed a mixture of both programming concepts. It took the student quite some time to really understand how to make full use of the concept of Object Oriented Programming, such as the superclasses, object and the class hierarchy.

Not being able to understand fully the concept of Object Oriented Programming, and also the syntax of Java language, the student also found it hard to organize the program structure in the initial stage. As the student progress further, he started to realize that the code became very long and disorganized. Many parts of the codes would be redundant if the concept of Object Oriented Programming was used in the first place.

Another difficulty encountered by the student was to print the document according to the specification of the user. The student has to ensure that the program is able to print according to the different margins set by the user, the different paper sizes, the different orientation, such as landscape and portrait, and also according to the specified start and end pages. This required the student to have very good knowledge on the superclass **Printable** in the Java API, as well as the page format, e.g. the DPI, the font size, font height and font ascent etc.



3.1.6 Recommendations

The first recommendation that the student wished to make is for the writer to have sufficient knowledge of the concept that he is going to use, in this case Object Oriented, before he really start on the actual program. It is advised that the **Unified Modeling Language (UML)** be used at the design stage to get a clearer picture of how the classes and methods are to be defined or constructed. This will help in ensuring that the writer will be able to make full use of each individual class, its properties and methods, and also the advantages of class inheritance, method overridden etc.

Another recommendation for the Text Editor application is to use the **Java Document Comment**. The student did not realize the usefulness of Java Doc, until the later part of the attachment when the supervisor told him. Although the student still managed to use the Java Doc for certain parts of the program codes, implementing it at an earlier stage would have helped the student in debugging, and at the same time, ensured that the supervisor and everyone else had a good understanding of the program.

Finally, the student would like to suggest that the program have a help file (if time allowed). A new menu item, called Help, should be included in the menu bar to assist those users who may not have enough computer knowledge. This Editor program should cater to anybody, thus a help file would be helpful.



3.1.7 Conclusion

The implementation of the first part of the project enabled the student to get used to writing a program using the Object Oriented concept and to know more about the new programming language, Java. It was a very useful experience for the student, as Java programming generally improves the programmer's efficiency. Java is a simple and elegant language with a well-designed, intuitive set of APIs. Programmers are able to write better code with fewer bugs than with other platforms, thus reducing development time.

Being able to use the Object Oriented concept to design and implement a problem, the student now found himself more confident in implementing the next stage of the project, which required the student to make use of applet and client/server communication.



3.2 Text Editing Applet

The Objective

The objective of this problem was to convert the text editing application to run as a downloadable applet. This part of the project provided an introduction to distributed computing. The student was also able to experience the **applet security** that severely restricts functionality, particularly file access and printing. A servlet was then introduced to handle this problem.

Overview

In the second part of the project, the student created a downloadable applet, using the existing Text Editor Application. The current Text Editor Application was only able to run as a standalone program. It was not able to run in any web browser. To convert it to run as a downloadable applet meant enabling it to run in any web browser. In the later part, the student discovered that the applet had certain security rules, which restricted the functionality of the Editor program. Thus, the student decided to use the servlet to handle that problem.

Basic Requirements

- Java Servlet Development Kit (JSDK)
- Web Server



3.2.1 Overview of Applets

Every applet is implemented by creating a subclass of the `Applet` class. The following figure shows the inheritance hierarchy of the `Applet` class. This hierarchy determines much of what an applet can do and how.

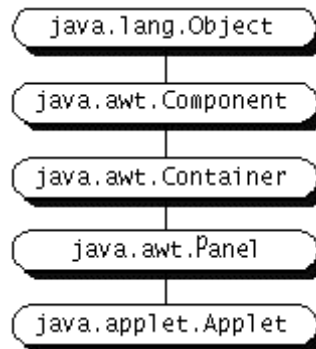


Figure 3.2 Overview of Applets

3.2.2 Standalone and Downloadable Applet Program

In the previous editor program, the program was only allowed to run as a standalone program. In this part of the project, the text editor was improved so that it could be displayed using any web browser, such as the Internet Explorer. To do this, the program had to be written using Applet. However, there were some security restrictions, which limited the use of applet in certain applications. The next section gives an overview of what an applets can and cannot do.



3.2.3 What Applets Can and Cannot Do

This section gives an overview of both the restrictions applets face and the special capabilities they have.

3.2.3.1 Security Restrictions

Every browser implements security policies to keep applets from compromising system security. This section describes the security policies that current browser adhere to. However, the implementation of the security policies differs from browser to browser. Also, security policies are subject to change. Current browsers impose the following restrictions on any applet that is loaded over the network:

- An applet cannot load libraries or define native methods.
- It cannot ordinarily read or write files on the host that's executing it.
- It cannot make network connections except to the host that it came from.
- It cannot start any program on the host that's executing it.
- It cannot read certain system properties.
- Windows that an applet brings up look different from windows that an application brings up.

Each browser has a `SecurityManager` object that implements its security policies. When a `SecurityManager` detects a violation, it throws a `SecurityException`. The applet can catch this `SecurityException` and react appropriately.



3.2.3.2 Applet Capabilities

The `java.applet` package provides an API that gives applets some capabilities that applications don't have. For example, applets can play sounds, which other programs cannot do yet.

Here are some other things that current browsers and other applet viewers let applets do:

- Applets can usually make network connections to the host they came from.
- Applets running within a Web browser can easily cause HTML documents to be displayed.
- Applets can invoke public methods of other applets on the same page.
- Applets that are loaded from the local file system have none of the restrictions that applets loaded over the network do.
- Although most applets stop running once you leave their page, they do not have to.

To have a better understanding of the applets, please refer to the website at the following

URL: <http://web2.java.sun.com/docs/books/tutorial/applet/index.html>

Directly or indirectly, this website covers everything one needs to know to write a Java applet.



3.2.4 Client/Server Computing With Applets and Servlet

Due to the applet security, which severely restricts functionality, the downloadable applet editor program will not be able to access any file, which means that the user will not be able to open, save or print any file. These functions, however, can be remotely handled by a servlet.

3.2.4.1 Servlet Overview

Servlets are modules that extend request/response-oriented servers, such as Java-enabled web servers. For example, a servlet might be responsible for taking data in an HTML order-entry form and applying the business logic used to update a company's order database.

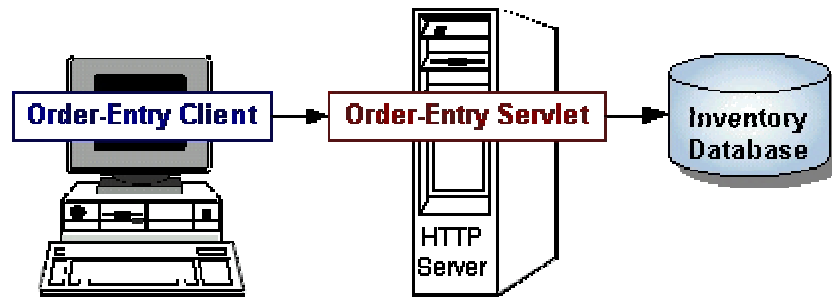


Figure 3.3 Servlet Overview



Servlets are to servers what applets are to browsers. Unlike applets, however, servlets have no graphical user interface. Servlets can be embedded in many different servers because the servlet API, which is used to write servlets, assumes nothing about the server's environment or protocol. Servlets have become most widely used within HTTP servers; many web servers also support the Servlet API.

Other Uses for Servlets

Here are a few more applications for servlets:

- Allowing collaboration between people. A servlet can handle multiple requests concurrently, and can synchronize requests. This allows servlets to support systems such as on-line conferencing.
- Forwarding requests. Servlets can forward requests to other servers and servlets. Thus servlets can be used to balance load among several servers that mirror the same content, and to partition a single logical service over several servers, according to task type or organizational boundaries.

For a more detailed explanation on servlets, the user is advised to refer to the web site at: <http://web2.java.sun.com/docs/books/tutorial/servlets/index.html>

The **Servlets** trail teaches one about servlets, the bodies of code that run inside servers and extend their functionality.



3.2.4.2 Sending User-Defined Input to a Web Server

The first thing to do is to get a user input from a web browser to a web server. Fortunately the HTTP protocol provides two main ways to send information to a web server above and beyond the URL of a requested file, that is, the POST and GET methods.

The GET Methods

The foundation of HTTP/0.9 (the first implementation of the HTTP protocol) was the definition of the GET method that was used by a web browser to request a specific document. The web browser formulates the actual GET request, sends it to the web server, receives the HTML document back, and then displays the HTML document according to the HTML instructions.

Problems With The GET Methods

Though the GET method was very useful, a couple of serious problems remained. First, the GET method only allowed a limited amount of data (1024 characters) to be sent as URL encoded data. If there were too many name/value pairs, some of them would be clipped and data would get lost. Further, since the information was sent as part of the URL, the user could see all of that data. On the one hand, that made URL's look really ugly and scary. On the other hand, it meant that the user got to see all of the inner workings of the CGI input. These all changed with the development of HTTP/1.0.



The POST Methods

The HTTP/1.0 protocol was developed from 1992 to 1996 in order to satisfy the need to exchange more than simple text information. The POST method of input was one of the important changes brought about by the introduction of HTTP/1.0. The POST method allowed web browsers to send an unlimited amount of data to a web server by allowing them to tag it on to an HTTP request after the request headers as the message body. Typically, the message body would be the old familiar encoded URL string after the question mark (?).

3.2.4.3 Implementation of the Client Interface

An editor interface was created as the client, using HTML. Microsoft FrontPage was used as a straightforward HTML editor. The editor will only require a Text Input for the filename, a Text Area for the user to edit the text, and a couple of Buttons for the different functions. A menu will no longer be needed. A sample of the editor interface is shown in *Figure 3.4*.

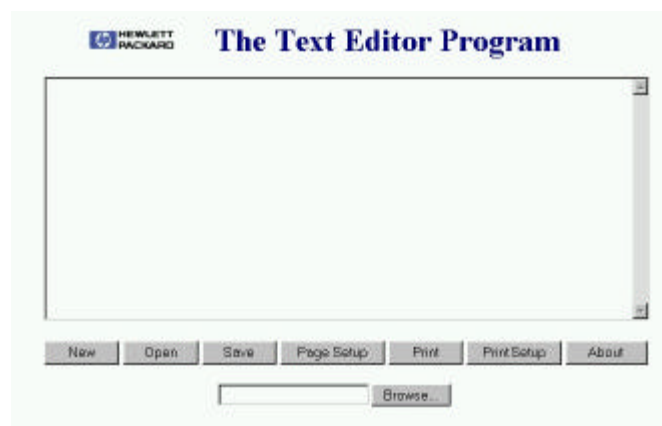


Figure 3.4 Editor Interface (Client)



3.2.4.4 Implementation of the Web Servlet

The web servlet is supposed to handle the information sent over by the client. The client program used the **Post** method of input to send an unlimited amount of data to the web server, and the servlet used the **doPost** method to handle the data received. The servlet then determine whether the user wants to open, close, save or print a file at the web browser. The servlet will then create a new HTML form with all the required information and display it on the web browser.

The diagrams and explanation below describe how the client activates the servlet to perform the expected task.

In *Figure 3.5a* below, the Editor Program will only activate the servlet when the user select any of the command buttons below.

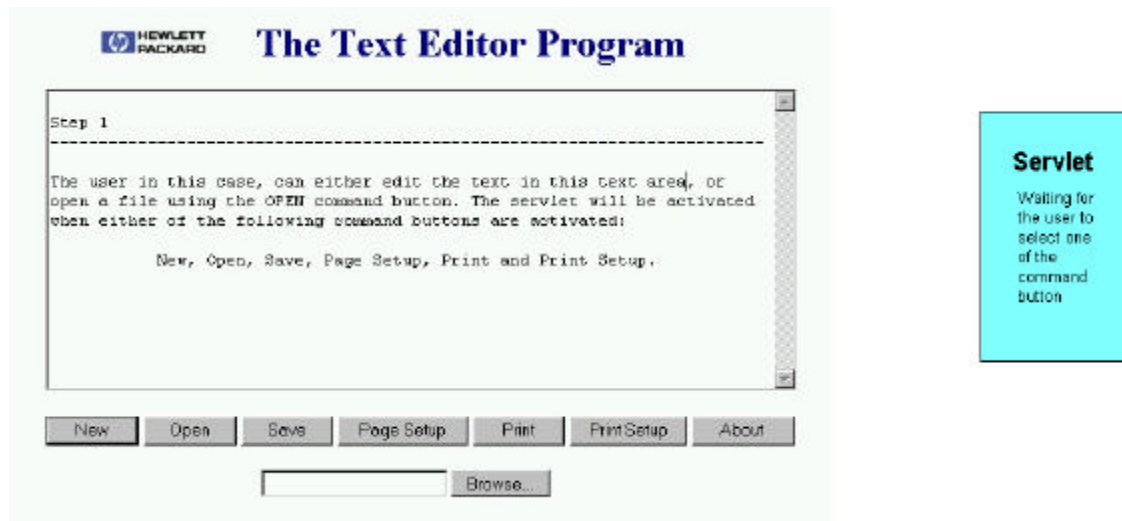


Figure 3.5a Implementation of the Web Servlet: Step 1



However, the user can also choose to edit the text in the text area or use the browse button to select or view the filename. Both options will not activate the servlet as they do not require any access to file. The servlet will only be activated when the user select a function that required access to any file.

When either of the command buttons is selected, the Editor Program (Client) will then send the information to the servlet. The servlet will now determine which button is selected and react accordingly to what the user wanted. This is shown in *Figure 3.5b* below.

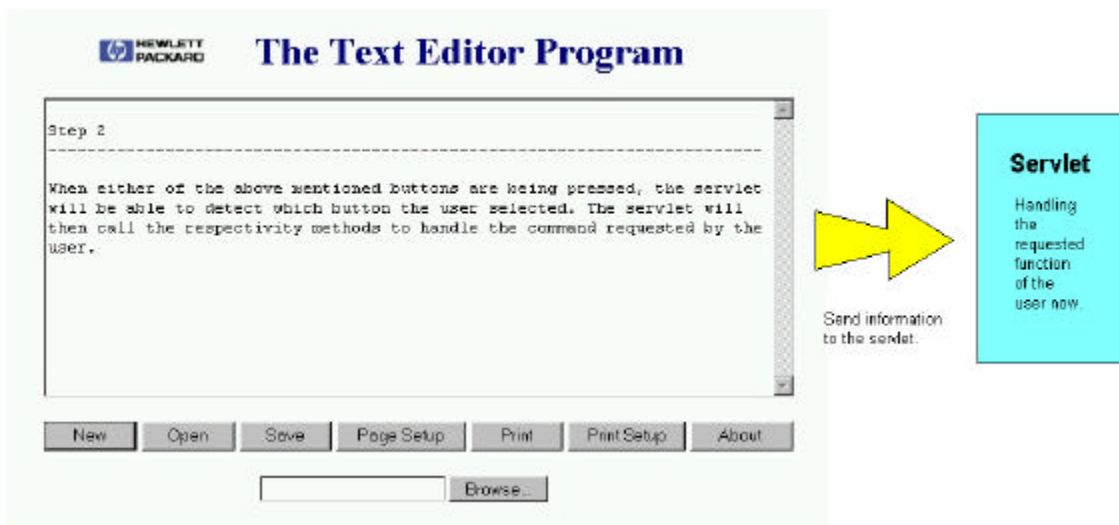


Figure 3.5b Implementation of the Web Servlet: Step 2

When the servlet finished executing the task expected by the user, the servlet will then reprint a new copy of the Editor Interface on the browser and wait for the next command button to be selected. This is again shown in *Figure 3.5c* below.

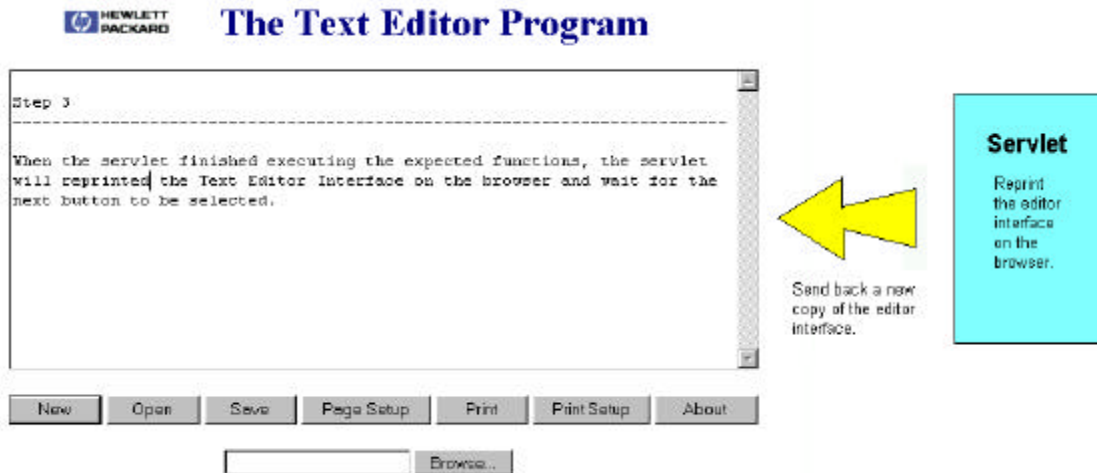


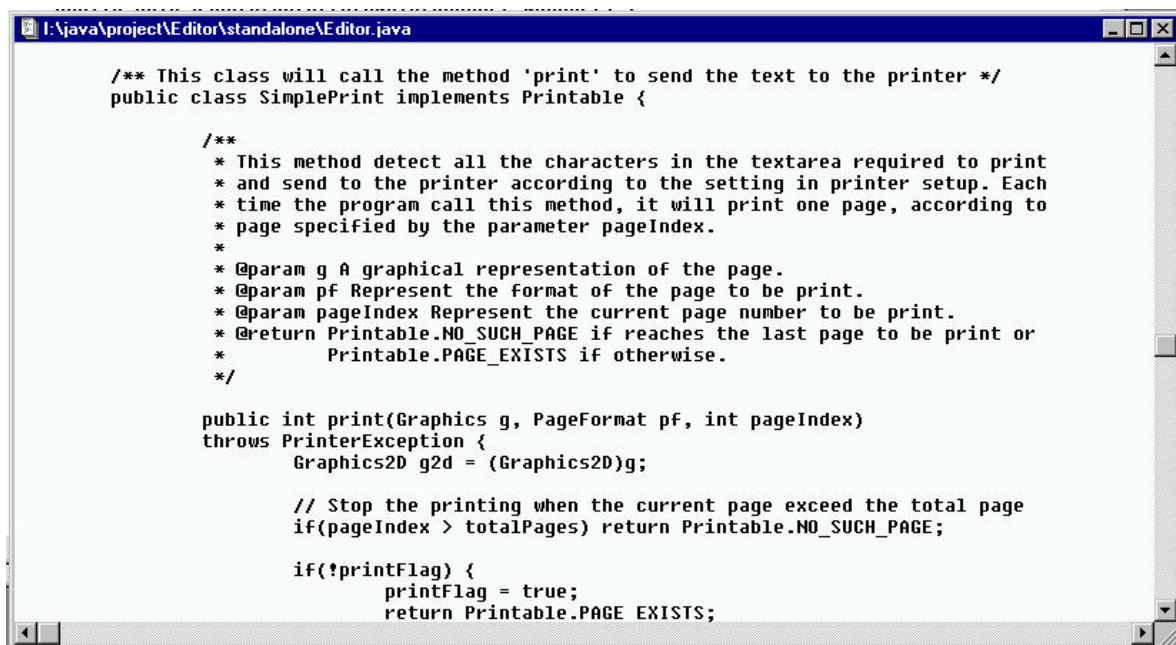
Figure 3.5c Implementation of the Web Servlet: Step 3

This basically complete the second part of the problem set, which is to use the servlet and applet application to create a downloadable Editor Program. The next section will cover two other aspects learned during the implementation of the project: the **Java Documentation Comments** and the **Unified Modeling Language (UML)**.



3.2.5 Java Documentation Comments

Most ordinary comments within Java code explain the implementation details of that code. In contrast, the Java language specification defines a special type of comment known as a doc comment that serves to document the API of the code. A doc comment is an ordinary multiple line comment that begins with `/**` (instead of the usual `/*`) and ends with `*/`. The following figures show the implementation of the java doc, as well as the output of the java doc in a web browser.



```

I:\java\project\Editor\standalone\Editor.java
/** This class will call the method 'print' to send the text to the printer */
public class SimplePrint implements Printable {

    /**
     * This method detect all the characters in the textarea required to print
     * and send to the printer according to the setting in printer setup. Each
     * time the program call this method, it will print one page, according to
     * page specified by the parameter pageIndex.
     */
    * @param g A graphical representation of the page.
    * @param pf Represent the format of the page to be print.
    * @param pageIndex Represent the current page number to be print.
    * @return Printable.NO_SUCH_PAGE if reaches the last page to be print or
    *         Printable.PAGE_EXISTS if otherwise.
    */

    public int print(Graphics g, PageFormat pf, int pageIndex)
    throws PrinterException {
        Graphics2D g2d = (Graphics2D)g;

        // Stop the printing when the current page exceed the total page
        if(pageIndex > totalPages) return Printable.NO_SUCH_PAGE;

        if(!printFlag) {
            printFlag = true;
            return Printable.PAGE_EXISTS;
        }
    }
}

```

Figure 3.6a Java Document Comments



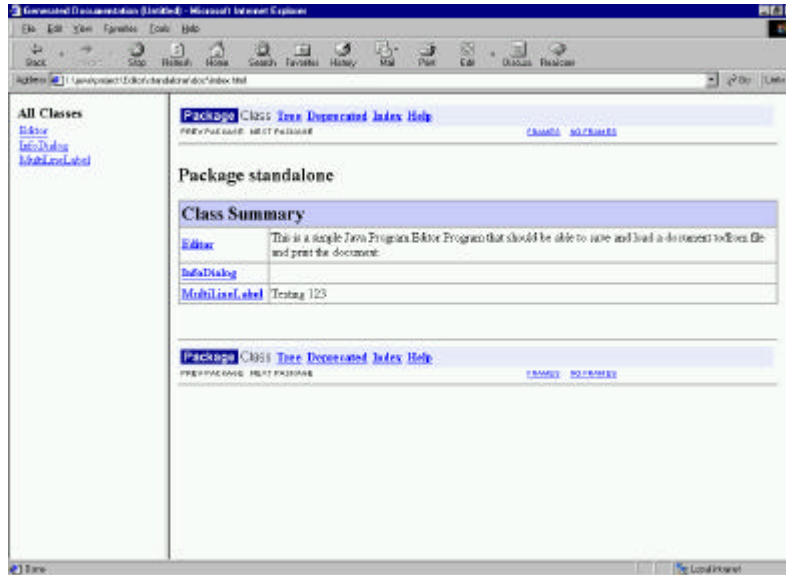


Figure 3.6b The Package Structure

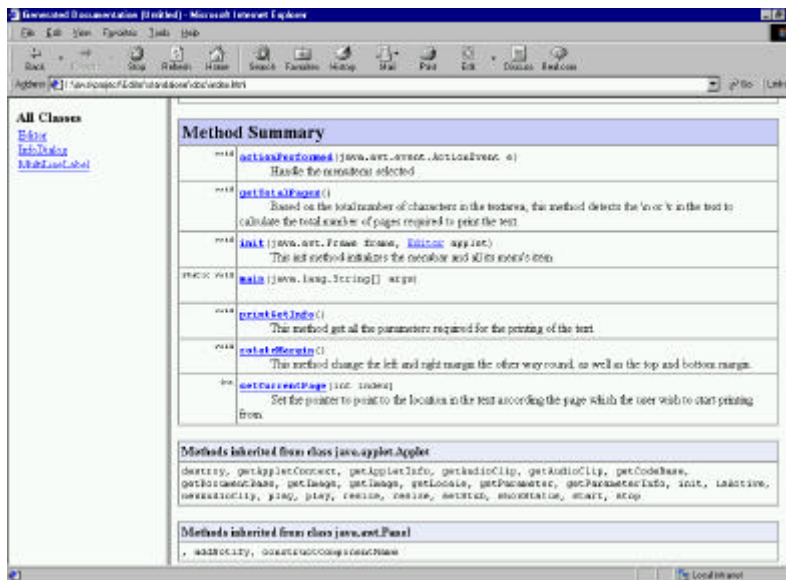


Figure 3.6c Java Method Summary



3.2.6 Unified Modeling Language

The UML is the standard language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. It can be used with all processes, throughout the development life cycle, and across different implementation technologies.

The UML combines the best of the best from

- Data Modeling concepts (Entity Relationship Diagrams)
- Business Modeling (work flow)
- Object Modeling
- Component Modeling

In this project, the student made use of the UML modeling elements in class diagrams, that define the classes and their structure and behavior, association, aggregation, dependency and inheritance relationships, multiplicity and navigation indicators and role names.



A simple UML model is shown in *Figure 3.7* below.

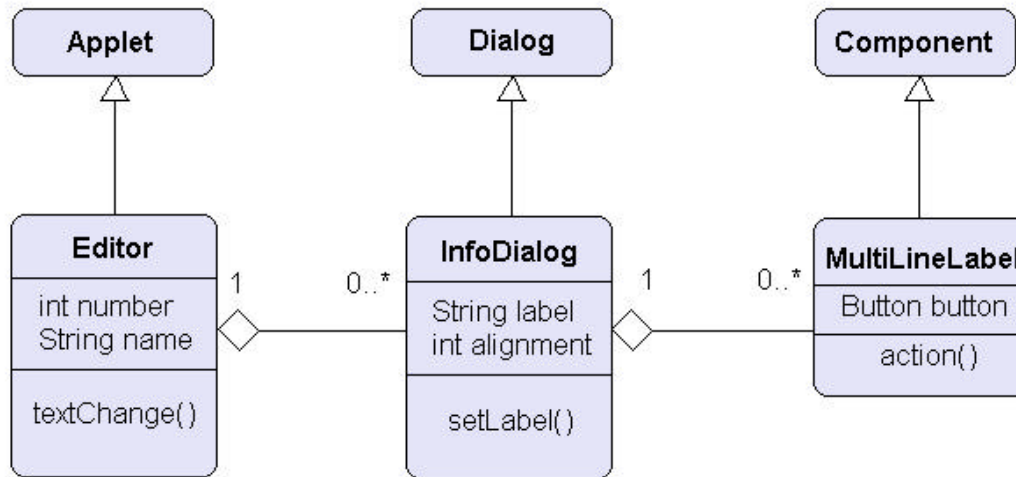


Figure 3.7 UML Representation of Class Structure

In the above example, class **Editor** is a subclass of **Applet**. It has two attributes; number and name, and a method call `textChange ()`. It can consist of 0 to infinity number of objects, which is an instance of class **InfoDialog**. **InfoDialog**, on the other hand, is a subclass of **Dialog**. It has two attributes; label and alignment, and a method called `setLabel()`.

This simple example illustrates part of the program code done by the student on the Text Editor Applet. To have a better understanding in UML, please refer to the web site at

<http://www.rational.com/uml/index.jtmpl>



3.2.7 Problems and Difficulties Encountered

The first problem that the student encountered was the limitation of the GET method. The client, which is also the web browser, made use of the GET method to communicate with the servlet. The web browser formulates the actual GET request, sends it to a web server, receives the HTML document back, and then displays the HTML document according to the HTML instructions. However, the student did not notice that the GET request only allowed a limited amount of data (1024 characters) to be sent as URL encoded data. Therefore when the user selected a huge file to be saved or print, the servlet was unable to process the request. This problem was solved when the student discovered the new POST method, which was introduced by the HTTP/1.0. The POST method allowed web browsers to send an unlimited amount of data to a web server by allowing them to tag it on to an HTTP request after the request headers as the message body. For a more detailed explanation on servlets, the reader is advised to refer to the web site at: <http://web2.java.sun.com/docs/books/tutorial/servlets/index.html>

Another problem that the student encountered is changing the output in the HTML interface, after the users made their selection. In HTML, the whole interface has to be reprinted by the servlet, even if it only consists of a small change in the output. This in fact lengthens the program code, and at the same time, limited the flexibility of the editor interface at run time.



3.2.8 Conclusion and Recommendation

In this part of the attachment, the student has managed to understand how the client and server can communicate to each other, overcoming the security restrictions imposed by the applet. However, using the HTML form to represent the client application was not a really good way. This is because the HTML form requires the servlet to reprint the whole interface whenever a change needed to be imposed (reflected) on the client interface. This lengthens the servlet program, and is not a good way of organizing the program structure. The alternative method of using Java code to write the applet application may be a better option. The only problem is that it will be more tedious to draw the graphical user interface using Java API, compared to using HTML. However, the Java allowed changes to be made on the interface without reprinting the whole interface. Due to the time constraint, this suggestion is not implemented.

Thus, this section can be concluded with the understanding of the application of applets, client/server communication, and the use of servlet to overcome the limitation, or security restrictions of the applets. The next section will cover the next part of the project, which required the student to acquire the knowledge of **E-services**, **E-speak**, and **XML**.



3.3 Text Editing Service

The Objective

This part of the project covers both the third and last part of the E-services problem set. These parts of the problem set have been done concurrently by the student and his partner. The objective is to use the e-speak framework to define a text editor service and its interface. The existing text editing applet will be converted to talk to the e-speak service.

Overview

This is the final part of the problem sets, and it requires the student to acquire the knowledge of e-speak and the XML. New electronic services go beyond the simple client/server model, employing many small service components. This project will use the e-speak framework to define a text editor service and its interfaces. However, the e-speak client cannot be downloaded by a browser for access via the Internet. A Java applet can speak to an e-speak service using the extensible Markup Language, XML. Thus, the text-editing applet will be converted to talk to the existing e-speak service.

The Requirements

- E-speak
- E-speak web-access
- Understanding of e-services, interfaces and vocabularies
- XML / XSL



3.3.1 E-speak and E-services

Please note that only the research and training done on the e-speak and services will be covered in this report. The actual implementation of e-speak and services, done by the student's partner, will not be covered here.

3.3.1.1 What is a service?

A service is traditionally considered to be a stand-alone application with a well-defined interface. For example, an instance of an ERP system, such as a SAP instance, may be considered a service. Similarly, a file system is considered a service.

In the e-speak world, a service is not necessarily restricted to a unit such as an application. Instead, the granularity of a service is much smaller. Think differently about what constitutes an e-speak service. Anything that can be described with a unique set of attributes is a service. Functions are invoked directly on this service.

Reference for E-service: <http://www.hp.com/e-services>



3.3.1.2 E-Speak Overview

E-speak is an open software platform designed specifically for the development, deployment and intelligent interaction of e-services.

With e-speak, users and e-services can interact regardless of their hardware or operation systems, system management strategies, development environments, or device capabilities.

Any service can become manageable by using the e-speak service management framework. Services do not need to explicitly add mechanisms to become manageable. The e-speak event infrastructure enables publish, distribute and subscribe operations across a geographically distributed environment and across multiple protection domains. This event infrastructure coupled with the security and management framework supported in e-speak, provide a comprehensive platform for managing service deployments that require dynamic business collaborations.



The e-speak engine is a software that performs the primary e-speak functions of:

- ***Discovery***

Once an e-service is e-speak-enabled, the provider registers it with a host system connected to and accessible by the Internet. During registration, the provider creates a description of the e-service that consists of its specific attributes. Users looking for e-services then describe the type of service they want and e-speak will automatically discover registered services that have the desired attributes.

- ***Negotiation***

After discovering e-service providers, e-speak negotiates between the requester and the provider to weed out any that offer services outside the criteria of the request.

- ***Mediation***

Once a user and an e-service have been brought together, e-speak is able to continuously monitor service delivery and make adjustments, or “mediate”, in real time. No other Internet technology or standard available today performs this function.

- ***Composition***

In the near future, e-speak-enabled e-services will be able to combine themselves into more complex, cascading e-services, even-on-the-fly.



A Simple Example

The following section provides a simple example of an e-speak Service written that illustrates some of the basic ideas in the e-speak infrastructure.

Client Service Discovery

A Client first creates a new connection to an e-speak Core. After connecting to the Core, the Client can look up or register Services. The Client locates a Service that satisfies a constraint expressed with attributes in the default Vocabulary. The result of the find Service is a stub (or proxy) to the Service provider's Service. Clients can use this stub as a network object reference and directly invoke methods on the Service (see *Figure 3.8*).

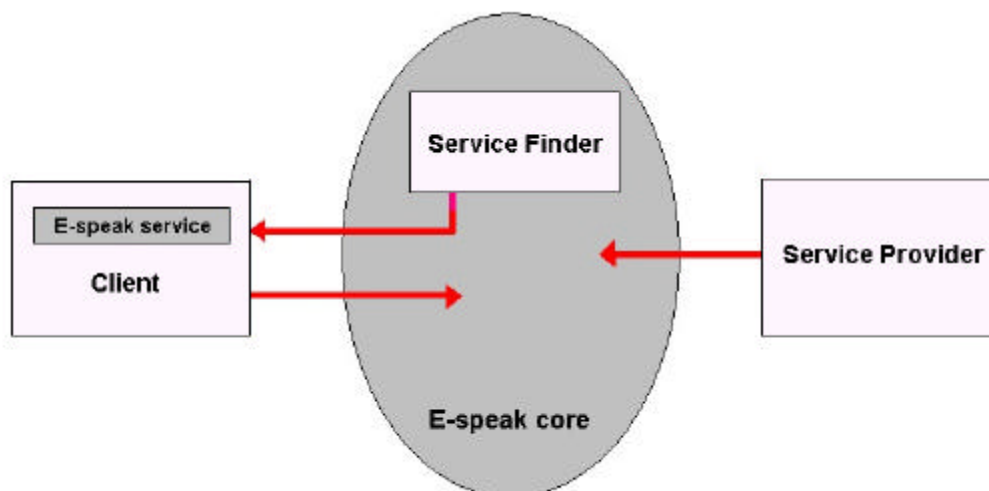


Figure 3.8 The e-speak Infrastructure

Client Service Usage

Clients interact with the Service with the set of interfaces for which stubs are available in the Client address space. Clients can preinstall the Service stubs that are generated using the e-speak IDL stub generator, or they may acquire the stub class from the Service provider by other means. When a Client invokes an operation, a well-defined e-speak custom serialization is used to ship the invocation to the target Service through the mediating e-speak infrastructure. In so doing, all method invocations are effectively mediated.

Reference for E-speak: <http://www.e-speak.net>



3.3.2 The Design Overview

The Text Editor program was rebuilt using e-speak and HTML. It looked like the old Text Editor, but was much better. The student looked at the front-end which has to work out what goes into the dialog box using information from the print service. His partner, on the other hand, built the back-end services and worked out how to talk to them.

THE DESIGN OVERVIEW

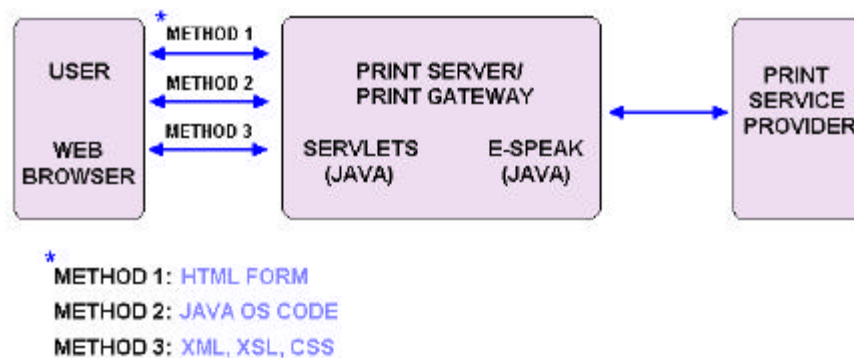


Figure 3.9 The Design Overview

User: This is a web browser, such as the Internet Explorer, which allows the user to view the software. This is also considered as the client.

Print Server: This is the servlet where the program codes of both the student and his partner will speak to each other.

Print Gateway: This is where all the print services are located.



The whole idea can be better explained by looking at the *Figure 3.9* provided above.

First, the student and his partner decided on the (java) interface between the two work packages. A **package structure** has to be worked out in which each work-package is a java sub-package. The student had to work out the minimal requirements and dependencies between the two packages. The servlet is the point of interaction between the two packages, as both the student and his partner will be coding part of the servlet.

Next, the student's partner implemented a **print server** based on the existing printing code. The print server can also be asked to list its print options that will go into the dialog box. He also implemented a file service for storing and saving text files. The file service can be asked to list the files it already contained so that the student can browse them remotely. He also wrote the part of the servlet code that finds the services and mediates between them and the student's code.

As for the student himself, he reused his existing editor's interface to allow the user to browse and open existing files, and create, edit, save files. When the user prints a file, the servlet requests the printing options, and builds the corresponding dialog box. The options chosen by the user are passed to the servlet when the user starts the print. The main idea is that the dialog is not hard-coded but is extensible at run-time. The aim is to look at several different ways of doing this, from **HTML forms** generated dynamically in the servlet, to an **XML** representation of the dialog box transformed in to HTML using **XSL** either in the servlet or the client, if time permits.



3.3.3 The Flow Of Events

Note that for the implementation of the design, the Text Editor Applet was used to simulate the web browser application, which is also the client, in this case.

3.3.3.1 The Step-by-Step Explanation

Section 3.3.3.2 contains the flowcharts for both the client and the servlet. Please refer to the flowcharts in the next section, along with the step-by-step explanation.

Step 1: The User Initiation

The user selects a file or creates a new text by editing in the text area of the Text Editor.

The user then presses the Printer Setup button. Please refer to *Figure 3.10a*.

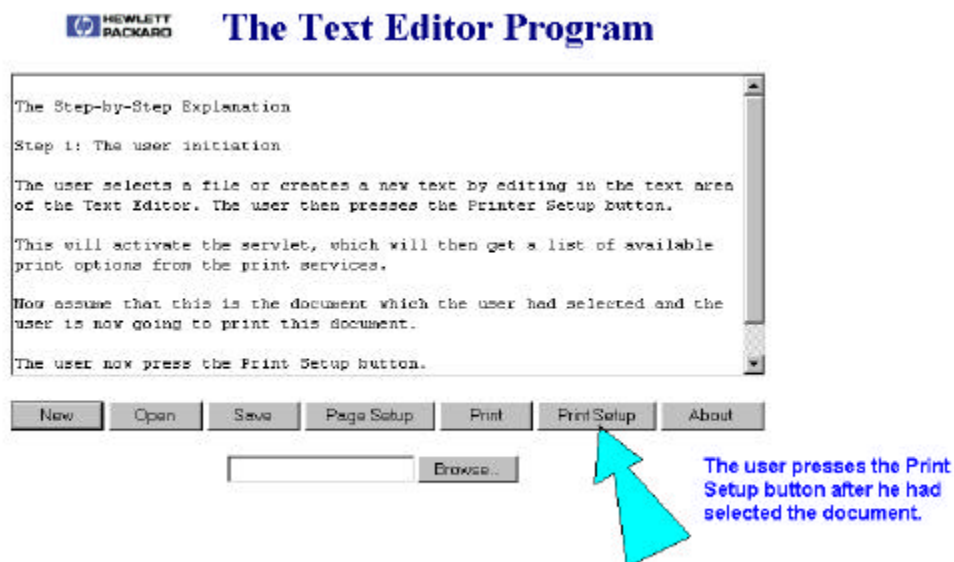


Figure 3.10a The User Initiation



Step 2: Client-Servlet Communication

This will activate the servlet, which will immediately get a list of print options from the print server through the use of an object, which is an instance of a **package structure**.

The structure of this package will be explained in the next section.

Also take note that as shown in *Figure 3.9*, there are actually three ways, which the Editor applet (client) can communicate to the servlet. The first method uses the HTML form to display the interface onto the browser and to pass information. The second method uses the Java operating system code to create the interface, and the last method creates a XML document and uses the XSL transformation language to transform it into a HTML document, which can be display in the web browser.

Step 3: Servlet Execution

Based on the set of print options obtained from the print service, the servlet will now decide what format will be used to display the print option, for example using a select button, a text area, or an option button. The servlet will then display the print options on the browser to allow the user to make their choice.



Step 4: The User Selection

Based on the available print options provided by the program, the user will then proceed to make their choice. When the user selects the Submit button, all the information will be send to the servlet. This is shown in *Figure 3.10b*.



The screenshot shows a web form titled "Welcome to the Editor's Print Setup." in blue text. Below the title is a red instruction: "To print your document, select the following options." The form contains several input fields and radio buttons:

- Name:** A text input field containing "Ivan Low".
- Telephone:** A text input field containing "4488123".
- Color:** Radio buttons for "Black", "Blue", "Green" (selected), and "Red".
- Print On Both Side:** Radio buttons for "None", "Short Side" (selected), and "Long Side".
- Paper Size:** A dropdown menu showing "B5 Envelope".
- Orientation:** Radio buttons for "Portrait" and "Landscape" (selected).
- Notification Option:** Checkboxes for "Post" and "Email" (both checked).

Figure 3.10b The User Selection

Take note that the above presentation of the print options is decided by the package structure, which will be discussed in the later part of the report. However, the representation of the interface is not an important part at this point in time. This is because the focus is on the implementation of the program and its functionality.



Step 5: The Servlet Validation of the Information

The servlet will now check if all the values entered by the user are valid. If any of the data is invalid, the servlet will print the error message accordingly. This is shown in *Figure 3.10c* below.

Welcome to the Editor's Print Setup.

You have entered an invalid argument... Please try again.

Name

Telephone ← This is an invalid telephone number.

(This is an invalid number)

Color

Black

Blue

Green

Red

Print On Both Side

None

Short Side

Long Side

Figure 3.10c The Servlet Validation of the Information

The user, after being prompted by the program, is expected to make the changes. Possible errors that can be detected by the program include:

- Keying in of non-number input into a numeric text box; that is, a text box that only allows numbers.
- No input in the text box, or rather, an empty field.
- Using of space in the beginning of the text field. E.g. Name: <space><space>



Step 6: Finalization of the Information

If there is no error, the servlet will now display a copy of the finalize choices made by the user (refer to *Figure 3.10d*), and at the same time, create an object that contains all the printing information, and send it back to the **Print Service Provider (PSP)**.

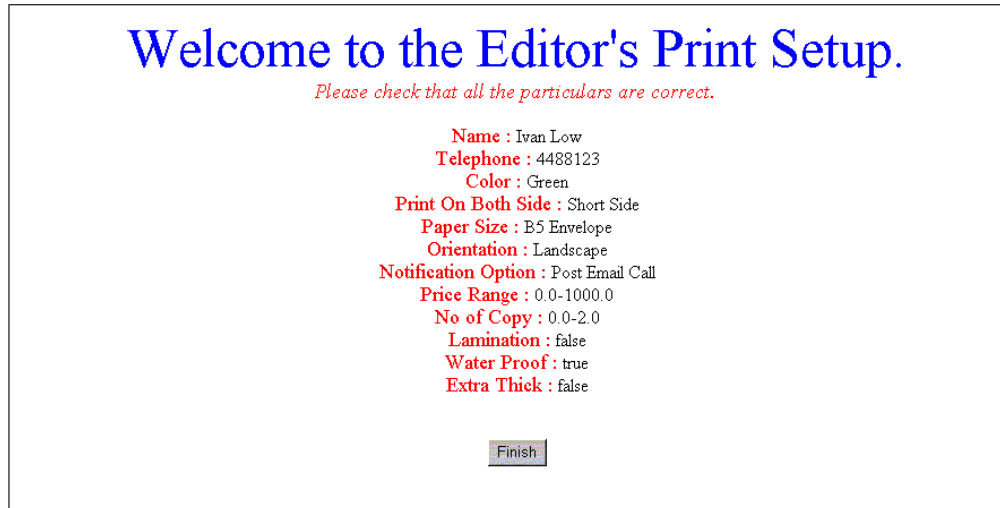


Figure 3.10d Finalization of the Information

When the user selects the Finish button, the user will go back to the Text Editor, and the user has just completed his request for the print service. The following section shows the flow charts for the flow of events for both the **user (client)** and **the servlet**.



3.3.3.2 The Flow Charts

Figure 3.11 shows a flow chart on how the user (client) activates the servlet. Figure 3.12 shows how the servlet reacts to the client and performs the necessary execution as well as it creates the user interface and communicates between the client (web browser) and the print service.

Flow Of Events (User)

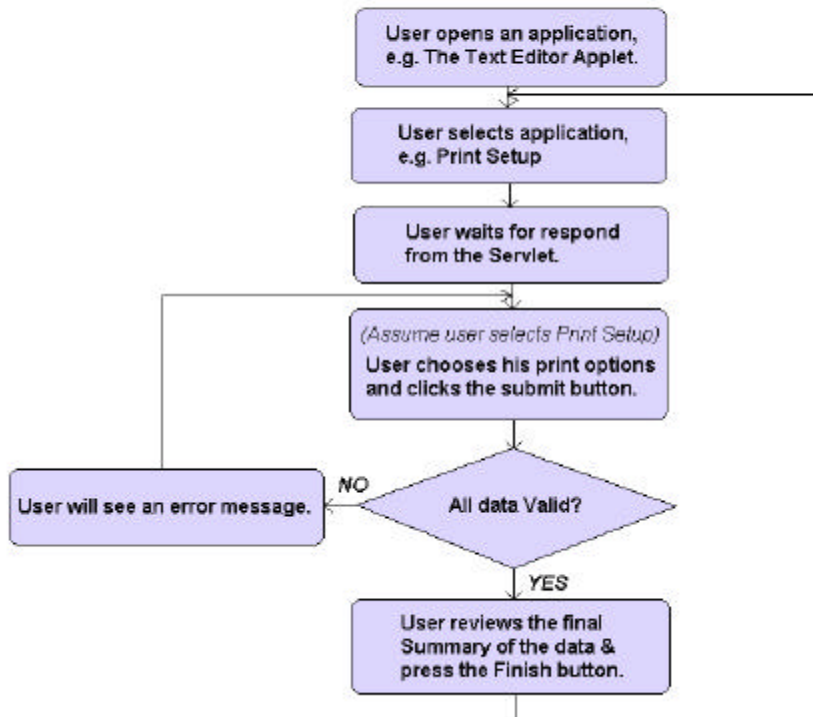


Figure 3.11 The Flow of Events for User



Flow of Events (Servlet)

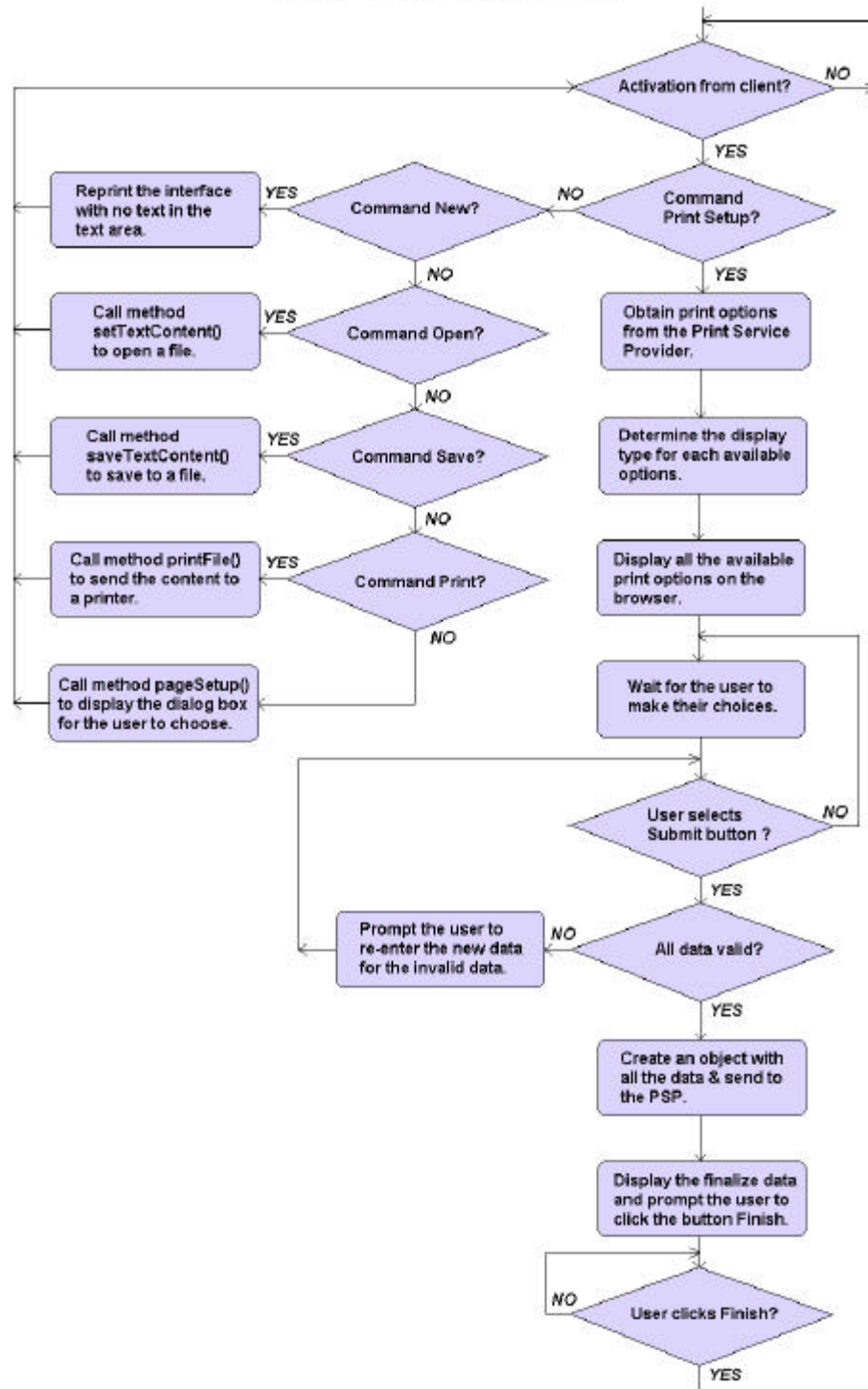


Figure 3.12 The Flow of Events for Servlet



3.3.4 The Package Structure

First, a package structure, shared between the student and his partner, was formed. This package structure was used in the servlet to contain the information regarding the print options. Each of the work-package is a **java sub-package**. The package structure made use of an abstract class **Property**, and follow by four other subclasses, which inherit the methods and attributes of the super class. A model of the package structure shown in *Figure 3.13* was drawn using the **Unified Modeling Language (UML)**.

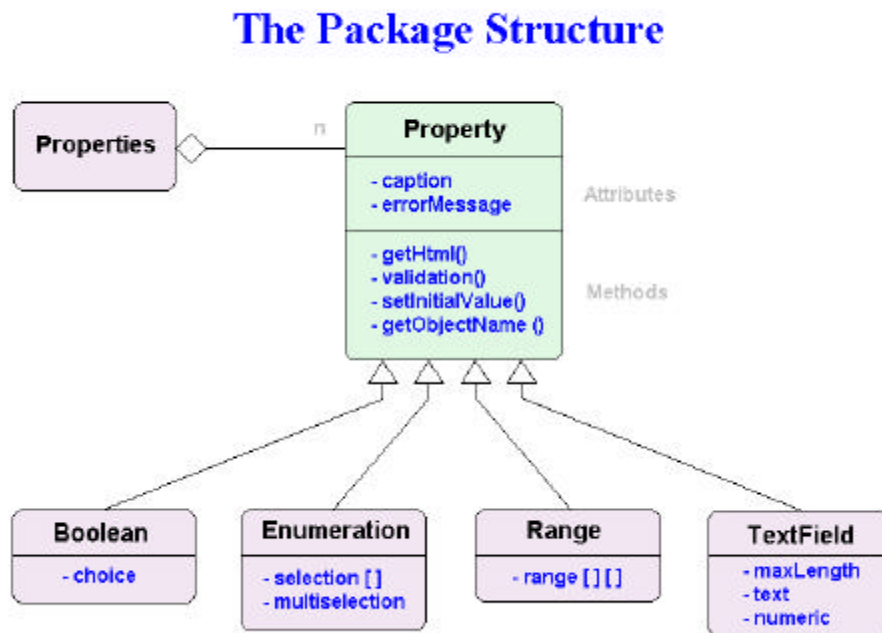


Figure 3.13 The Package Structure



First, an object by the name of properties can be created as an instance of the **Property**.

This is done using the command below:

```
Property [ ] properties = new Property ( );
```

The student can create as many instances of Property as he like. However, an array of Property was created in this case.

3.3.4.1 Implementation of the Class Attributes and Abstract Methods

This Property class will have two main attributes, which will be used by the rest of the subclasses. The first attribute is the **Caption**, which contains the text displayed in the dialog box. The second attribute is the **ErrorMessage**, which ensures that the error message displayed when the user is entering wrong information. Other than these two attributes, there are four methods required by the subclasses (Refer to *Figure 3.14a*) as follows:

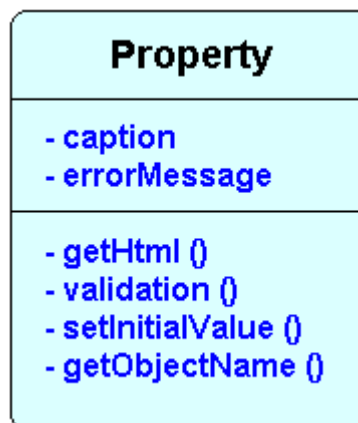


Figure 3.14a Class Property



- *String tohtml ()*

This method determines the type of presentation the object belongs to and creates the HTML form for that object. It returns a string containing the HTML form. For

Example:

```
Properties[0].tohtml()
```

The above sentence will create a string with the html form of the object, and return to the servlet program.

- *boolean validation (String)*

This method accepts a string from the program, checks the value of the string and returns a **TRUE** if the value of the string is valid. Otherwise, it returns **FALSE** .

- *void setInitialValue (String)*

This method accepts a string from the program and sets the string as the initial value. Different classes will set the initial value for the different attributes. For example, the class TextField will set the initial value for the text in the text area.

- *String [] getObjectNames ()*

This method returns an array of the name of the object to the servlet.



3.3.4.2 The Subclasses

The package structure consists of four subclasses that extend the super class. The four subclasses inherit the properties of the super class, and at the same time, define their own attributes. However, because they extend from an abstract super class, they are not allowed to define their own methods. Refer to *Figure 3.14b* to *Figure 3.14e* for the individual graphical class representation.

- *Public class Boolean extends Property*

Attribute: choice (Boolean)

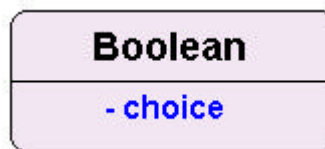


Figure 3.14b Class Boolean

This class has a single attribute, **choice**, which define the initial state of the Boolean object. An object under the Boolean class can only have an initial value of either true or false. An example of a print option under **Boolean** is whether the print document is to be laminated. The choice is only true or false.



- *Public class Enumeration extends Property*

Attributes: selection [] (String)

MultiSelection (Boolean)

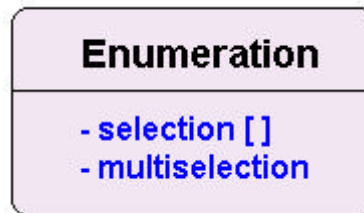
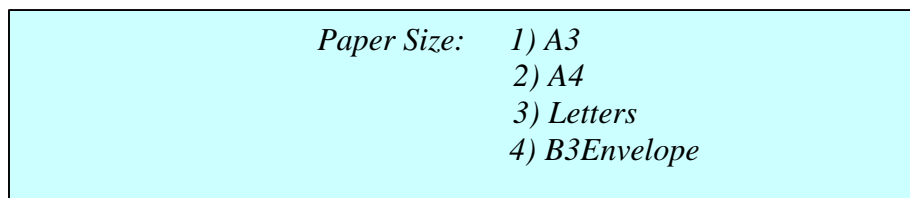


Figure 3.14c Class Enumeration

Attribute **selection** [] contains all the possible choices the user can have. The other attribute, **multiSelection**, defines whether a user can choose more than one item. An example of a print option under Enumeration is shown below:



Note that in certain cases, the user may be allowed more than one option, and in other cases, the user is only allowed to choose one.

- *Public class Range extends Property*

Attribute: `range [][]` (Double)

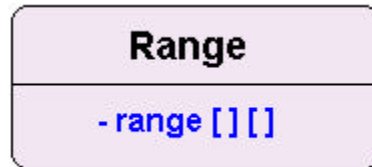


Figure 3.14d Class Range

This is a double array whereby the first level array defines the number of range the object can have, and the second level array consists of the minimum and maximum values of each range.

For example:

```

Range [0][0] = 10;   Range [0][1] = 20;
Range [1][0] = 20;   Range [1][1] = 30;
  
```

The attribute in this example consists of two ranges;

The first one is from **10 to 20**, while the second one is from **20 to 30**. An example of when a range should be used for print option is shown below:

```

Cost Budget:  1) $100 to $500
              2) $501 to $800
              3) $801 to $1000
              4) $1001 to $5000
  
```



- **Public class TextField extends Property**

Attributes: *maxLength (integer)*

text (String)

numeric (Boolean)

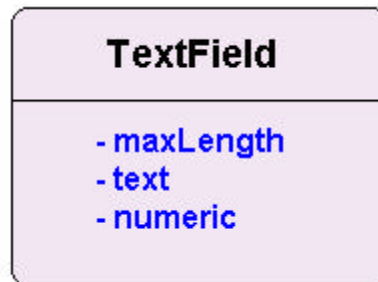


Figure 3.14e Class TextField

This class consists of three different attributes. The first attribute, **maxLength**, defines the maximum number of characters that can enter into the text field. The second attribute, **text**, is the initial text that will appear in the text field. The last attribute, **numeric**, tells the program whether the text field should allow only numbers, or both numbers and characters.



3.3.5 The New Package Structure

This new package structure was required because the servlet had to send the finalized version of the choices made by the user to the **Print Service Provide (PSP)**. This was a much simpler structure, and does not require the use of an abstract class. *Figure 3.15* below shows the simple layout for the new package structure.

The New Package Structure

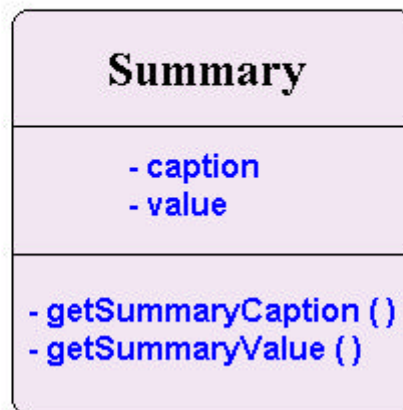


Figure 3.15 The New Package Structure

The public class, **Summary**, consists of only two attributes and two methods. It does not have any subclass that extends it. The first attribute, **caption**, contains the name of the object that appears on the interface, while the other attribute, **value**, contains the value of the object.



The two methods, **getSummaryCaption ()** and **getSummaryValue ()**, enable the servlet to get the caption and value of the object respectively.

With this simple package structure, the servlet can easily pass the information to the **Print Service Provider (PSP)** involved.

3.3.6 Implementation of The Servlet Code

After the two packages structure have been created, the servlet can easily use the class attributes and abstract methods defined in the package structure to perform the required functions. As mentioned before, the student and his partner wrote the servlet code. His partner created an object, and then used the constructor for the abstract class **Property** (refer to the section on the Package Structure) to initialize the initial values for all the print options. The object can either be a **TextField**, **Boolean**, **Range** or **Enumeration**. An example of a program, which created 4 different objects, is shown below.

```
Property [] properties = new Property ();  
Properties [0] = new TextField ("Name", 30, "Ivan Low ", false);  
Properties [1] = new Enumeration ("Color", string0, false);  
Properties [2] = new Range ("Price Range", newRange1);  
Properties [3] = new Boolean ("Lamination", false);
```



After the student's partner initialized the object properties with all the available print options from the **Print Service Provider (PSP)**, the student then proceed on to determine the type of structure to use for displaying the print options. This is being done in the package structure itself. All the servlet has to do is to call the method **tohtml()**, which is an abstract method defined in the abstract class **Property**. This returns a string, which contains all the HTML form of the interface of the print option, and the servlet can just send this string to the browser, which will then display the interface.

The servlet will then wait for the user to activate itself again by clicking the Submit button. Based on the data given by the user, the servlet then call the method **validation()** to validate the data. When all the data are validated, the program then use the method **setInitialValue()** to set the initial value of each object, and display the final data provided by the user on the browser. The servlet, at the same time, creates an object of the class **Summary** (refer to the new package structure) and sends it to the **Print Service Provider (PSP)**.



3.3.7 Problems and Difficulties Encountered

The main difficulty the student encountered was the designing stage of the project. The student and his partner had to design a package structure, which would be used by both of them on different machines. The package consisted of an abstract super class with four subclasses that implemented the abstract methods defined in the superclass. The main problem was that the servlet had to convert the object defined as either one of the subclasses, into an Html format. However, the four subclasses accepted different format of arguments, it can be a string, string array or even Boolean, float and double. The problem was how to implement the abstract method if all the subclasses used different argument formats. This problem was resolved by implementing a new abstract method **toHtml()** within the package structure, to convert the object to Html format within the package structure, and not in the servlet code. The servlet just have to call the abstract method to get the Html format. This was a good advice given by the student's supervisor, Mr. Tom Gardner, who again introduced the concept of Object Oriented Programming to the student.

Other problems encountered by the student included the validation of data, creating a new object in the new Package Structure described earlier on, and the presentation of the interface. However, all these problems are resolved under the efforts of the student and with the help of the supervisor.



3.3.8 Conclusion and Recommendation

At this point of time, the student had already completed the main part of the project, with knowledge of the following:

- Java
- HTML
- Client/Server Computing (servlet)

The design and implementation of the two package structures allowed the student to once again, understand and experience the advantages of using Object Oriented Programming. The student began by writing the package structure using object, classes and other Object-Oriented concepts. But the implementation of the program still consists of both the knowledge of Function Oriented and Object Oriented. The student later discovered the difficulties of not implementing the whole structure using the Object-Oriented concept learnt earlier on, and thus went on to rebuild the package structure. This was in effect, learning from mistakes, without which the student would never improve.

The project however, may be further improved with the use of the **Extensible Markup Language (XML)** and the transformational language **XSL** to transform it into a HTML document. This will be discussed in the later part of the chapter.



3.4 Introduction to the Extensible Markup Language (XML)

3.4.1 What is XML?

XML is a subset of the **Standard Generalized Markup Language (SGML)** defined in ISO standard 8879:1986 that was designed to make it convenient to interchange structured documents over the Internet. XML files always clearly mark the start and end of each of the logical parts (called elements) of an interchanged document occurring. XML restricts the use of SGML constructs to ensure that fallback options are available when access to certain components of the document is not currently possible over the Internet. It also defines how Internet Uniform Resource Locators can be used to identify component parts of XML data Streams.

Reference on XML: <http://www.oasis-open.org/cover/xml.html>



3.4.2 What's Wrong with HTML

Originally, the intention of using HTML was that the elements should be used to mark up information according to their meaning, without regards as to how this would actually be rendered in a browser. In other words, the title, main header, emphasized text and the contact information of the author should be placed inside the elements TITLE, H1, EM (or possibly STRONG) and ADDRESS. To use FONT or I and similar elements to get a nice layout makes it a lot more difficult to present the information to the best possible effect regardless of the user's environment. Processing the information consequently becomes difficult (or even impossible).

In addition, this is not the only problem. If one wants to mark up his information very precisely according to its meaning, one will need a lot of elements that are not available in HTML. Catering to the needs of people from all trades will obviously mean requiring an enormous amount of elements, which is a horror for both developers and users.

Another problem is that HTML has limited internal structure, which means that one can easily write valid HTML that does not make sense at all when he considers the semantics of the elements. This is because (among other things) the contents of BODY have been defined so that one can place the elements allowed there in any order as he pleased. This suggests that one does not need a H1 with the H2s inside it and H3s inside the H2s. (Think of H1 as a book title, H2 as part title and H3 as chapter title.) HTML should ideally be written this way, but the HTML standard does not require it.



3.4.3 Replacing HTML with XML

The plan now is to use the **eXtensible Markup Language, XML** to represent the information instead of using HTML. In the package structure defined earlier on, there is an abstract method **toHTML ()**, which is implemented by all the subclasses. The idea now is to define a new abstract method, **toXML ()**, which will change the object into its respective XML form (instead of HTML form). **XSL**, a transformational language, will then be used to transform the XML document into an HTML document, either in the servlet or in the client.

One might probably ask: Why is there a need to use XML since HTML can do the job too? The HTML format only allowed the document to be displayed on the browser. It has limited elements that can be used for other purposes. The XML, on the other hand, allowed more structure to be introduced onto the document. It can be used as a document to pass information to other programs. HTML is good when the server needs to send information to be displayed on the web browser. However, when the receiver is not a web browser but some other programs, e.g. service or machine, it may not recognize HTML. Thus, XML is used as a common language that can be sent to anybody who is at the receiving end. If the data needs to be displayed on the web browser, the XML document can be conveniently converted into a HTML document. The Internet Explorer 5, for example, acts as a XSL converter processor.



3.4.4 XSL: Converting XML to HTML

XSL is a transformational language that transforms a document written in one language (XML) into a document of another language (e.g. HTML). The transformation requires a XML document and a XSL document, which consists of a template that defines the behavior of the elements in the XML document. To create a HTML document, all the program has to do is to send the XML document to a browser, for example the **Internet Explorer 5**, and indicate where the XSL document is. A HTML document will then be created based on the XML document, and the predefined XSL document. *Figure 3.16* illustrates this process.

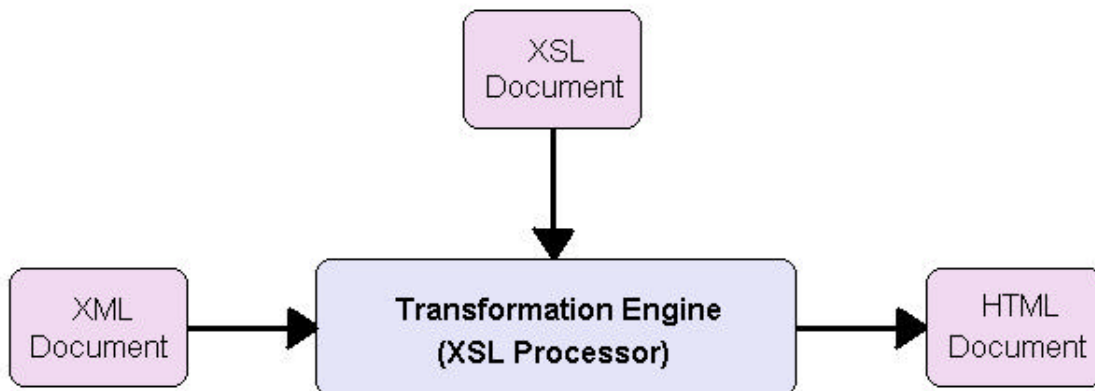


Figure 3.16 The XSL processor

Figure 3.17 and 3.18 below show a sample of the document written in XML and XSL respectively.

```

E:\graphics\xmlsample
<?xml version = "1.0"?>
<!DOCTYPE PrintOption SYSTEM "file://localhost/PrintOption.dtd">
<PrintOption>
  <Option>
    <Caption>Name</Caption>
    <Type>TextField</Type>
    <Numeric>False</Numeric>
    <Default>Ivan Low</Default>
  </Option>
  <Option>
    <Caption>Orientation</Caption>
    <Type>Enumeration</Type>
    <Selection>Portrait,Landscape</Selection>
    <Default>Portrait</Default>
  </Option>
  <Option>
    <Caption>Lamination</Caption>
    <Type>Boolean</Type>
    <Default>True</Default>
  </Option>
</PrintOption>
    
```

Figure 3.17 A Sample of XML Document

```

E:\graphics\xmlsample
<?xml version = "1.0"?>
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version = "1.0">
  <xsl:template match = "/">
    <HTML>
    <HEAD>
    <TITLE>Print Option</TITLE>
    </HEAD>
    <BODY>
      <xsl:apply-templates/>
    </BODY>
    </HTML>
  </xsl:template>
  <xsl:template match = "PrintOption">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match = "Caption">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match = "Type">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match = "Default">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match = "Numeric">
    <xsl:apply-templates/>
  </xsl:template>
</xsl:stylesheet>
    
```

Figure 3.18 A Sample of XSL Document



3.4.5 The Implementation of XML

The student, unfortunately, is still at the design stage of this part of the project. The idea of using XML to replace the HTML has not been implemented at the time of submission of this report. It should, however, be implemented before the end of the attachment. This would then fulfill the objective of the project, meeting the requirements of the attachment.

3.4.6 Problems Encountered and Conclusion

The most serious problem that the student faced was time constraint. The student was expected to complete the whole project before the end of the attachment. However, due to the early submission of the report, the implementation of the XML and the results observed thereafter by the student would not be recorded in this report. Nevertheless, the student had indeed successfully implemented the last part of the project and benefited from this enriching attachment.



Chapter **Four**

The Attachment Conclusion

The overseas industrial attachment, which lasted for nearly five months, had come to an end. The program had greatly enhanced the student's knowledge in programming and improved his computing skills.

The following sections discuss the conclusion of the attachment, as well as the experiences and benefits gained by the student.



Conclusion

The E-service problem set, a step-by-step approach used by the student, has ensured that the student acquired enough knowledge required in various aspects for all the stages, before proceeding on. During the 5 months attachment in Hewlett-Packard Lab, the student had done the following:

- Designed a business model (represented in XML) that enabled information in the form of XML documents to be shared between businesses.
- Implemented a GUI that allowed a user to view and modify the information in an XML document. XSL was used to transform XML into HTML for display in a browser.
- Wrote a Java Servlet code to handle the file access application required by the web client. These are application that cannot be handled by a Java Applet.
- Implemented the Text Editor Interface (client) using HTML, a straightforward language used to display certain interface program onto a web browser.
- Designed a class diagram using the Unified Modeling Language (UML) that defines the classes and their behavior.
- Created a Text Editor Interface Application using the Java Applet. This allows greater flexibility than HTML in terms of the layout and functionality.



Having said that, it was nevertheless difficult to master the Object Oriented paradigm within a time frame of 5 months and use the new language to implement an application that was very different from the usual single-user program. However, due to the determination of the student, and under the guidance of the supervisor, what seemed initially daunting soon became manageable. The student had not only picked up the skill of writing program using the object oriented concept, but also the knowledge to write a servlet program to handle the events initiated by the client applet application. The student is now capable of using the Unified Modeling Language (UML) to represent the class diagrams that define the classes and their structure and behavior, dependency, association, and inheritance relationships.

The student is also able to understand the concept of e-service, and how the e-services can interact to the users using e-speak. By the end of the attachment, the student will be able to use the eXtensive Markup Language (XML) to replace HTML, in representing the information that needed to be sent to any user, programs, or company. The eXtensible Stylesheet Language (XSL) will then be used to determine the display and layout of the XML documents, which will be displayed on the web-browser.

Throughout the 'Baskerville Project' development, the student had learned valuable lessons regarding Windows application programming. Having the knowledge of Java, HTML, XML, XSL, and UML proved to be immensely useful. This had also built up the student's confidence in dealing with feature-rich, user-friendly Windows applications.



Other Experiences Gained

The student would like to thank his school for the chance to work with the renowned Hewlett-Packard Lab in Bristol. The student's involvement in the project and department meetings has indeed allowed him to gain valuable insight. The student was introduced to the latest technologies used in the Laboratory especially. The development of the 'Baskerville Project' was a totally new concept, making use of different scenarios, actors, use-cases and domain models that the student had never been exposed to before.

In many more ways than can be listed here, the IA program has enabled the student to understand the wonders/usefulness of computer engineering, its nature as an academic endeavour and various applications in reality.



Chapter Five

Bibliography and References

5.1 Java and Object Oriented Programming

- Java in a Nutshell, a Desktop Quick Reference by David Flanagan
- Java Examples in a Nutshell, a Tutorial Companion to Java in a Nutshell, by David Flanagan
- Thinking in Java
<http://bruceeckel.com>
- Sun's Java Tutorial
<http://java.sun.com>
<http://java.sun.com/docs/books/tutorial>
- Sun's Java API Index
<http://java.sun.com/products/jdk/1.2/docs/api/index.html>
- Sun's Java Web Server API Index
http://jserv.java.sun.com/products/java-server/documentation/webserver1.1/servlets/servlet_tutorial.html
- O'REILLY's Java Servlet Programming
<http://www.oreilly.com/catalog/jservlet/chapter/ch03.html>



5.2 HTML, XML and XSL

- The XML Cover Page
<http://www.oasis-open.org/cover/xml.html>
- A Technical Introduced to XML
<http://nwalsh.com/docs/articles/xml/>
- XML Tutorial
<http://msdn.microsoft.com/xml/tutorial/default.asp>
- Using the XSL Processor
<http://msdn.microsoft.com/xml/xslguide/transform-overview.asp>
- Introduction to Web Design
<http://wdvl.internet.com/Authoring/HTML/Tutorial/toc.html>

5.3 E-service and E-speak

- E-speak
<http://www.e-speak.net>
- E-speak Programmers guide and API Index
<http://www.e-speak.net/library/pdfs/Jesi-PgmGuide.pdf>

5.4 Unified Modeling Language

- Unified Modeling Language Resources Center
<http://www.rational.com/uml/index.jtmpl>

