NANYANG TECHNOLOGICAL UNIVERSITY

School of Electrical and Electronic Engineering

Report on Overseas Industrial Attachment With Hewlett-Packard Laboratories Bristol

Prepared by : LEE KENG HOCK
982657H03
May 2000

Overview

What's Inside

Abstract

This report represents the experience gained by the writer throughout the Overseas Industrial Attachment in Hewlett-Packard Laboratories Bristol, England. It describes the steps taken to complete the projects assigned, which includes :

- Object Oriented Programming (OOP) Using Java
- Distributed Computing with Applets & Servlets
- E-Services

Skill in OOP was gained through implementation of a text editing application using the java language. The capabilities includes saving and loading a document to and from file respectively. It also includes printing the document.

As a step towards distributed computing, the text editing application was converted into a client/server application. It involves using HTML and applets for the front-end and servlets for the back-end.

The last phase involves the use of e-speak to develop a system that will serve as an intelligent interaction of e-services. A step towards future e-commerce.

For further details, please refer to Steve Battle (HPLB).

Acknowledgements

The writer would like to thank Hewlett-Packard Laboratories Bristol for the opportunity of working in the company. The writer would like to express his sincere appreciation to Steve Battle, the HP Lab supervisor-in-charge, for his trust and guidance in all the projects assigned and his willingness to impart his valuable knowledge, experience and skill to the writer.

The writer would like to extend his gratitude to Anthony Wiley, the project manager and the other members of the project group for their assistance and support in various ways towards the successful completion of various assignments.

Next, to the writer's supervisor-in-charge in NTU, A.P. Lin ZhiPing from the School of Electrical & Electronic Engineering for his time spent in coordinating between the writer and the school.

Last but not least, special credits must be given to the writer's family and girlfriend Dian for their support throughout the entire period of Attachment.

List of Figures

List of Tables

Chapter One - Introduction

1.1 Aim

The purpose of this report is to present and discuss the works done on the following projects by the writer during his NTU Overseas Industrial Attachment period, from 15th February 2000 to 1st July 2000, at Hewlett-Packard Laboratories Bristol : -

- **Text Editing Application**
- **Text Editing Applet**
- **Baskerville - Print E-Services**

The text editing application and applet exercises given by the writer's supervisor was meant for the writer to develop core competencies required to build new electronic services.

However, the required date of submission inevitably limits the accuracy and completeness of this report. Further work that is scheduled for execution after the date of reporting cannot be described fully at this point. That work will be completed before the end of the attachment.

Furthermore, the delayed commencement of the attachment had reduced the amount of training gained and work done that are reportable.

A difficulty faced when writing this report was the level of detail and assumptions that need to be made on the reader's level of familiarity with concepts. In general, the report will not discuss at length the implementation details of application development. As stated, the emphasis will be on the experience gained and a summary of the work done. The reader is assumed to know some computing concepts like object-oriented programming and electronic commerce because the project required extensive use of these paradigms for implementation, and many references will be made to these concepts. However, a simple overview of the various concepts will be introduced at the beginning of each chapter.

1.2 Scope

In **Chapter 2**, the company, Hewlett-Packard's background is presented. This is followed by an introduction of Hewlett-Packard Laboratories, the research goals, University partnerships and the definition of the HP WAY.

Chapter 3 begins with an introduction to Object Oriented Programming using Java. Next, the **T*ext Editing Application*** is discussed. The objective of the exercise is then highlighted, followed by the implementation. The last section covers the problems encountered, some tips on organising of files and the difference between the Java AWT and Java Swing.

Chapter 4 begins with an introduction to Applets and Servlets. Next, the ***Text Editing Applet*** is discussed. The objective of the exercise is then highlighted, followed by the implementation. The next section covers the problems encountered and some tips on refreshing of re-compiled programs in the web browser. The last section covers the use of JavaDoc and Unified Modelling Language for documentation purposes.

Chapter 5 begins with an introduction to E-speak, an open software platform designed specifically for the development, deployment and intelligent interaction of e-services. Next, an overview of the ***Baskerville Project*** is discussed. The objective of the project is then highlighted, followed by the implementation. The last section covers the problems encountered, some tips on using JAR files and setting of environments.

Chapter 6 covers the overall conclusion of the Overseas Industrial Attachment.

Chapter Two - The Company, Hewlett-Packard

2.1 Company's Background

Figure 2.1 : Bill Hewlett and David Packard helped craft the first set of corporate objectives in 1957.

HP was founded in 1939 by Bill Hewlett and Dave Packard. They were two Stanford engineers who combined their product ideas and unique management style to formed a working partnership. The company's first product, built in a Palo Alto garage, was an audio oscillator – an electronic test instrument used by sound engineers. Today, the 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). The company's more than 36,000 products are used by people for personal use and in industry, business, engineering, science, medicine and education. In addition, the company make networking products, handheld calculators and printers.

HP employs more than 120,000 people, of whom some 67,500 work in the United States. HP product development and manufacturing sites are located in 28 U.S. cities and in Australia, Brazil, Canada, China, France, Germany, India, Ireland, Italy, Japan, Korea, Malaysia Mexico, the Netherlands, Spain, Singapore and the UK.

2.2 HP Laboratories

Figure 2.2 : HP Lab Bristol

HP Laboratories (HPL), directed by Dick Lampman is the company's central research facility, ranks as one of the leading industrial-research centres in the world. At its headquarters in Palo Alto, as well as in Laboratories in the UK, Japan and Israel, researchers develop and apply leading-edge technologies that support HP's current businesses and create new opportunities for the company, In 1998, between HP Labs and some 70 product divisions, HP invested $3.4 billion in R&D.

HP Lab Bristol (HPLB) is HP's largest research centre outside its headquarters in Palo Alto, California, and employs around 284 researchers – one third of HPL's total. HPLB is a multinational research community with a network of relationship spanning HP, its customers and academia across the globe. It shares a site with one of HP's manufacturing division on the hi-tech edge of Bristol, with fast road, rail and air links to the rest of the UK and the world beyond.

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. The role of Bristol is mainly to think about how people want to live and work in the next decade.

This assignments were carried out in the Digital Fulfilment Department, which is part of the Publishing Systems and Solutions Lab, under Robin Gallimore.

2.4 University Partnerships

HPLB has an extensive network of relationships with selected departments in leading academic institutions in over 20 countries worldwide, 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 skill 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.

For more information of Hewlett-Packard, please refer to Http://www.hp.com

2.5 HP WAY

"What is the HP Way?

I feel that in general terms it is the policies and actions that flow from the belief that men and women want to do a good job, a creative job, and that if they are provided the proper environment they will do so."

Bill Hewlett
HP Co-Founder

Chapter Three – Object Oriented Programming Using Java

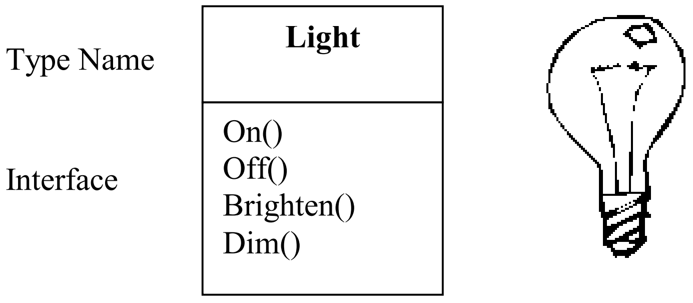
3.1 Object Oriented Programming (OOP) Overview

Since its introduction in late 1995, the Java language and platform have taken the programming world by storm. 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 overly complex features that have bogged down other object-oriented languages, such as C++. By keeping the language simple, the designers also make it easier for programmers to write robust, bug-free code.

Below is a summary of the five basic characteristics of Smalltalk, the first successful object-oriented language and one of the languages upon which Java is based.

a. Everything is an object.
b. A program is a bunch of objects telling each other what to do by sending messages.
c. Each object has its own memory made up of other objects.
d. Every object has a type.
e. All objects of a particular type can receive the same messages.

A simple example might be a representation of a light bulb :



Light lt = new Light();

lt.on();

Here, the name of the type/class is `Light`, the name of this particular object is `lt`, and the requests that you can make of a `Light` object are to turn it on, turn it off, make it brighter or make it dimmer. You create a `Light` object by defining a "handle" (`lt`) for that object and calling `new` to request a new object of that type. To send a message to the object, you state the name of the object and connect it to the message request with a period (dot). From the standpoint of the user of a pre-defined class, that's pretty much all there is to programming with objects.

4 basic steps needed to create a GUI application

a. **Create the components**. A GUI component is created just like any other objects in Java – simply call the constructor. For example, to create a `Button` component that displays the label "Quit", you simply say :

Button quit = new Button("Quit");

b. **Add the components to a container**. All components must be placed within a container. To add a component to a container, you simply pass the component to the `add()` method of the container. For example, you might add a `quit` button to an applet with code like the following

this.add(quit);

c. **Arrange, or layout the components with their containers**. In other words, you need to set the position and size of every component so that the GUI has a leasing appearance. While it is possible to hardcode the position and size of each component, it is more common to use a `LayoutManager` object to automatically layout the components of a container according to certain layout rules defined by the particular `LayoutManager` you have chosen.

d. **Handle the events generated by the components**. There are low-level user input events, such as keyboard and mouse events generated by the operating system and the higher-level semantic events that are generated by the components themselves, in response to the lower-level input events.

For further reading on Java and OOP, please refer to http://bruceecke.com.

3.2 Objectives – Text Editing Application

This exercise provides an introduction to OOP using Java. The objective is to build a simple text editing application. This editor will be able to save and load a document to and from file respectively. In addition, it should be able to print the document.

Requirements

a. Java Development Kit (JDK)

b. Project Organisation

c. Use of AWT and IO packages

3.3 Specifications

The top-level flow diagram of the text editing application is as shown *figure 3.1* below. It specifies the minimal required functions needed. With OOP and modular programming, it enables easy addition of new functions. Explanations of the various functional blocks will be done in the next section.

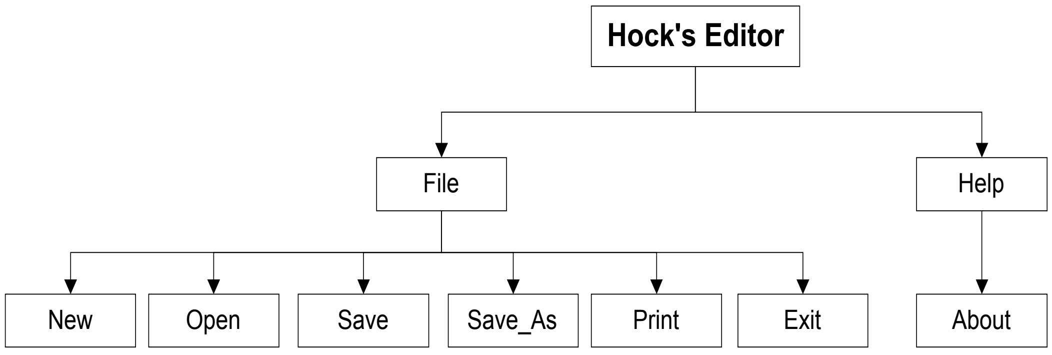
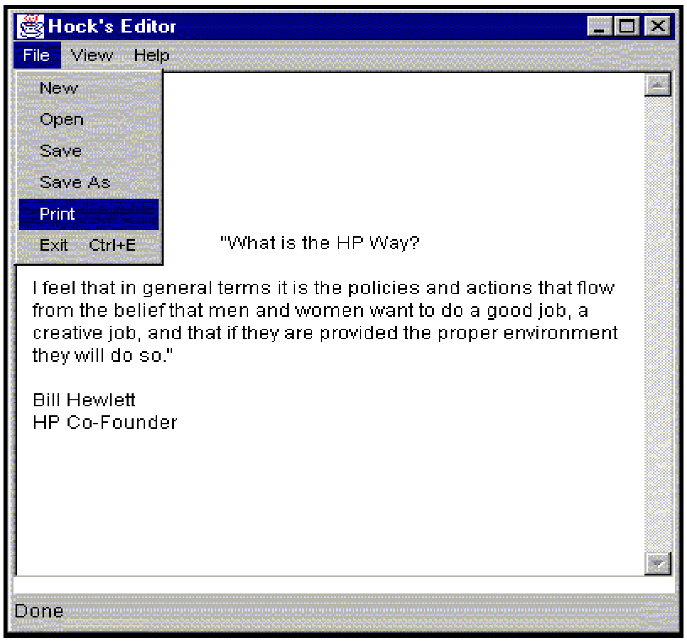


Figure 3.1 : Flowchart of Text Editing Application

The basic GUI is shown in *figure 3.2* below. It consists of a menu bar object, a text area object and a label object used as a status bar. All the components of the GUI were built using the Abstract Windowing Toolkit (AWT), which defines all of the GUI components in Java.



"What is the HP Way?

I feel that in general terms it is the policies and actions that flow from the belief that men and women want to do a good job, a creative job, and that if they are provided the proper environment they will do so."

Bill Hewlett
HP Co-Founder

Figure 3.2 : GUI of Text Editing Application

3.4 Implementation

The Java 1.1 event-handling model is based on the concept of delegation. An object interested in receiving events is an event 'listener'. An object that generates events (an event source) maintains a list of listeners that are interested in being notified when events occur, and provides methods that allow listeners to add themselves and remove themselves from this list of interested objects. When the event source object generates an event (or when a user input event occurs on the event source object), the event source notifies all of the listener objects that the event has occurred.

3.4.1 File-New

Upon selecting File-New, the text area will be cleared.

3.4.2 File-Open

Upon selecting File-Open, a file dialog will appear requesting the user for a file to open. If the file is readable, the text area is cleared, the file is then read and display.

The flow chart is shown in *figure 3.3* below.

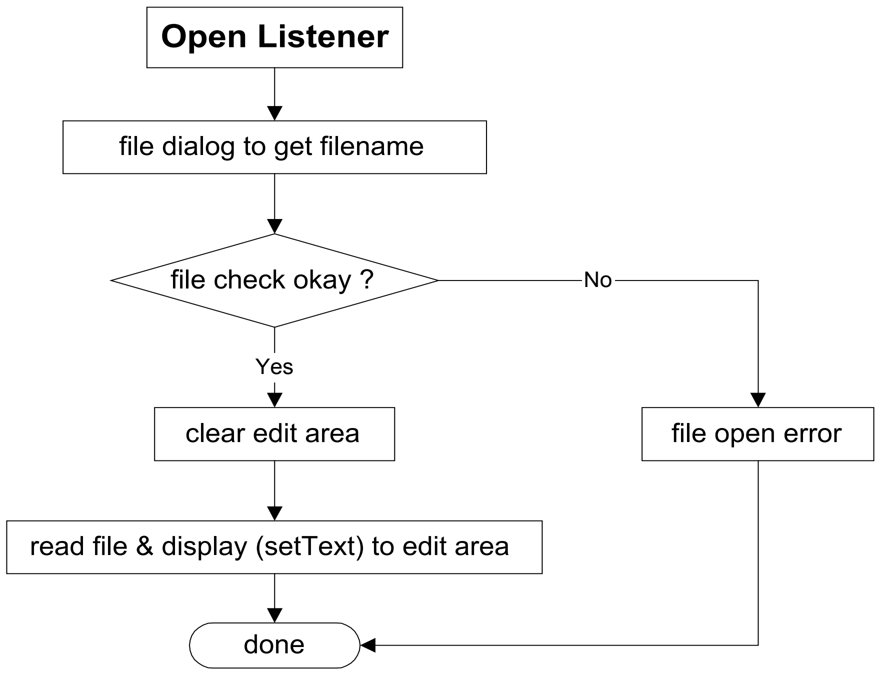

Open Listener
file dialog to get filename
file check okay ?
No
Yes
clear edit area
file open error
read file & display (setText) to edit area
done

Figure 3.3 : Flowchart of File-Open

3.4.3 File-Save_As

File-Save_As is programmed before File-Save. Reason being that File-Save can be a subset (or a method) of File-Save_As.

Similarly, a file dialog will appear requesting the user for a file name. If the directory is writable, content of the text area is written to the specified file.

The flow chart is shown in *figure 3.4* below.

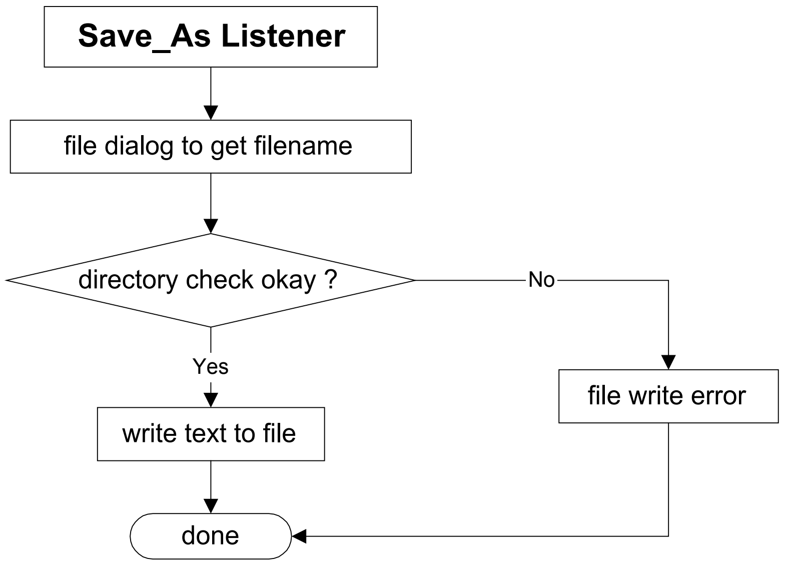


Figure 3.4 : Flowchart of File-Save_As

3.4.4 File-Save

A check will be done to detect any text change and whether there is a file loaded. If there is any text changed in a loaded file, the file will be saved to the loaded filename by calling a save method in File-Save_As module. If it is a newly edited file without a filename, the entire File-Save_As module will be invoked.

This is shown in *figure 3.5* below.

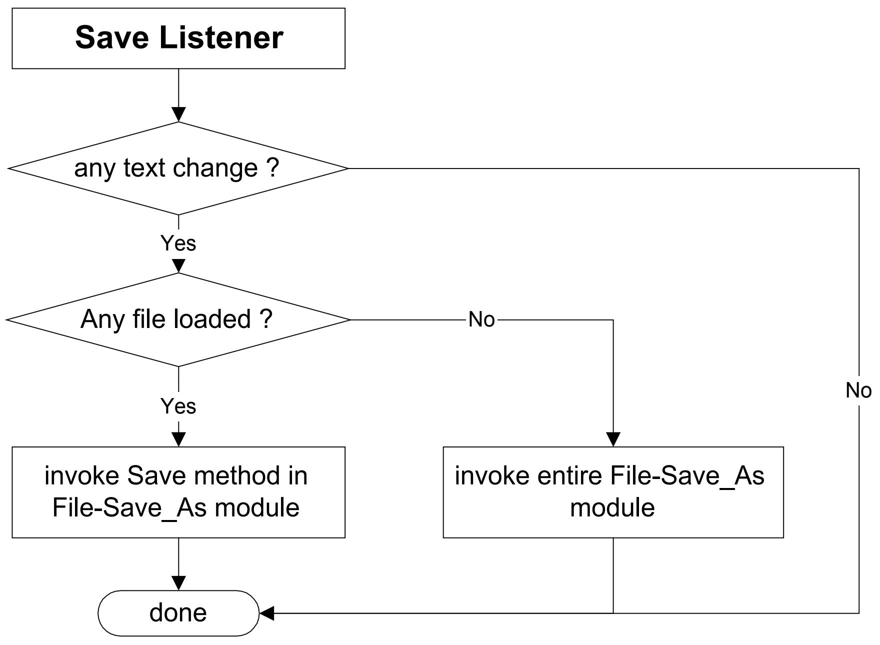

Save Listener
any text change ?
Yes
Any file loaded ?
No
No
Yes
invoke Save method in File-Save_As module
invoke entire File-Save_As module
done

Figure 3.5 : Flowchart of File-Save

3.4.5 File-Print

Method used was to get the page format and the print format, calculate the total number of pages required and print by character. The reason for not printing by string or line format is to have more freedom of manipulating the print. The main drawback will be the prolonged printing time.

The flow chart is shown in *figure 3.6* below.

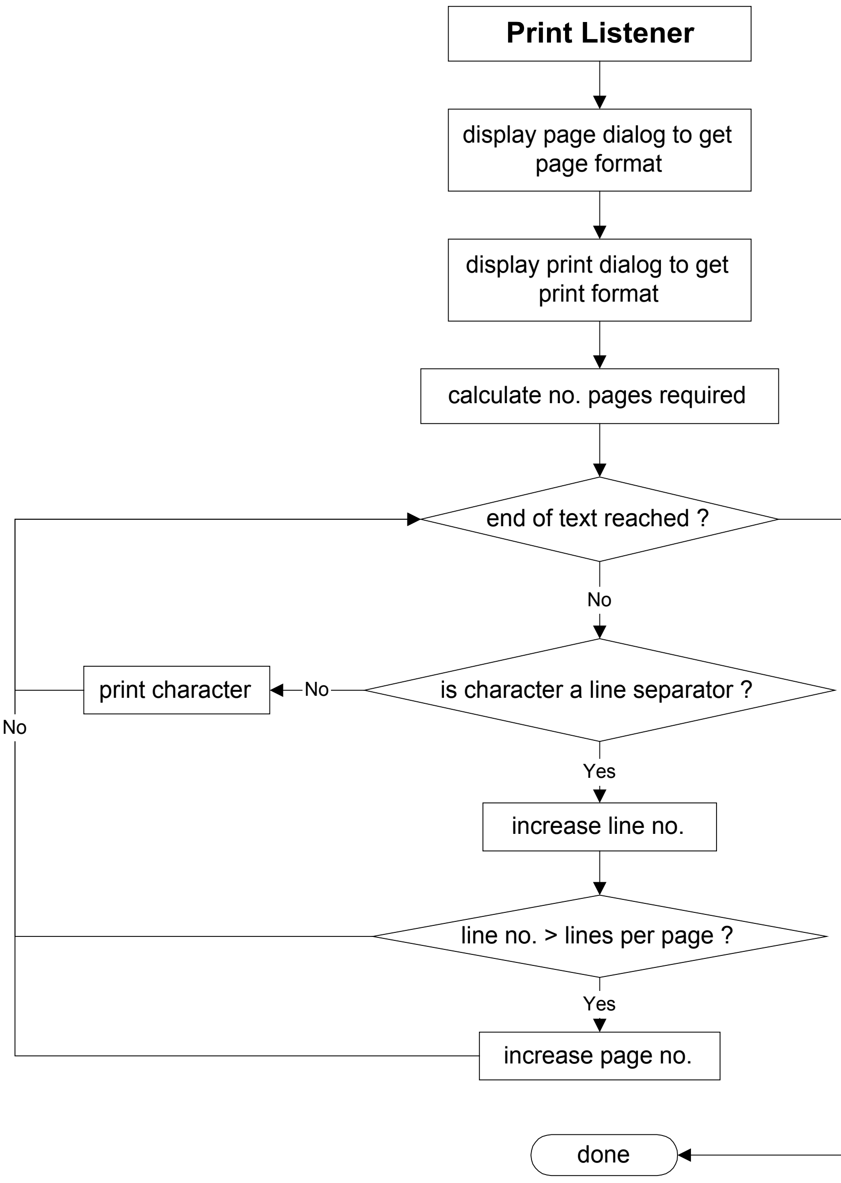

Print Listener
display page dialog to get page format
display print dialog to get print format
calculate no. pages required
end of text reached ?
No
is character a line separator ?
No
print character
No
Yes
increase line no.
line no. > lines per page ?
Yes
increase page no.
Yes
done

Figure 3.6 : Flowchart of File-Print

Additional features added to the print option include printing multiple copies and proper termination when printing near the right margin. The algorithm uses a buffer to store the characters and print only when a space is encountered. The algorithm used and the code written for proper termination when printing near the right margin was not very efficient and modular as there wasn't enough brainstorming and planning done before writing this module.

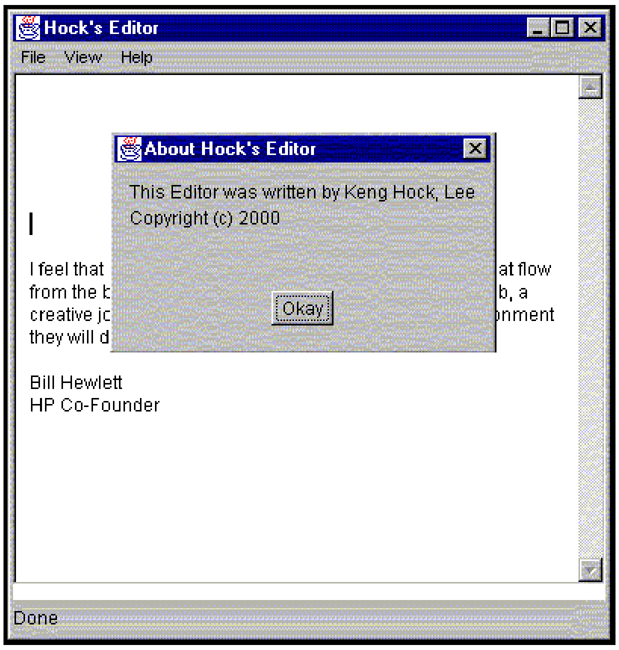
3.4.6 File-Exit

This option terminates the Program.

3.4.7 Help-About

This option was created to allow the writer to experiment with the functionality of a pop-up dialog. Upon selecting this command, an info dialog that contains a multi-line label will be shown. The multi-line label was created using an array of label objects because Java doesn't support multi-line labels. The MultiLineLabel class will calculate and create the necessary number of lines & label objects required.

The GUI is as shown in *figure 3.7* below.



This Editor was written by Keng Hock, Lee
Copyright (c) 2000

I feel that ... at flow
from the b... b, a
creative jc... nment
they will d...

Bill Hewlett
HP Co-Founder

Figure 3.7 : GUI of Help-About

3.5 Problems Encountered

3.5.1 Using OOP

In the past, the writer was mainly involved in function oriented programming using C and Visual Basic. He has not been involved in OOP, and Java is the very first OOP language being learned. Conceptually, Java is very different from C or Visual Basic that the writer had learned. The writer started out with a lot of doubts and uncertainties. A lot of time was spent in practically experimenting with the language and in writing small programs to aid understandings. More time will be needed to appreciate the concept of OOP and to use it effectively.

However, the supervisor had provided the writer with references, examples and tips along the way.

3.5.2 TextChange Listener

A textChange flag was used in the textChange event to store the status of a text change occurring in the text area. However, the TextChange event seems to be triggered even after a file is loaded.

The problem was solved by using the setText method instead of the append method to output the file to the text area. This is because, these two methods trigger the TextChange event differently.

3.5.3 Printing

Calculating of the page format was not correct because the dpi read from the system was not correct. Calculation of the printable space was not accurate when double was used to store the values. The values will only be reasonable by using integer casting which rounds off the values.

Another problem with the getCopies method of the printerJob class is that it always gives the value of '1' when the value was printed on the screen. However, it works fine when the value was used in a `for` loop.

3.6 Tips

3.6.1 Organization of Files

Proper file organization will lead to effective programming. For example the use of package namespace to group files.

For instance, in directory c:\pset, you have a few files under the package, `pset`. Compilation and execution of the program will be done in the root directory, one directory below the package directory.

Compilation : javac pset\Ps1.java

Execution : java pset.Ps1

3.6.2 AWT and Swing

The original design goal of the Graphical User Interface (GUI) library in Java 1.0 was to allow the programmer to build a GUI that looks good on all platforms. That goal was not achieved. Instead, the Java 1.0 *Abstract Window Toolkit* (AWT) produces a GUI that looks equally mediocre on all systems. However, it does have some restrictions. You can use only four fonts and you cannot access any of the more sophisticated GUI elements that exist in your operating system. The Java 1.0 AWT programming model is also awkward and non-object-oriented.

Much of the situation was improved with the Java 1.1 AWT event model, which takes a much clearer, object-oriented approach, along with the addition of Java Beans, a component programming model that is oriented toward the easy creation of visual programming environments. Java 2 finishes the transformation away from the old Java 1.0 AWT by adding the *Java Foundation Classes* (JFC), the GUI portion of which is called "Swing." These are a rich set of easy-to-use, easy-to-understand Java Beans that can be dragged and dropped (as well as hand programmed) to create a GUI that you can (finally) be satisfied with.

3.7 Conclusion

Through the Text Editing Application, the writer was able to understand the concepts of Object Oriented Programming through the use of the Java language. As such, it has equipped the writer with a certain level of confidence in creating object-oriented programs using the Java language. The skill acquired in writing object-oriented programs will definitely be very useful as this is the current trend of the programming world along with the increase popularity of the java language.

Of course, more functionalities (e.g. copy, paste, find, undo, etc.) can be added to the Editing Application. However, it is not the main objective of this exercise to build a fool proof commercial editor.

The use of the Java 1.1 event model enables any event type `XEvent` to be dispatched directly to a corresponding `processXEvent()` method. This is more efficient than the Java 1.0 event model where the program needs to determine which component triggers the event from the event handle.

The knowledge gained in proper organisation of files will be a key pointer to building large systems in future. Not only does it help in the development of the system, more importantly, it will help anyone who took over the implementation or maintenance of the system.

However, due to shortage of planning and brainstorming, some of the algorithms used were not very effective & efficient. Hence, proper planning will be the key to good programming.

Chapter Four – Distributed Computing with Applets & Servlets

4.1 Applets Overview.

An applet, as the name implies, is a kind of mini-application, designed to be run by a Web browser, or in the context of some other "applet viewer." Applets differ from regular applications in a number of ways. One of the important differences is that there are a number of security restrictions on what applets are allowed to do. An applet often consists of untrusted code, so it cannot be allowed access to the local file system, for example.

4.2 Capabilities of Applets

Below is an overview of the capabilities an applet has.

4.2.1 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 java applications can't 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 (from a directory in the user's `CLASSPATH`) have none of the restrictions that applets loaded over the network do.
- Although most applets stop running once you leave their page, they don't have to.

For further reading on applets, please refer to http://bruceecke.com.

4.3 Objectives – Text Editing Applets

This exercise provides an introduction to distributed computing. The objective is to convert the text editing application to run as a downloadable applet. This raises issues about applet security that severely restricts functionality, particularly file access and printing. However, these functions may be remotely handled by a servlet.

Requirements

d. Setting up web server

e. Client/server computing with applets and servlets (Using JSDK)

The next section covers some drawbacks of applets.

4.4 Drawbacks of Applets

Below is an overview of the drawback of an applet.

4.4.1 Security Restrictions

Every browser implements security policies to keep applets from compromising system security. This section describes the security policies that current browsers adhere to. However, the implementation of the security policies differs from browser to browser. Also, security policies are subject to change. For example, if a browser is developed for use only in trusted environments, then its security policies will likely be much more lax than those described here.

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 than 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`. Your applet can catch this `SecurityException` and react appropriately.

However, as stated earlier, some of the functions particularly file access and printing can be remotely handled by a servlet.

4.5 Servlets Overview

Servlets provide a Java-based solution used to address the problems currently associated with doing server-side programming, including inextensible scripting solutions, platform-specific APIs and incomplete interfaces. Servlets are objects that conform to a specific interface that can be plugged into a Java-based server. Servlets are modules that extend request/response-oriented servers, such as Java-enabled web servers.

For example, as shown in *figure 4.1* below, 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.

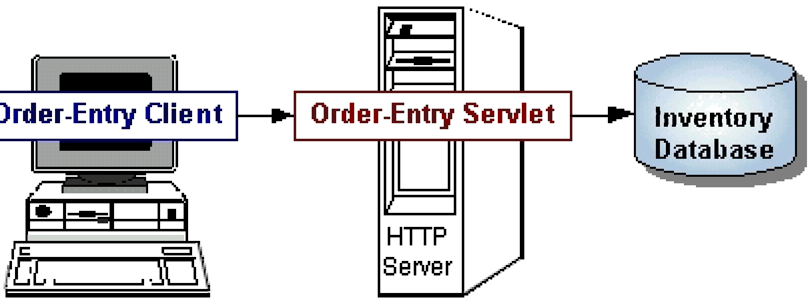


Figure 4.1 : Example of a servlet

In summary, servlets are to servers what applets are to browsers. Unlike applets, however, servlets have no graphical user interface.

For further reading on servlets, please refer to

http://www.oreilly.com/catalog/jservlet/chapter/ch03.html

4.6 Specifications

The top level flow diagram of the text editing applet is as shown *figure 4.2* below. It specifies the minimal required functions needed. Explanations of the implementation will be done in the next section.

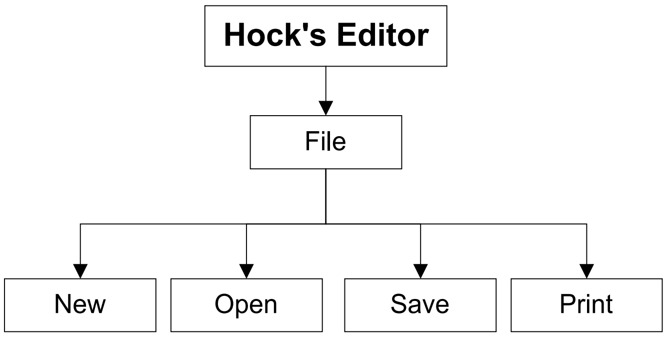
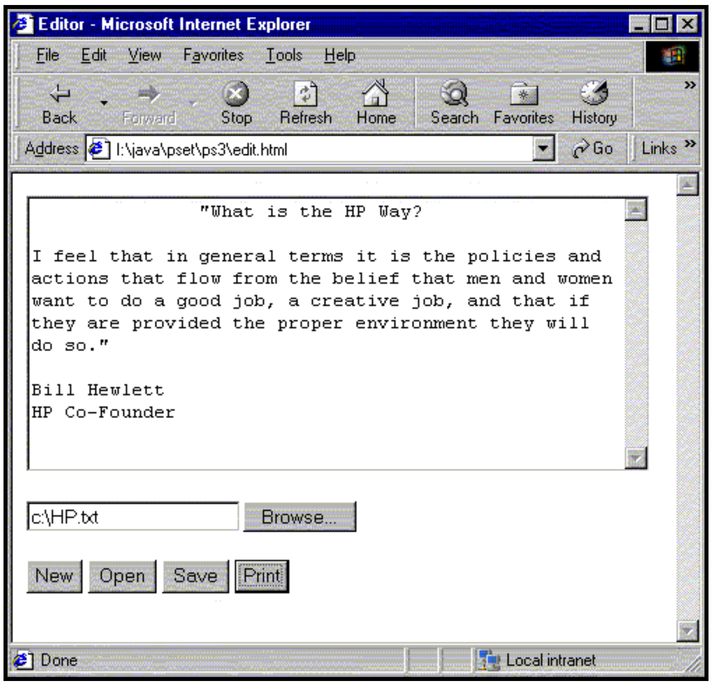


Figure 4.2 : Top level flowchart of Text Editing Applet

Although the GUI can be implemented using applets, HTML proofs to be a faster method of implementation. As such, the GUI was written using HTML instead of applets. The use of HTML is another method of creating a client/server application.

The basic GUI is shown in *figure 4.3* below. It consists of a textarea component, a file upload component and a few command buttons. In addition, there is a hidden field for storing the name of the opened file.



```
                "What is the HP Way?

I feel that in general terms it is the policies and
actions that flow from the belief that men and women
want to do a good job, a creative job, and that if
they are provided the proper environment they will
do so."

Bill Hewlett
HP Co-Founder
```

c:\HP.txt

Figure 4.3 : GUI Implemented Using HTML

Although using HTML is a faster solution for producing the GUI, it has a number of shortcomings as compared to using applets. The advantages of using applets will be discussed in more details later in the chapter.

4.7 Implementation

The basic GUI was done using FrontPage. Upon clicking a command button (i.e. submitting an HTML form), the HTML file will invoke the servlet to process the request (e.g. Open, Save, Print, etc) through either the GET or the POST method. Over here, the POST method was the one chosen due to some limitations in the GET method which will be discuss later in the chapter. After processing each request, the servlet will then repaint the page in HTML format with the necessary update.

For example, if the user clicked the New command, the servlet will repaint the HTML page with an empty text area. However, if the user clicked the Open command with a valid filename specified, the servlet will read the file and repaint the HTML page with the text area containing the file.

Implementations of the various functional blocks were similar to that of the text editing application in Chapter 3 with a few slight modifications as listed below.

a. Save_As Command

A Save_As command button is not needed in the current applet program. Instead, the File-Save_As module will be invoke when there is a valid file specified in the file upload text field when the Save button is depressed.

b. Exit Command

An Exit command button is also not needed in the web browser environment as it already contains one. Furthermore, it is conceptually not needed in a web browser environment.

c. Event Capturing

Capturing of an event is not done using an event listener but by reading the attributes of the URL to determine which button is depressed. As such it may be slower and less efficient as compared to the text editing application because it involves checking which button did the user select.

d. TextChange

In the text editing application, checking for text change before saving file was done by a textchange event provided by Java previously. Over here, upon loading a file, the initial text will be stored in a variable. When the save command is invoke, the initial text will be compared with the text in the text area. However, this method of implementation may waste resources when the size of the file is large.

4.8 Possible Enhancement

Although the use of HTML as a tool for creating the GUI may be a solution, it does have some shortcomings. For a distributed system, the time spent in repainting each page after each user command may slow down the operations of the entire system. Furthermore, for a tiny modification on the GUI, the entire page has to be repainted.

One solution to this problem will be to return to creating the GUI using applets, which allows the programmer to have more control of each component. You can do as much work as possible on the client before and after making requests of the server. For example, you won't need to send a request form across the internet to discover that you've gotten a date or some other parameter wrong, and your client computer can quickly do the work of plotting data instead of waiting for the server to make a plot and ship a graphic image back to you. Not only do you get the immediate win of speed and responsiveness, but also the general network traffic and load upon servers can be reduced, preventing the entire internet from slowing down.

A simplified example of one for the text editing applet is shown in *figure 4.4* below.

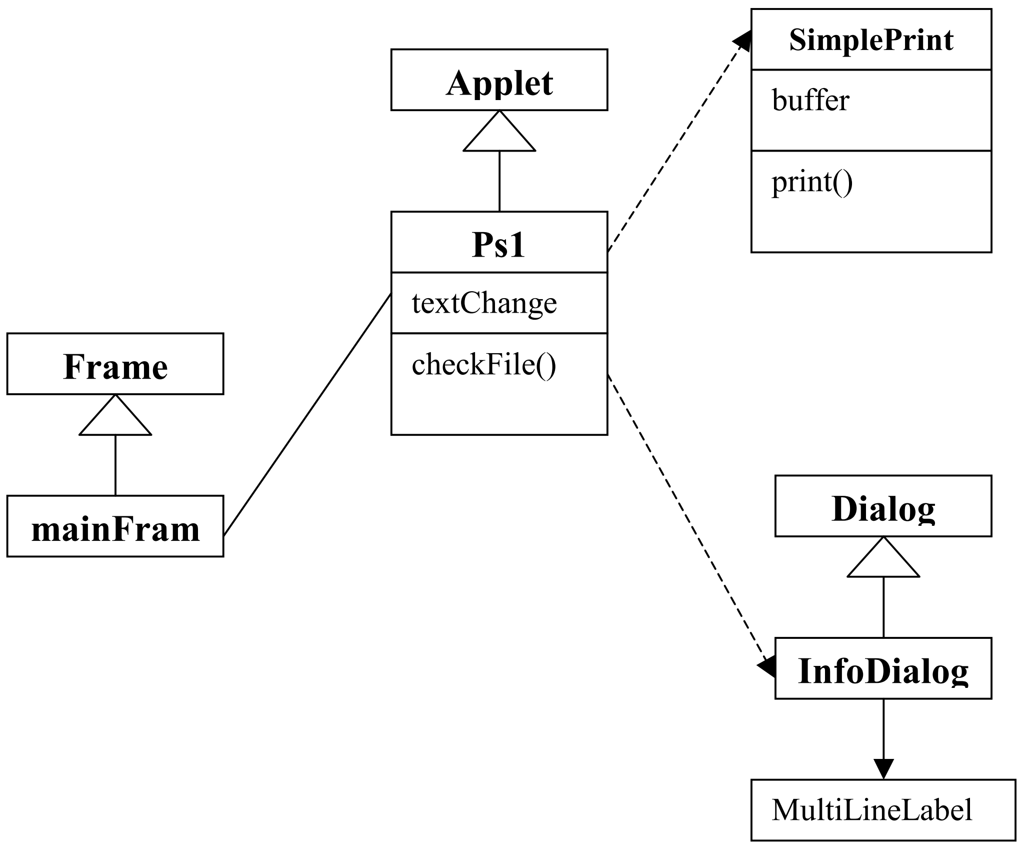


Figure 4.4 : UML of a Text Editing Applet

4.9 Problems Encountered

4.9.1 GET and POST 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. 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 your CGI input.

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 familiar encoded URL string after the question mark (?) as shown below.

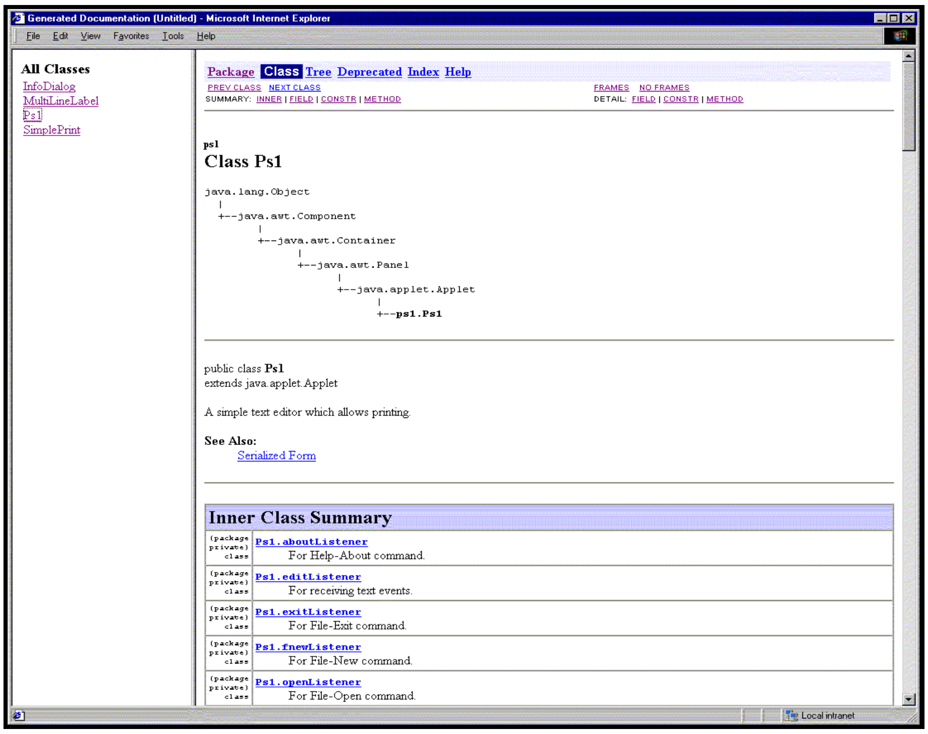
http://localhost:8080/servlet/Editor/POST?button=Print

4.10 Tips

4.10.1 Refreshing in Browser

When an explorer is used to view applet programs, use control-refresh to update the display after recompilation instead of refresh. This is because explorer caches applets, therefore refresh will only reload the old version.

4.10.2 JavaDoc


Generated Documentation (Untitled) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
All Classes
InfoDialog
MultiLineLabel
Ps1
SimplePrint
Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS
FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD
Local intranet

ps1

Class Ps1

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Panel
                    |
                    +--java.applet.Applet
                          |
                          +--ps1.Ps1
```

public class **Ps1**
extends java.applet.Applet

A simple text editor which allows printing.

See Also:
Serialized Form

Inner Class Summary	
(package private) class	**Ps1.aboutListener** For Help-About command.
(package private) class	**Ps1.editListener** For receiving text events.
(package private) class	**Ps1.exitListener** For File-Exit command.
(package private) class	**Ps1.fnewListener** For File-New command.
(package private) class	**Ps1.openListener** For File-Open command.

Figure 4.5 : JavaDoc of Text Editing Application

Using JavaDoc to create a documentation of programs proofs easy and efficient. It uses some of the technology from the Java compiler to look for special comment tags that are put in the programs. It not only extracts the information marked by these tags, but it also pulls out the class name or method name that adjoins the comment. This way one can get away with the minimal amount of work to generate decent program documentation.

The output of JavaDoc is an HTML file that can be viewed with the web browser. This tool allows one to create and maintain a single source file and automatically generate useful documentation. Also, with JavaDoc, there is a standard for creating documentation, and it's easy enough that one can expect or even demand documentation with all Java libraries. The beautiful part is that it will create and format the various fields and methods in an easily readable format.

An example of one for the text editing application is as shown in *figure 4.5* above, which has the same format as that of the Sun's Java API.

4.10.3 Unified Modeling Language (UML)

As the saying goes, "A picture is worth a thousand words". Using UML for developing and documenting programs will present a quick "Helicopter View" of the program.

The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing and documenting the artefacts of software systems. It is the proper successor to the object modeling languages of three leading object-oriented methods: Booch, Object Modeling Technique (OMT) and Object-Oriented Software Engineering (OOSE). UML was originally added to the list of OMG adopted technologies in 1997, and has become the industry standard for modeling objects and components.

4.11 Conclusion

Through this exercise, the writer was able to gain some understandings on implementing a client/server application. It can be done using HTML & servlet or applet & servlet. In short, servlets are to servers what applets are to browsers.

Apart from that, some basic understandings of HTML were acquired through the process of creating the GUI using the FrontPage program.

In fact, a better markup language to use will be the Extensible Markup Language (XML), a subset of the Standard Generalized Markup Language (SGML) intended to make it more usable for distributing materials on the World Wide Web. For example, with a very “clean” HTML, apart from the colours the HTML of a document says nothing about how it should be displayed.

XML differs from HTML primarily in allowing the user to specify new tags, marking types of elements not foreseen in the HTML specification and making it possible for common off-the-shelf browsers and other software to handle such user-defined element types usefully. The main point of XML is that, by defining your own markup language, you can encode the information of your documents much more precisely than is possible with HTML. This means that programs processing these documents can “understand” them much better and therefore process the information in ways that are impossible with HTML.

However, XML elements or tags do not have any intrinsic meaning. In order to define these aspects of documents and the way they are used, a transformation process is needed. This can be achieved using the Extensible Stylesheet Language (XSL) that controls the presentation, style and content through the use of style sheet. XSL is often described as 'transforming XML for display'. The basic principle of XSL is to act as a 'processing engine' to transform any XML document into a different document of any format. An example of this format is the HTML or an equivalent language that does have accepted display characteristics.

Another important lesson learnt was the technique of documenting a program using JavaDoc. In the past, for procedural programming, comments and explanations are written and viewed in the source code itself. As such references to the variables and methods may be tedious and not object-oriented. However, with JavaDoc, documentation will be more centralised and easily readable.

With reference to the above, documentation of a program can be greatly enhanced with the introduction of Unified Modeling Language.

Chapter Five – E-Services

5.1 E-speak Overview

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

An e-service is a service available via the internet that complete tasks, solves problems, or conducts transactions. Virtually any asset – from hardware and software to businesses processes, data and expertise – can be made available as an e-service to drive new revenue streams or create new efficiencies in the internet economy.

But what does it mean by an "open" software platform?

With e-speak, users and e-services can interact regardless of their hardware or operating systems, system management strategies, development environments, or device capabilities. This is because e-speak has the unique ability to extract the information needed to offer, use, or combine e-services without having to know their detailed specifications or how they were built. So any e-speak-enabled e-service can work with any other.

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

a. **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.

b. **Negotiation**

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

c. **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.

d. **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.

It also provides a unique dynamic firewall traversal.

Below provides a simple example of an e-speak service that illustrates some of the basic ideas in the e-speak infrastructure.

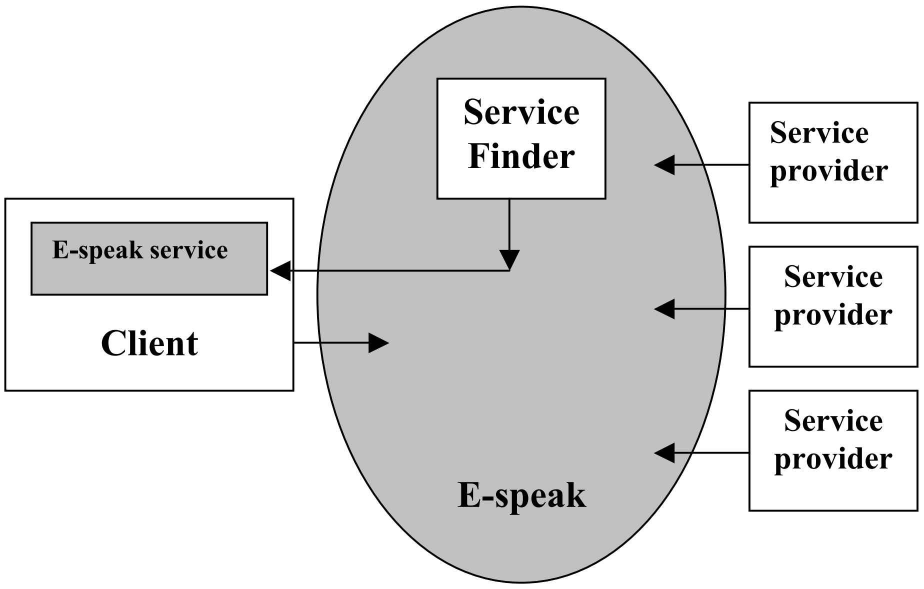


Figure 5.1 : Basic idea of E-speak infrastructure

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. Clients interact with the service with the set of interfaces for which stubs are available in the client address space. 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.

For further reading on e-speak, please refer to http://e-speak.net

5.2 Baskerville Overview

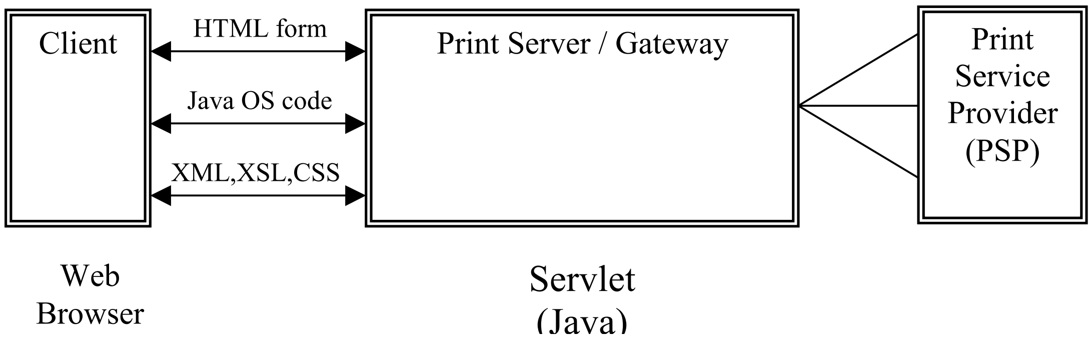


Figure 5.2 : Baskerville Overview

Figure 5.2 above illustrates the overall functional blocks namely, Client, Print Server/Gateway and the Print Service Provider (PSP).

- The user is essentially a browser interface that can select the file to be printed and the PSP that fulfil its requirements. It can also initial printing, which submits the print job to the PSP.
- The Print Server/Gateway will execute request between the Client and the PSP. The writer and his colleague, will implement part of this module.
- The PSP will be simulated with a Print Server that provides a set of print option and accept print job submission.

In summary, when the Client selects a print option, the Print Server/Gateway will search for the PSP that best matches the requirements of the user. Upon displaying to the Client the found services, negotiation between them may occur and finally the request is processed.

5.3 Objectives – E-Printing

The writer and his colleague will produce components of a workflow concerning out-sourced printing. They will each code part of the Print Server/Gateway whereby the writer's colleague will look at the front-end which has to work out what goes to the user using information from the writer. On the other hand, the writer will build the back-end services and work out how to communication with them.

The writer's colleague will reuse his existing editor's interface to allow the user to browse & open existing files, and create, edit, save files. When the user prints a file, the servlet requests the printing options from the writer and builds the corresponding dialog box. The options chosen by the user will then be sent back to the writer.

The writer will implement a print service based on existing printing code. The Print Server/Gateway will be able to find the print services upon request from the user. The print service can also be asked to list its print options that will go into the dialog box implemented by his colleague. After which, the print options selected by the users will be returned and used to invoke the print service.

Requirements

a. E-Speak and E-Speak web-access

b. Using services, interfaces and vocabularies

c. XML

5.4 Specifications

A java interface between the two work packages will be decided upon. This interface will outline the ***minimal*** requirements and dependencies between the two packages. The servlet will be the point of interaction between the two work packages. Thus, the writer and his colleague will both code part of the servlet.

5.5 Implementation

5.5.1 Property Class Definition

The very first step was to define an object for storing the print options to be used as an interface between the writer and his colleague as shown in the *figure 5.3* below.

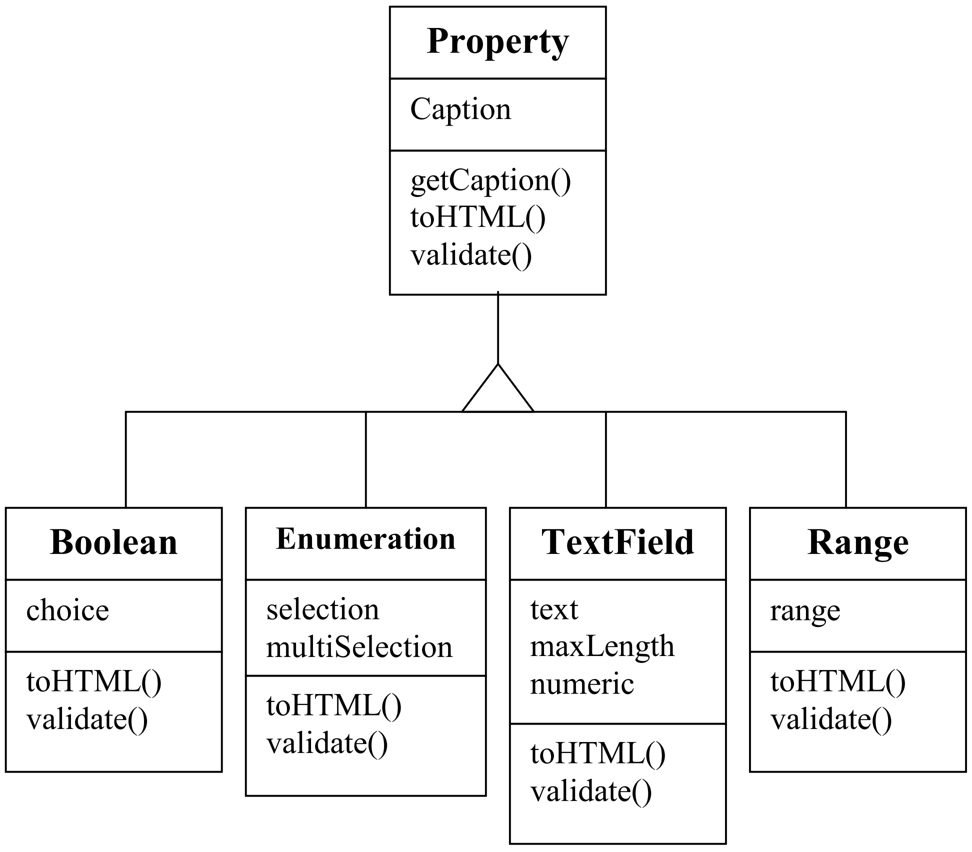


Figure 5.3 : UML of Property Class

The base object, Property will have a field caption and the 4 objects below will inherit this field. Caption is the name of the option and Boolean, Enumeration, TextField or Range etc will define the type of the option. There are also various abstract methods in each object used by the writer's colleague. Below is a brief description of the various classes.

a. Boolean

For options having a Boolean value (i.e. True or False)

E.g. Lamination	*: True/False*
E.g. Colour	*:True/False*

b. Enumeration

For options with an array of choices

E.g. Paper Size	*: A1, A2, A3, A4, etc.*
E.g. Orientation	*: portrait, landscape*

c. TextField

For options that requires a text input

E.g. Left Margin	*: 2 inches*
E.g. Right Margin	*: 1.5 inches*

d. Range

For options having an array of ranges

E.g. Page No	*: 1-3, 4-6, etc.*

However, it was not possible to use the same object to retrieve the print options from the PSP to the Print Server/Gateway. This is because the IDL interface used in E-Speak does not recognise classes with methods. IDL compiler does not recognise constructors too. As such, a similar object was created which contains no method as shown in *figure 5.4* below.

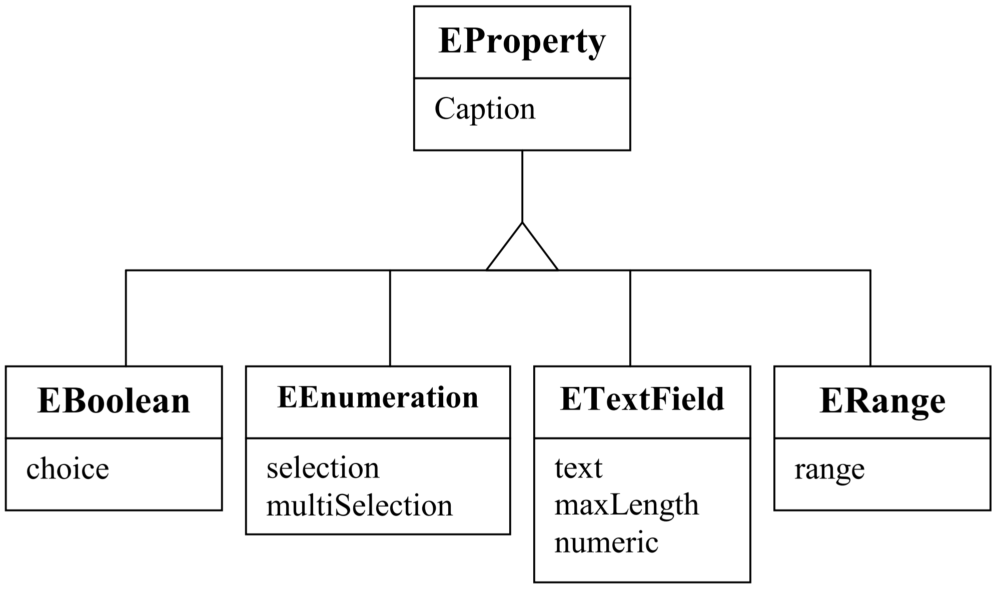
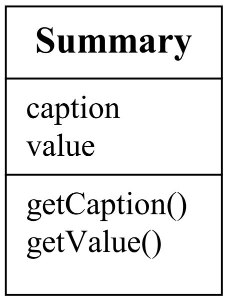


Figure 5.4 : UML of EProperty Class

Results of the selection of the options is passed back to the writer as a Summary object as shown in *figure 5.5a* below. This class will simply have a caption and a value (e.g. Lamination : True), with 2 separate methods for getting the caption and the value used by the writer's colleague.

Again, a similar object was created which contains no methods to be passed back to the print service provider through the e-speak platform. This is shown in *figure 5.5b* below.



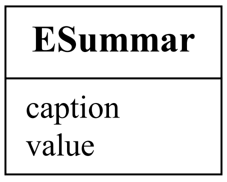


Figure 5.5a : Summary Class **Figure 5.5b : ESummary Class**

5.5.2 Starting a Print Service

Objective here is to start a print service that provides a set of print options. A sample example of the print options is shown in *figure 5.6* below.

Boolean	**Enumeration**	**TextField**	**Range**
laminated	size : A1, A2, A3, A4	LeftMargin	PageNo : 1-3, 4-6
colour	printerName : printer1, printer2	RightMargin	
	orientation : portrait, landscape	TestToPrint	

Figure 5.6 : Example of a set of print options

Below are the various file of the print service.

a. PrintIntf.esidl : Declares the interface of the print service

b. PrintImpl.java : Implements the interface PrintIntf.esidl

c. PrintServer : Task of the PSP is summarise in *figure 5.7* below.

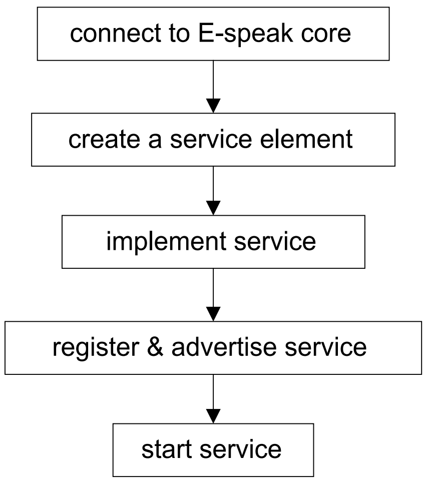


Figure 5.7 : Flowchart of the task of a Print Server

5.5.3 Starting a Print Client

Task of the print client is summarise in *figure 5.8* below.

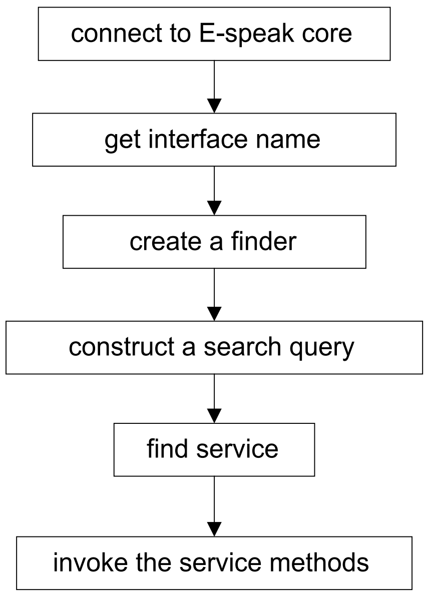


Figure 5.8 : Flowchart of the task of a Print Client

5.5.4 Adding Vocabulary to the Services

The service deployer describes the service in a chosen vocabulary by setting the attribute values appropriately using the `ESServiceDescription` class. The vocabulary describes the attributes that are used to uniquely describe a service. It also determines the names and types of the attributes used in the description of the service. This vocabulary is also registered and advertised so that other service providers or clients may find and use it.

An example of the attributes added is as shown in *table 5.1* below :

Name	String value
Manufacturer	String value
Location	String value
DPI	Integer value

Table 5.1 : Example of attributes of a Print Service

Clients on the other hand will need to find this vocabulary first and then use it to find the required service.

5.5.5 Creating Multiple Print Services

Objective here is to start more than one print service. However, each print service will have different service description for users to choose from. Service descriptions are sets of attribute-value pairs expressed in a certain vocabulary describing the service.

Service descriptions of the print services may be set as shown in *table 5.2* below. The only differentiating factor between the two services created is the location.

	Service 1	Service 2
Name	Print1	Print2
Manufacturer	HP	HP
Location	**UK**	**US**
DPI	72	72

Table 5.2 : Example of different service description

Note : Description of the print vocabulary is only done in Service 1 whereas Service 2 will only requires to find the vocabulary and add attributes to it. As such, Service 1 must be started before Service 2.

5.5.6 Invoking the Print Services from a Servlet

This section will form part of the integration to be made with the writer's colleague. The objective will be to connect to the e-speak core and find the services from a servlet.

This was done simply by calling the methods of the clients from the servlet. The methods being called consisted of :

a. Connecting to the e-speak core and finding the services using certain constraints.
b. Retrieving the print options from the PSP.
c. Sending the user selected options back to the PSP.

5.5.7 Integrating with Front-End

The objective here is to integrate the writer's code (back-end) with his colleague's code (front-end).

The implementation done was to have two processes, one process for the program execution after the user initiate a print job. Another process for program execution after the user selected and finalised the print options.

- Process 1

  a. Connect to e-speak, find the vocabulary and the PSP.

  b. Get the print options from the PSP.

  c. Convert the print options (EProperty) to a Property object used by the front-end.

- Process 2

  a. Convert the Summary object returned by the front-end to ESSummary object used by the service interface.

  b. Process the print job

5.6 Problems Encountered

Although invoking the print services from a servlet was just an addition of a few lines of code, a number of details were overlooked.

5.6.1 Service Not Found

First of all, the servlet was not able to find the various services. However, after much de-bugging, it was because some classes were used without specifying the package name in front.

For example, the class PrintIntf was written in a package called "espeak" and the servlet program was written in another package called "testing". The servlet program should use the class name as ***espeak.PrintIntf*** instead of ***PrintIntf.***

However, using the findAll method of e-speak to find the service when the package name was not specified gave an interesting result. It actually finds zero services without throwing an exception.

Note : Problem still under investigation.

5.6.2 No Matching Constructor Found

Another error occurs during compilation when the compiler shows an error message that it was not able to find any matching constructions in some of the Property object (e.g. Boolean, Enumeration, TextField and Range) defined in another package.

This error occurs because these constructors written in package “espeak” was not declared as public, as such the servlet program written in package “testing” was not able to access it.

5.7 Tips

5.7.1 JAR Files

In the past, to run the servlet, all the class files required by the program must be copied to the running directory of server, which can be rather tedious at times.

E.g. c:\jsdk2.1\webpages\web-inf\servlet

In order to solve this problem, the writer had combined all the class files into a single JAR (Java Archive) file and placed it on the CLASSPATH. This will enable the server to reference the files.

This single, compressed file (it is a ZIP file essentially) can also be transferred (loaded) from the web server to the browser more efficiently.

5.7.2 Danger in Too Much Pathing

One danger for placing too many jar files explicitly on the `CLASSPATH` is that the system may reference the wrong file if there are more than one files which contains classes with the same name.

This problem was encountered during the writer's initial testing done to integrate the back-end and the front-end in section 5.5.6 earlier. During the testing phrase, a class, named `PrintClient` was use, which contains a method `connectToEspeak(String serviceName)`. The compiled class files, `I:\e-speak\test.jar` was placed in the `CLASSPATH`.

In the actual integration shown in section 5.5.7, the same class was used, but a method with no argument (i.e. `connectToEspeak()`). Again, another jar file, `I:\e-speak\eprint.jar` was placed in the `CLASSPATH`.

As the program was executed, an exception was thrown showing that, method `connectToEspeak()` was not found. After much de-bugging, it was because the program was referencing to the wrong file (i.e. `connectToEspeak(String serviceName)`). This example shows the problem with `CLASSPATH`.

One solution is to delete the jar file (`I:\e-speak\test.jar`) from the `CLASSPATH`. However, this may not be a very good solution.

What the writer had done was to create a batch file for setting the environment for each program or working environment. First, the variable `Path` and `CLASSPATH`, `ESPEAK_HOME`, `JAVAC` and `JRE` were deleted from the environment. Next a batch file was created for setting the environment of the current program as shown in *figure 5.9* below. This batch file will set up the environment for e-speak and also path the file, `eprint.jar`.

```
@echo OFF

set MYDIR=C:\e-speak\lib

set MYPATH=%MYDIR%\esi.jar;%MYDIR%\escore.jar;%MYDIR%\esservices.jar
set MYPATH=%MYPATH%;c:\jdk1.2.2\jre\lib\rt.jar;c:\jsdk2.1\servlet.jar
set MYPATH=%MYPATH%;i:\e-speak\eprint.jar

set CLASSPATH=.;%MYPATH%

set Path=C:\WINNT;C:\WINNT\System32;C:\Perl\bin;C:\jdk1.2.2\bin;c:\jsdk2.1

set ESPEAK_HOME=c:\e-speak
set JAVAC=c:\jsk1.2.2\bin\javac
set JRE=c:\jdk1.2.2\bin\java

@echo Done!!!
```

Figure 5.9 : Batch File for Setting Environment

As such, the environment created will be temporary and it will exist (be created) as and when required.

That is,

- to run program 1, invoke the batch file to set up the environment of program 1 in a command prompt window.
- to run program 2, invoke the batch file to set up the environment of program 2 in another command prompt window.

Effectively, this creates two temporary environments with different `CLASSPATH`. Hence 2 programs with different environment can be run simultaneously in the same machine.

Although this method may seem troublesome, it will be useful when developing large and complicated systems.

Another point noted by the writer was that, the original `path` variable in the environment includes the directories, `C:\WINNT` and `C:\WINNT\System32`. These two directories contain a lot of files, which may cause referencing problems.

For example, the Java1.2.2 compilation and execution file (javac & java) were invoked from the directory `C:\WINNT\System32` instead of `C:\jdk1.2.2\bin`. This is because upon installation, a copy of the file `java.exe` and some other java `dll` files were also placed in directory `C:\WINNT\System32`.

In Summary, when there is any unresolved bug in a program, it is worthwhile to check this directory.

5.8 Conclusion

The involvement of the writer with e-speak had exposed him to e-commerce. With e-speak, e-services that are capable of intelligent interaction can dynamically discover and negotiate with each other, mediate on behalf of their users, and compose themselves into more complex services. Only e-speak offers all of these aspects of intelligent interaction in a single platform.

In addition, the e-speak engine software has been released to the open source community and is freely downloadable from the e-speak website (www.hp.com/e-speak/developers), along with the HP e-speak Services Framework Specification.

So far, APIs have been used to create and manage descriptions in Java. In addition to these APIs, J-ESI supports XML description of vocabularies and services using constructors. XML is the preferred mode for describing vocabularies and services. Typically, XML is used for data descriptions passed as parameters to the description constructors.

The use of JAR files for clustering class files had helped in the development stage of the project by reducing the time spent on unnecessary file manipulation.

In relation to the above, the in-depth understanding of setting environments and the usage of batch files for doing this was an important finding which will be very useful in any System.

Finally, the successful integration of the writer's code (back-end) with his colleague's code (front-end) was indeed an achievement.

The next phrase will be to understand the mediation process and participate in coding it.

Chapter Six – Conclusion

First of all, being able to work overseas enabled the writer to be exposed to the working environment in other parts of the world. This includes learning and understanding the cultures of the country. In other words, it is beneficial to understand the view of someone from a different culture, which can best be acquired by living and working in the country itself. It helps especially when interacting with people from different walks of life in his future working life. This is increasing in importance in the world of business and is a skill that is highly prized by employers.

The writer also took the opportunity to get to know how people (e.g. Senior engineers) go about analysing and solving problems, including their attitude towards them. Undoubtedly, this will dramatically influence the way he thinks and works in future. Having learned the experiences of others enabled him to analyse and produce a better way to solve his own problems and at the same time exercised his innovation and creativity. The writer believes this will help him to adjust himself more effectively and efficiently to future employment, and adapt to foreign work environments during overseas assignments.

This trip has also provided him with hands-on training that were far more beneficial than theory from books and further expands his technical knowledge base. This training will put him in good stead for prospective job opportunities.

Apart from seeing the highly advanced technology used in the company, he also had the opportunity to meet senior management staff who had offered their invaluable advice. From them, he gets to understand their management style.

Furthermore, it is not just about learning another culture and another society, it is about learning about the writer's own as well.

Finally, the writer feels that this trip was very memorable and worthwhile.

References

1. Java in a Nutshell, a Desktop Quick Reference by David Flanagan
2. Java Examples in a Nutshell, a Tutorial Companion to Java in a Nutshell, by David Flanagan
3. Thinking in Java

   http://www.bruceeckel.com
4. Sun's Java Tutorial

   http://www.java.sun.com

   http://www.java.sun.com/docs/books/tutorial
5. Sun's Java API Index

   http://www.java.sun.com/products/jdk1.2/docs/api/index.html
6. Sun's Java Web Server Tutorial

   http://www.jserv.java.sun.com/products/java-server/documentation/webserver1.1/servlets/servlet_tutorial.html
7. Sun's JSDK API Index
8. O'REILLY's Java Servlet Programming

   http://www.oreilly.com/catalog/jservlet/chapter/ch03.html
9. E-speak

   http://www.e-speak.net
10. E-speak Programmer's Guide

    http://www.e-speak.net/library/pdfs/Jesi-PgmGuide.pdf
11. E-speak API Index