# The Hand Drawn Web Editor

A thesis presented in partial fulfilment of the requirements
for the degree of
Master of Science in Computer Science at Massey University

Meihua Cui
1999

# Acknowledgement

Firstly, I would like to thank Professor Chris Jesshope, my thesis supervisor, for helping me to draft out the contents. This thesis would not have been possible without the patient, advice and guidance from him.

Next, I would like to express my appreciation to all staff and postgraduate students in the Department of Computer Science for their assistance and friendship. Thanks go to Mr. Horia Slusanschi, Ms. Regina Gehne, Ms. Jane Q.Zhao, and Mr. Yongqiu Liu for their assistance.

I would like to thank Massey University Library for using the facilities for my literature search.

Lastly, I would like to thank my husband, Jiong Zheng, and my family for their continued support and encouragement.

Meihua Cui, Bs(InfoSc)

Master of Science (Computer Science) candidate,
Massey Unviersity,
Computer Science,
Institute of Information Science and Technology,
Massey University,
Palmerston North,
New Zealand.

# Abstract

The Web is increasingly the most important part of the Internet for many users. Millions of new Web pages are being posted in the Internet everyday. The Internet has also become a mass-medium for lecturers distributing the lecture notes.

Most of the Web editors currently available in the market can not provide the users, especially the lecturers, with a convenient way to handle special scientific symbols or characters that are not on the keyboard directly. It always takes several steps to insert or edit those special characters. It slows down the data input dramatically.

Hand Drawn Web Editor (HDWE) is a stand-alone electronic publishing application. It is designed to provide the user with the integrated environment to edit and browse Html documents. It can also provide a user with a Hand Drawn Panel (HDP) so that he or she can input and edit special scientific symbols and characters freely upon the request.

The development environment, frameworks, tools have been discussed in detail. The full development life cycle has been documented using Rational Rose. Some problems have been encountered and their solutions have been described.

# Table of Contents

# List of Figures

# List of Acronyms

| | | |
|---|---|---|
| API | : | Application Programming Interface |
| AWT | : | Abstract Windowing Toolkit |
| GUI | : | Graphical User Interface |
| JAR | : | Java Archive File |
| JDK | : | Java Development Kit |
| JIT | : | Just In Time Compiler |
| JFC | : | Java$^{TM}$ Foundation Classes |
| HDP | : | Hand Drawn Panel |
| HDWE | : | Hand Drawn Web Editor |
| IDE | : | Integrated Development Environment |
| MDI | : | Multiple Document Interface |
| MVC | : | Model-View-Controller architecture |
| PL&F | : | Pluggable Look and Feel |
| RMI | : | Remote Method Invocation |
| VM | : | Virtual Machine |

# Chapter 1

# Introduction

## 1.1 The motivation for Developing Hand Drawn Web Editor (HDWE)

Many people, especially the lecturers, need to create HTML documents that contain many special scientific symbols or characters for their students. It is very important to have a convenient way to insert or edit those symbols into a web page. However, after examining several web editors currently available in the market, such as Word97, HotDog, Claris Homepage, HtmlPad, and Internet Editor, we found that it is very inconvenient for a user to insert or edit special symbols. The following three examples show how hard it is to handle superscripts, equations, and special characters using Microsoft Word 97, which is considered to be the most full-fledged text editor:

Example 1.

Superscript is the text that appears slightly higher than the other characters on the same line. In order to create superscript text or numbers, a user needs to:

1. Select the text the user wants to format as superscript.
2. On the Format menu, click the Font menu item, and then click the Font tab.
3. Select the Superscript check box.

Example 2.

When trying to insert an equation into the document, a user needs to:

1. Click where the user wants to insert the equation.
2. On the Insert menu, click the Object menu item, and then click the Create New tab.
3. In the Object type box, click Microsoft Equation 3.0.
4. Select or clear the Float over text check box.
5. Click OK.
6. Build the equation by selecting symbols from the Equation toolbar and typing variables and numbers. From the top row of the Equation toolbar, the user can choose

from more than 150 mathematical symbols. From the bottom row, the user can choose from a variety of templates or frameworks that contain symbols such as fractions, integrals, summations, and so on.

7. To return to Word, click the Word document.

Example 3.

A user can insert special characters, international characters, and symbols by using the Symbol command on the Insert menu. The user can also insert a character or symbol by typing the character code on the numeric keypad. In order to quickly insert a symbol that a user uses frequently, the user may needs to assign the symbol to a shortcut key.

To insert those symbols that are not on the keyboard, a user needs to:

1. Click where the user wants to insert the symbol.
2. On the Insert menu, click the Symbol menu item, and then click the Symbols tab.
3. Double-click the symbol or character the user wants to insert.

These three examples from Microsoft Word97 show us that, it is very inconvenience to handle those characters that are not on the keyboard directly. It slows down the data input - a user has to choose from several menu items or toolbar buttons to perform insertion or edition and this is time-consuming. However, if the user has been provided with a draw panel and an interface (from which the user can choose different kinds of pen style and color), the user can input and edit special symbols and characters conveniently. On the draw panel, the user can hand write or draw anything freely, just like traditional blackboard. The user can draw random curves, straight lines, or other primitives, such as circles, squares, ellipses, rectangles, etc. This is simple and intuitive. It is because of this that the Hand Drawn Web Editor (HDWE) was designed and implemented.

## 1.2 Objective of HDWE

The major functionality of the HDWE is to provide the user with an integrated environment to edit and browse Html documents. The HDWE needs to have an html editor that can be used to edit Html documents, and a browser to display the Html documents. In the addition, whenever the user wants to insert or edit those special symbols, the hand draw panel can be activated. After the user finishes the hand writing, the content of the drawing panel can be automatically embedded into the Html document as an Applet at a given position with the right size. In HDWE, a hand drawn object is represented as a Java Applet within a Html document. If the Html document contains a Java Applet, a Web browser that supports Java can be used for its display.

## 1.3 Object-oriented Programming Languages

Although dozens of Object-Oriented languages have been introduced over the past decade, only a few have gained any significant foothold in the marketplace, that is: Smalltalk, Eiffel, C++, and Java [98].

Smalltalk, a "foundation" object-oriented language, was originally developed in the early 1970s to explore OO concepts. Today, versions of Smalltalk are available on computers of all types, although the use of the language for the development of products and industry quality systems is limited.

Eiffel is one of a number of "new" object-oriented languages that are robust enough for industry applications. Like C++ and Smalltalk, Eiffel provides direct support for class definitions, inheritance, encapsulation, and messaging.

C++ is an object-oriented version of C. It is compatible with C (it is actually a superset). C++ programs are fast and efficient, qualities which helped make C an extremely popular programming language. It sacrifices some flexibility in order to remain efficient.

Java(tm) is a "simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, dynamic, buzzword-compliant, general-purpose programming language" [3]. Java supports programming for the Internet in the form of platform-independent Java applets. Java is object oriented from the ground up and draws on the best concepts and features of previous object-oriented languages, primarily Eiffel, SmallTalk, Objective C, and C++. Java goes beyond C++ in both extending the object model and removing the major complexities of C++.

### 1.3.1 How the Java Language Differs from C and C++

It is no secret that the Java language is highly derived from the C and C++ languages. Because C++ is currently considered the language of choice for professional software developers, it is important to understand what aspects of C++ Java inherits. Of possibly even more importance are what aspects of C++ Java doesn't support. Because Java is an entirely new language, it was possible for the language architects to pick and choose which features from C++ to implement in Java and how. Listed below is detailed information about the differences between these languages:

*The Preprocessor*

The C++ preprocessor basically performs an intelligent search and replace on identifiers that have been declared using the #define or #typedef directives. The problem with the preprocessor approach is that it provides an easy way for programmers to inadvertently add unnecessary complexity to a program. Java does not have a preprocessor. It provides similar functionality (#define, #typedef, and so on) to that provided by the C++ preprocessor, but with far more control.

*Pointers*

Most developers agree that the misuse of pointers causes the majority of bugs in C/C++ programming. The Java language does not support pointers. Java provides similar functionality by making heavy use of references. Java passes all arrays and objects by reference. This approach prevents common errors due to pointer mismanagement.

*Structures and Unions*

There are three types of complex data types in C++: classes, structures, and unions. Java only implements one of these data types: classes. Java forces programmers to use classes when the functionality of structures and unions is desired. Although this sounds like more work for the programmer, it actually ends up being more consistent, because classes can imitate structures and unions with ease.

*Functions*

In C, code is organized into functions. C++ added classes and in doing so provided class methods. However, because C++ still supports C, there is nothing discouraging C++ programmers from using functions. This results in a mixture of function and method use that makes for confusing programs. Java has no functions. Being a pure object-oriented language than C++, Java forces programmers to bundle all routines into class methods. There is no limitation imposed by forcing programmers to use methods instead of functions. As a matter of fact, implementing routines as methods encourages programmers to organize code better.

*Multiple Inheritance*

Multiple inheritance is a feature of C++ that allows a programmer to derive a class from multiple parent classes. Although multiple inheritance is indeed powerful, it is complicated to use correctly and causes many problems otherwise. It is also very complicated to implement from the compiler perspective. Java takes the high road and provides no direct support for multiple inheritance. A programmer can implement functionality similar to multiple inheritance by using interfaces in Java.

*Strings*

C and C++ have no built-in support for text strings. The standard technique adopted among C and C++ programmers is that of using null-terminated arrays of characters to represent strings. In Java, strings are implemented as first class objects (String and StringBuffer), meaning that they are at the core of the Java language.

*The goto Statement*

The dreaded goto statement is pretty much a relic these days even in C and C++, but it is technically a legal part of the languages. The goto statement has historically been cited as the cause for messy, impossible to understand, and sometimes even impossible to predict code. Java does not provide a goto statement. The Java language specifies goto as a keyword, but its usage is not supported. Not including goto in the Java language simplifies the language and helps eliminate the option of writing messy code.

*Operator Overloading*

Operator overloading, which is considered a prominent feature in C++, is not supported in Java. Although roughly the same functionality can be implemented by classes in Java, the convenience of operator overloading is still missing. However, in defense of Java, operator overloading can sometimes get very tricky. No doubt the Java developers decided not to support operator overloading to keep the Java language as simple as possible.

*Automatic Coercions*

Automatic coercion refers to the implicit casting of data types that sometimes occurs in C and C++. For example, in C++ you can assign a float value to an int variable, which can result in a loss of information. Java does not support C++ style automatic coercions. In Java, if coercion will result in a loss of data, a programmer must always explicitly cast the data element to the new type.

*Variable Arguments*

C and C++ let a programmer declare functions, such as printf, that take a variable number of arguments. Although this is a convenient feature, it is impossible for the compiler to thoroughly type check the arguments, which means problems can arise at runtime. Again Java takes the high road and doesn't support variable arguments at all.

*Command-Line Arguments*

The command-line arguments passed from the system into a Java program differ in a couple of ways from the command-line arguments passed into a C++ program. First, the number of parameters passed differs between the two languages. In C and C++, the system passes two arguments to a program: argc

**Chapter3** describes the specification of the HDWE system, and analyzes the use case, the actor, and the use case diagrams of the HDWE system. The architecture of the HDWE has been designed using Rational Rose. The main functionality, architecture, and design patterns of the Html package and the Java 2D package, which are two main Java API packages that have been used by the HDWE system, are described in detail.

**Chapter4** A lot of problems have been encountered during the implementation of the HDWE, some of these problems and solutions are discussed in detail in this chapter. Such as:

- how to use Resource bundles to contain locale-specific objects
- how to create a generalized UI Creator to create UI
- how to implement the Item class so that it can be serialized
- how to set the appropriate content type for the JEditorPane
- how to insert an Applet from the HDP (Hand Drawn Panel) into the current Html document.

**Chapter5** discusses the results, achievements, and the list of the further work needed to be done. Such as:

- Implements AppletView class
- Extends the functionality of the HDP so that a user can select, resize, zoom in or out, rotate, stretch or skew.
- Completely resolve the MDI (Multiple Document Interface) concurrency problem
- Completely provide undo redo support for the whole system
- Provides the user with a context sensitive help.

The definition, Pros, and Cons of the Framework are also discussed in this chapter.

# Chapter 2

# Details of Environment

## 2.1 The Java Phenomenon

### 2.1.1 What Is Java?

Java is two things: a programming language and a platform.

#### 2.1.1.1 The Java Programming Language

Java is a high-level object-oriented programming language. Java is unusual in that each Java program is both compiled and interpreted. The compiler translates a Java program into an intermediate language called Java byte-codes--the platform-independent codes interpreted by the Java interpreter. With an interpreter, each Java byte-code instruction is parsed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed. Figure 2.1 illustrates how this works.

Java Program
Interpreter

### 2.1.1.2 The Java Platform

A platform is the hardware or software environment in which a program runs. The Java platform differs from most other platforms in that it is a software-only platform that runs on top of other, hardware-based platforms. Most other platforms are described as a combination of hardware and operating system.

The Java platform has two components:

*The Java Virtual Machine (Java VM)*

*The Java Application Programming Interface (Java API)*

The Java VM is the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries (packages) of related components.

The following figure depicts a Java program, such as an application or applet, that's running on the Java platform. As the figure shows, the Java API and Virtual Machine insulates the Java program from hardware dependencies.

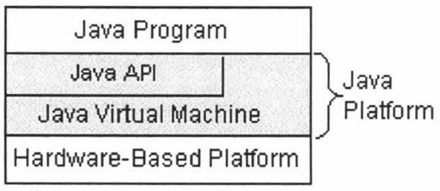


Fig.2.2 Java Platform

That extra layer between the application and the hardware and operating system uses up a lot of the system's performance. As a consequence, Java is never going to be as fast as compiled language like C. Today, with the existence of highly optimized JITs (Just in Time Compilers), most Java applications run with as little as 20-40% overhead compared to traditional optimized C++ code. And in multithreaded applications, such as applications relying heavily on I/O, no measurable performance difference exists between Java and C++, due to Java's excellent built in support for multithreading.

The Java platform supports the Write Once/Run Anywhere model of application development. Developers can compile their Java program into byte-codes on any platform that has a Java compiler. The byte-codes can then be run on any implementation of the Java VM. This, combined with the easy distribution mechanisms provided by the World Wide Web and intranets, makes Java a powerful tool for many network based systems.

Java also opens up a lot of security issues. The essence of the problem is that running programs on a computer typically gives that program access to certain resources on the host machine. In the case of executable content, the program that is running is untrusted. If a Web browser downloads and runs Java code and is not careful to restrict the access that the untrusted program has, it can provide a malicious program with the same ability to do mischief as a hacker who had gained access to the host machine. The reason that one gives programs access to resources in the first place is that in order to be useful a program needs access to certain resources. Thus, an important part of creating a safe environment for a program to run, is in identifying the resources and then providing certain types of limited access to these resources.

Java's powerful security mechanisms act at four different levels of the system architecture. First, the Java language itself was designed to be safe, and Java compiler ensures that source code does not violate these safety rules. Second, all byte-codes executed by the run-time are screened to be sure that they also obey these rules. Third, the class loader ensures that classes do not violate namespace or access restrictions when they are loaded into the system. Finally, API-specific security prevents applets from doing destructive things. The final layer depends on the security and integrity guarantees from the other three layers.

We have used the Java 2 platform to develop the last version of HDWE. The Java 2 platform was released by Sun Microsystems Inc. on Dec 8th 1998. The new release provides significant performance improvements, a new, flexible security model and a complete set of APIs.

## 2.1.2 Java API

Java API support all of these kinds of programs with packages of software components that provide a wide range of functionality. The core API is the API included in every full implementation of the Java platform. The core API gives programmers the following features:

- *The Essentials*

  Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- *Applets*

  The set of conventions used by Java applets.
- *Networking*

  URLs, TCP and UDP sockets, and IP addresses.
- *Internationalization*

  Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- *Security*

  Both low-level and high-level, including electronic signatures, public/private key management, access control, and certificates.
- *Software components*

  Known as JavaBeans, can plug into existing component architectures such as Microsoft's OLE/COM/Active-X architecture, OpenDoc, and Netscape's Live Connect.
- *Object serialization*

  Allows lightweight persistence and communication via Remote Method Invocation.
- *Java Database Connectivity (JDBC)*

  Provides uniform access to a wide range of relational databases.

Java not only has a core API, but also standard extensions. The standard extensions define APIs for 3D, servers, collaboration, telephony, speech, animation, and more.

### 2.1.3 Java Development Kit

The Java Development Kit contains the software and tools that a developer needs to compile, debug, and run applets and applications that the developer has written using the Java programming language.

The main components of the JDK are:

*Runtime Interpreter*

Is used to run standalone Java executable programs in compiled, byte-code format. The runtime interpreter acts as a command-line tool for running Java programs.

*Compiler*

Is used to compile Java source code files into executable Java byte-code classes.

*Applet Viewer*

Is a tool that serves as a minimal test bed for final release Java applets.

*Debugger*

Is a command-line utility that enables you to debug Java applications. It uses the Java Debugger API to provide debugging support within the Java runtime interpreter.

*Decumentation Generator*

Is a tool for generating API documentation directly from Java source code. The documentation generator parses through Java source files and generates HTML pages based on the declarations and comments.

*Archiver*

Is a tool used to combine and compress multiple files into a single archive file, which is commonly referred to as a JAR file.

*Digital Signer*

Is a tool that generates digital signatures for archive files. Signatures are used to verify that a file came from a specified entity, or signer.

*Remote Method Invocation Tools*

JDK includes three different tools for working with and managing remote method invocation(RMI). These tools consist of an RMI stub compiler, a remote object registry tool, and a serial version tool.

*Demo Applets and Applications*

Examples, with source code, of programming for the Java platform. These include examples that use Swing and other Java Foundation Classes.

*Source Code*

Java programming language source files for all classes that make up the Java 1.2 platform core API.

*Documentation*

This directory is created when the JDK documentation is installed. It contains release documentation, Java API specifications, developer guides, tool documentation, demos, and links to related documentation.

## 2.2 Swing

Swing is a new GUI component kit that simplifies and streamlines the development of windowing components. Windowing components are the visual components, such as menus, tool bars, dialogs and the like, that are used in graphically based applets and applications. The Swing component set is part of a new class library called the Java$^{TM}$ Foundation Classes, or JFC.

Designing a good user-interface toolkit is a complex operation -- especially in this age of cross-platform programming. To develop programs that can be written just once and can then run anywhere, programmers need a flexible set of user-interface (UI) components that are both configurable and extensible, as well as a comprehensive infrastructure for managing input and output effectively.

Swing addresses this problem using a small set of design techniques that come from two main sources: the Java Beans model (which addresses the need for component properties, persistence, and event-based communication), and a new kind of pluggable look-and-feel (PL&F) mechanism for designing components. With Swing's PL&F design tools, a developer can create components that not only can be used without modification on different computer platforms, but even have the same look and feel as components that were designed specifically for those platforms.

The following section explains the basics of the component architecture called the model-view-controller architecture, or MVC.

## 2.2.1 MVC Architecture

### 2.2.1.1 Classic MVC Architecture

In the classic model-view-controller design, every component is divided into three parts -- a model part, a controller part, and a view part. These three parts, and their connection paths, are shown in the following illustration.

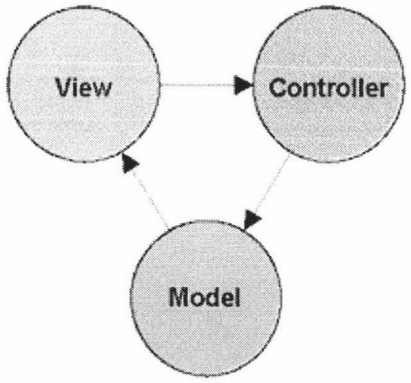


Fig.2.3 Model-View-Controller Architecture

Briefly, this is how MVC architecture works in Swing:

1. The model part of an MVC-based component provides information that can be used to specify the component's value, provided the component has any value properties. For example, the current value of a scrollbar or a slider control -- as well as the component's minimum possible value and maximum possible value -- are stored in the component's model part. (Some components do not have models.)

2. The controller part of an MVC-based component modifies information maintained by the component's model part in response to input from the user. For example, the controller part of a menu-item component notifies the component's model part when a mouse click is detected inside a given menu item.

3. The view part of an MVC-based component manages the way in which the object is drawn on the screen. For example, if a user equipped an application with a Windows-95 style menu bar and then wanted to change that menu bar's appearance to make it look more like a Solaris-style menu bar, the user would perform that operation by modifying the view part of his menu-bar component.

#### 2.2.1.2 Advantages of MVC-Based Architecture

To change the appearance of an MVC-based component, all developers have to change is its view part. To change the way it responds to user events, all developer have to change is its controller part. In this way, developers can equip a component with multiple appearances, multiple behaviors, or both, without disturbing the component's underlying architecture.

MVC architecture also makes it easy to change or extend the characteristics or capabilities of any pre-existing view or controller, without disturbing other parts of the component that is being modified.

Finally, MVC architecture lets a developer design components with appearances and behaviors that can be changed at any time -- not just at design time, but also dynamically, while the application that uses developer's component is actually being executed.

#### 2.2.1.3 Modifying the MVC Architecture

Although classic MVC architecture meets the overall needs of Swing components quite well, designing completely separate views and controllers for a component can be a difficult task. The main reason this is true is that the communication paths between the view part and the controller part of a component can quickly become quite complex and difficult to manage.

To simplify the communications-path jumble, a modification in the classic MVC design has been made: combining the view and controller parts of the classic MVC design into a single element called a view-controller. The following diagram shows how the view and controller parts of a Swing component are combined to form a combined view/controller element:

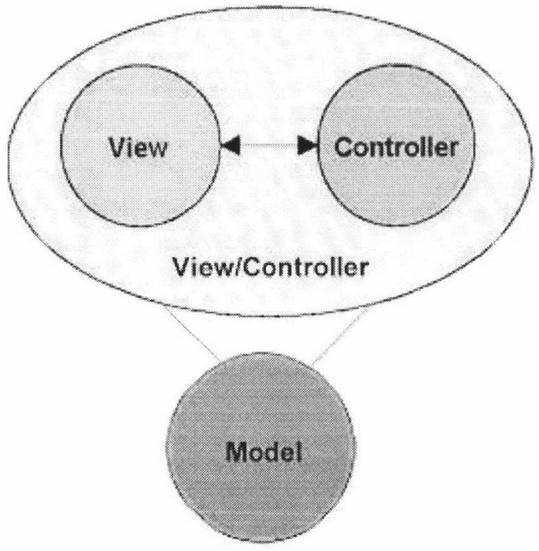


Fig.2.4 Modified MVC Architecture

In the component represented by this diagram, notice that a simple two-way arrow represents the connection paths between the component's view and component parts. There is no jumble of arrows here because the view/controller part of the component handles all communications between the view and controller parts.

Another major benefit of combining a view and a controller into a single view-controller object is that both the appearance and the behavior of a component -- that is, a component's look and feel -- can be managed together, using just one view-controller object. If Swing used the classical MVC model, which requires a view and a controller to be two separate elements, the look and feel of each Swing component would have to be managed separately. That requirement would make it more difficult to implement Swing's PL&F capabilities.

### 2.2.2 Swing´s PL&F Capabilities

PL&F is the portion of a Swing component's implementation that deals with the component's appearance (or look), as distinguished from its event-handling mechanism (its feel), which is delegated to a separate UI delegate.

PL&F is one of the most important capabilities of the Swing toolkit: a feature that increases the reliability and consistency of applications and applets deployed across platforms. The PL&F design of the Swing component set provides an easy yet powerful mechanism for individualizing an application's visual personality without

## 2.3 Integrated Development Environment

Although the JDK is certainly sufficient for professional Java development, the advanced features of third-party's IDEs can improve productivity significantly. In this section, the IDEs currently available for Java are discussed in detail.

### 2.3.1 Symantec Cafe

Symantec Cafe is the first development environment that became widely available for Java programming after the JDK.

Cafe is a sophisticated IDE that offers an excellent source editor with color highlighting of syntax, and editor for class and hierarchy modification, and a Studio tool for interface design. To aid in the design of a class hierarchy, Cafe has a class editor for navigating through classes and editing class methods, and a hierarchy editor for viewing and modifying Java class relationships. Changes in the source code that affect the class hierarchy can be seen as the program is being written, instead of requiring that it be compiled before changes are reflected in the hierarchy. A developer can also change the source code from within the class editor - clicking the function or method within a class brings up its source code in a window the developer can use to edit the code.

In the source editor, Java syntax is highlighted, making it easier to spot type and other errors immediately. The editor also uses the standard Windows cut, copy, and paste commands.

With Cafe Studio, designing a graphical user interface for the developer's Java program can be done in a visual, drag-and-drop manner. Studio enables the developer to develop the dialog boxes and other interface elements visually, and it creates event handlers for these components automatically. There's also a menu editor with an active window in which the developer can test the menu.

Cafe provides the option to use Sun's JDK compiler or the Cafe compiler. The Cafe debugger provides several different ways to temporarily halt the execution of code, including a quick-breakpoint feature for a one-time run that stops at a specific line.

The debugger also enables a large amount of control over threads in multithreaded programs. During debugging, programmers can use a watch view to monitor the contents of variables.

The environment of Symantec Cafe is highly customizable - all toolbars and palettes can be resized and placed where a user want them on-screen. Several windows can be open at the same time, making it possible to view the object hierarchy while entering source code and using the form editor.

### 2.3.2 SunSoft Java WorkShop

SunSoft Java WorkShop, the development tool offered by the Java language's home team, is an IDE written almost entirely in Java. It is one of the most approachable IDEs for a novice programmer. WorkShop uses a Web interface to offer the following features: a source editor, class browser, debugger, project management system, and Visual Java - a tool for the visual design of a graphical interface and an easier means to create windowing software.

The most striking difference between Java WorkShop and other IDEs is its interface. Java WorkShop looks more like a Web browser than a programming development environment.

The WorkShop has a source browser for viewing a class hierarchy, public methods, and variables. The browser creates HTML pages in the same format as HTML documentation generated by the JDK's *javadoc* utility. The WorkShop source editor works in conjunction with WorkShop's debugger and compilation errors create links directly into the source editor for fixing. The WorkShop debugger provides breakpoints and other methods of debugging.

The Visual Java feature provides a way to graphically design an interface, much as Cafe Studio does. Visual Java enables a developer to develop dialog boxes and other visual elements and automatically creates event handlers for these components.

The environment is not customizable in the way Cafe is, but the Web interface makes it easy to integrate other tools and programs into WorkShop. The program is a collection of Web pages with Java programs embedded in and around them. The developer can go to a different page from within Java WorkShop as easily as you can enter a URL in a Web browser. This approach makes it possible for the developer to create original pages of Java development tools that can be linked to WorkShop pages. This arrangement may be unusual for someone accustomed to development environments written as cohesive, single-independent programs linked together by HTML pages, which can be modified as individual elements without affecting the other parts of the whole.

### 2.3.3 Microsoft Visual J++

Microsoft Visual J++ is Microsoft's answer to a Java development environment. Designed to integrate with Microsoft's Visual Studio suite of development tools, Visual J++ features extensions to the Java class library that are specific to the Windows platform. Visual J++ supports the look and feel of the popular Visual C++ development environment, as well as most of its advanced features. Visual J++ provides the only real support of any IDE for fully integrating Java with ActiveX.

The Visual J++ editor is very nice and fully supports color syntax highlighting. There is also a class viewer, which shows all the Java classes that have definitions as well as the members of those classes, including properties and methods. Visual J++ has a very powerful graphical Java debugger that supports the dubugging of multiple applets simultaneously from within a browser. The debugger comes complete with bytecode disassembly, bytecode-level stepping and tracing, and the ability to assign values to variables while debugging.

There are some valid concerns over Microsoft's proprietary extensions for Java, but Microsoft knows it must adhere to the core Java API to lure developers. As long as the company continues to do that, it can bundle in as many proprietary extensions as it wants.

### 2.3.4 Metrowerks CodeWarrior

Metrowerks CodeWarrior is the only Java IDE that supports three other programming languages(C, C++, and Pascal) in the same environment. In addition, CodeVarrior supports all these languages on both the Macintosh and Windows 95/NT platforms. CodeWarrior ships standard with support for all four languages, meaning that Java developers have the option of using other languages without purchasing any additional software.

CodeWarrior has been well established in the Macintosh community for some time now as the standard Macintosh C/C++ development environment. For this reason, it isn't surprising that it has quickly gained popularity as a Java development environment - at least on the Mac. The CodeWarrior IDE includes a project manager, resource editor, text editor, graphical debugger, and class browsers - basically everything a developer expects in a professional development environment.

### 2.3.5 Tek - Tools Kawa

Kawa is an integrated development environment from Tek-Tools. As opposed to other development environments that lean more toward the "application builder" model, Kawa is indeed for those who like to get their hands into the code. If a developer want to be close to the code, but he feels going out to the command line is a little too close, then he should look at Kawa. It puts a GUI on the JDK and provides a lot of features to help the programmer program more efficiently. All of the flags, compiler options, and interpreter options of the JDK can be set through Kawa.

The current version of Kawa is 3.13. Some of the highlights of this version are JDK switching, (which Kawa has had all along), RMI compile, the ability to build JAR files, manifest files, and generate Javadoc files from within the development environment.

*Tabbed /paned work area*

The Kawa environment features a tabbed /paned work area. The developer can select Project view, Package view, Debugger view, and Help view. The package view tab is particularly nice in that it lets programmer view all the classes in the JDK with real ease.

*Direct access to the Java APIs*

A working principle of Kawa seems to be to allow the developer to have as much direct access to the Java APIs, the messages it generates, etc, as possible. When some vendors are sticking in their own proprietary code underneath the developer's class definitions, Kawa does nothing of the kind - "Just the code, ma'am". Another plus of Kawa is that it is not a resource hog. Contrary to some other IDEs which claim they run with 48 MBytes of ram, but still end up being sluggish with 64, Kawa ran fine with 32 MBytes of ram.

*GUIBuilde*

Also, Tek-Tools recently released a GUIBuilder for Kawa. There are two versions, one to build GUIs based on the AWT, and one that builds GUIs based on Swing. The GuiBuilder is fairly new, and Tek-Tools will improve it in the coming months.

Among these IDEs mentioned above, Kawa is the only one that be can be freely downloaded from the Internet, therefore, Kawa has been chosen to develop the HDWE.

# Chapter 3

# Analysis and Design HDWE

## 3.1 Use Case of the HDWE

### 3.1.1 Specifications of HDWE

The objective of the HDWE is for the users, especially lecturers, to create HTML documents for their students. This kind of document may contain many special scientific symbols or characters, for example, mathematical scripts, which are difficult to typeset. It is very important to have an easy way to insert or edit those characters. To solve this problem, HDWE is designed to provide the user with a drawing panel on which he or she can hand write or draw anything freely, just like traditional blackboard. Using HDWE, the user can draw random curves, straight lines, or other primitives, such as circles, squares, ellipses, rectangles, etc, using the normal computer-based drawing tools.

In order to provide the user an integrated environment to edit and browse the html document, HDWE needs to have an html editor that can be used to edit and browse html documents. The major functionality of the HDWE is that the user can use it to edit an html page. The user can perform this task in two modes: with or without tags. A user can switch between these two modes by selecting the View Source Menu item from the View Menu. The default mode should be the one that most users prefer - an html file without tags. A user must be able to perform the normal editing operations:

- Select, select all
- Cut, copy, and paste any portion of the document
- Undo, redo

A user must also be able to insert various html elements into the html page, such as Applet, Image, Form, URL, Table, Frame, and List. It should be noted that the hand drawn components are inserted as applets.

HDWE should have a Multiple-Document-Interface, so that it can be used to manipulate multiple files simultaneously, the file type can be one of text, html, or hand drawn files. Hand drawn files are created by the *Hand Drawn Panel* (HDP). The first two file formats can be handled by the editor panel of HDWE, and the latter one can be handled by the HDP. That is, there are two user interfaces, one is for conventional text or Html editing, the other one for hand drawing. The result of the latter being integrated into the format as is it was an image. The main manipulations of the HDWE include:

- create a new html file
- open an existing html file - this file may not have been created by HDWE
- save a file
- save a file to a different file name
- close a file

Because a user may open several files at the same time, the user can close the all these opened files with saving all at once, or prompt user to decide for each file, or close all without saving.

In HDWE, the applets are drawings that a user has drawn using HDP. When the user chooses to insert an applet into the html page, the HDP will be activated. This panel has its own user interface from which the user can choose different kinds of pen styles and color. After the user finishes the drawing, he can choose the Insert Applet menu item from the File menu to insert that drawing into the Html document as an Applet. A user can create a new hand drawn file, open, save, save as or close the current hand drawn document.

A user can use the hand drawn panel to draw anything, such as:

- Lines
- Curves
- Rectangles
- Circles

- Ellipses
- Squares

The user can choose the pen style that is applied to the outline of a shape, draw lines with any point size and dashing pattern, and apply endcap and join decorations to a line. The user can also choose the fill style that is applied to a shape's interior, he or she can fill shapes with solid colors, gradients, and patterns. The user can also choose the option called "Rendering Hints" that specifies preferences in the trade-offs between speed and quality. For example, a user can specify whether or not antialiasing should be used if it is available.

There is a color palette, from which a user can choose foreground and background colors. The Color palette consists of basic colors, which are those colors the user uses most frequently, and the other colors (by clicking on the other button) that the user can choose from an RGB or an HIS selection dialogue. The user can clear all the drawing items on Panel at once, or can use an eraser to erase one item at a time. After the user finishes a drawing, it can be inserted into the Html document by selecting the Insert Applet menu item from the File menu of the HDP.

### 3.1.2 Rational Rose

Visual modeling is a way of thinking about problems using models organized around real-world ideas. Models are useful for understanding problems, communicating with everyone involved with the project. Visual modeling gives developers a graphical representation of the structure and interrelationships of a system by constructing models of the design. This blueprint makes explicit the requirements of the project, ensuring that the final product meets the needs of the end users. By using the common language of visual modeling:

- communication is improved
- development time is shortened
- complex systems are easily understood and therefore easily constructed
- designs are made cleaner and more maintainable.

Rational Rose [2] is the world's leading visual modeling tool from Rational Software Corporation, it can be used to object-oriented analysis, modeling, and design. It allows the developers to define and communicate a software architecture.

We have chosen Rational Rose to analysis, modeling and design HDWE, because it is the only visual modeling tool available in Massey University.

The behavior of the HDWE, that is what functionality must be provided by the HDWE, is documented in a use case model that illustrated the HDWE's intended functions (use cases), its surroundings (actors), and relationships between the use cases and actors(use case diagrams) [2].

### 3.1.3 Identifying the Actors for the HDWE.

Actors are not a part of the system - they represent anyone or anything that must interact with the system. An actor may input or receive information to or from the system.

Because all HDWE users play the same role in the system, and they use the system in the same way, they are all represented by the same and only one actor. Thus we define the *user actor* - as a person who is using HDWE to maintain their html files.

### 3.1.4 Identifying the use cases

A use case is a sequence of transaction performed by a system that yields a measurable result of values for a particular actor. Use cases represent the functionality provided by the system, that is, what capabilities will be provided to an actor by the system. The collection of use cases for a system constitutes all the defined ways the system may be used.

Each use case of the HDWE has been documented with a flow of events. The flow of events for a use case is a description of the events needed to accomplish the required behavior of the use case. The flow of events is written in terms of what the system should do, not how the system does it. Because the *edit an opened file use case* is the most important use case, we document its Flow of Events in detail in follow section.

### 3.1.5 Flow of Events for the *Edit an opened file use case*

1. Preconditions

The C*reate a new file* or *Open an existing file use case* must be executed before *Edit an opened file use case.*

2. Main Flow

This use case begins when a user edits an Html document. The user can type in characters, select, cut, copy, paste, change the font of a document, or can insert various html tags and objects into the document, such as Applet, Image, Form, URL, Table, Frame and List. In HDWE, the insert Applet is the most important feature and we will discuss this in more detail:

If the user selects the Insert Applet menu item, the s-1: Insert Applet subflow is performed.
If the user selects the Insert Table menu item, the s-2: Insert Table subflow is performed.

3. Subflows

s-1: Insert Applet

If the user selects the Insert Applet menu item, the HDP will be activated. The user can draw or fill circles, ellipses, rectangles and squares by choosing the primitives and then press-drag-release the mouse on the draw panel. The user can also use the panel to manipulate the files. After he or she finishes the drawing, the user can insert the drawing as an applet into the html document by clicking on the Insert button from the toolbar or selecting the Insert menu item from the File menu of the HDP. The Applet tag and related content will be inserted into the html document automatically.

s-2: Insert Table

If the user selects the Insert Table menu item from the HDWE, the *actionPerformed* method of the **insertTableAction** class will be performed.

### 3.1.6 Use Case Diagrams

The main use case diagram of the HDWE is a graphical view of some or all of the actors, use cases, and their interactions identified for the HDWE, it is a picture of the HDWE boundary (*user actor*) and the major functionality provided by the HDWE (use cases).

#### 3.1.6.1 Main Use Case Diagram

The use cases of the HDWE including:

- *Create a new html file use case*
- *Open an existing file use case*
- *Edit an opened file use case*
- *Save a file use case*
- *Save a file as use case*
- *Close a file use case*

The main use case diagram of the HDWE is as follows:

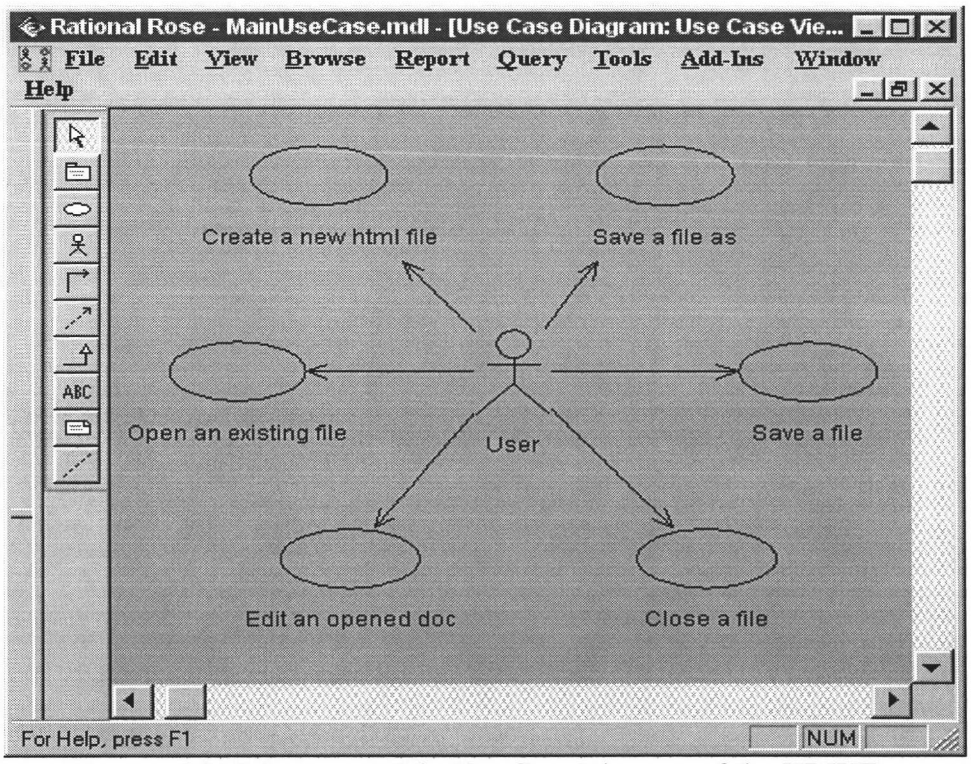

Rational Rose - MainUseCase.mdl - [Use Case Diagram: Use Case Vie...
File Edit View Browse Report Query Tools Add-Ins Window Help
Create a new html file
Save a file as
Open an existing file
User
Save a file
Edit an opened doc
Close a file
For Help, press F1
NUM

Fig.3.1 Main Use Case Diagram of the HDWE system

### 3.1.6.2 Edit an opened file use case Diagram

*Edit an opened file use case* includes sub use cases as follows:

- *select portion of doc use case*
- *layout adjustment use case*
- *cut, copy, paste use case*
- *undo, redo use case*
- *change font use case*
- *insert html element use case*

The *Edit an opened file use case* Diagram is as follows:

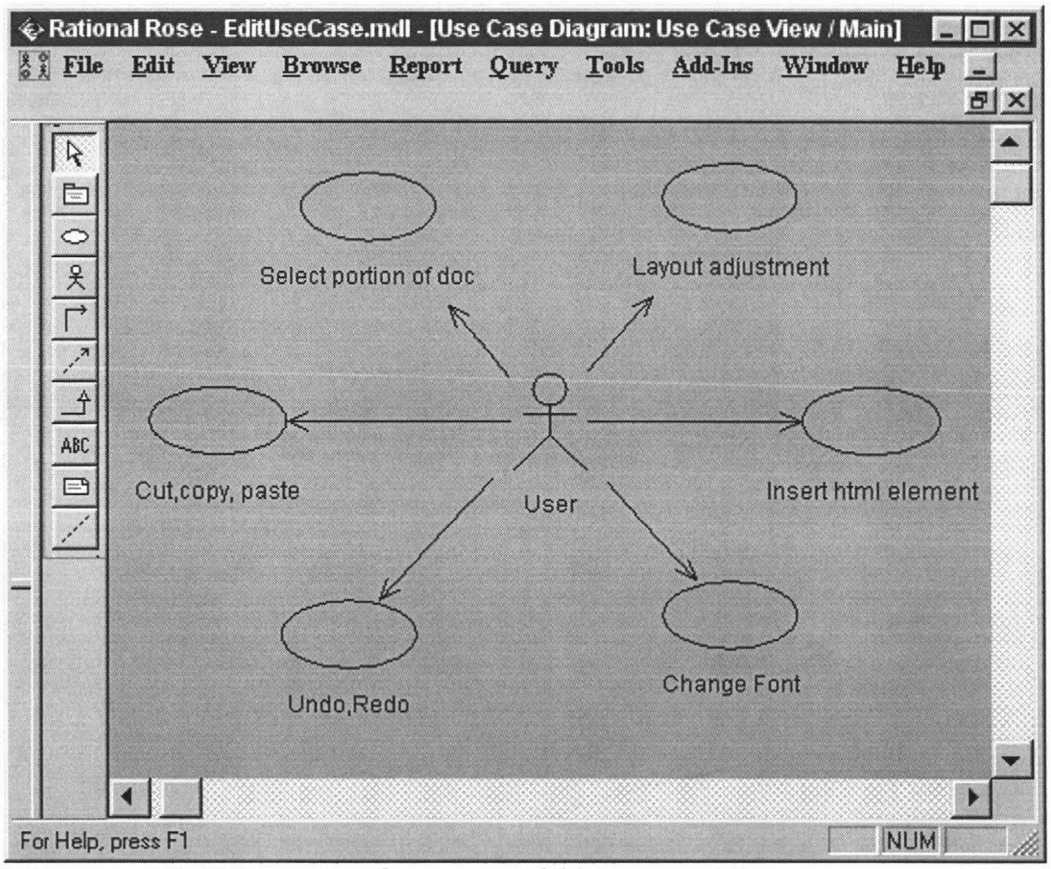

Rational Rose - EditUseCase.mdl - [Use Case Diagram: Use Case View / Main]
File Edit View Browse Report Query Tools Add-Ins Window Help
ABC
Select portion of doc
Layout adjustment
Cut,copy, paste
User
Insert html element
Undo,Redo
Change Font
For Help, press F1
NUM

Fig.3.2 *Edit an opened file use case* Diagram

### 3.1.6.3 Insert Applet use case Diagram

*Insert Applet use case* is a sub use case of *insert html element use case,* it includes sub use cases:

- *Draw primitives use case*
- *Fill primitives use case*
- *Choose pen style use case*
- *File manipulate use case*
- *Insert the drawings use case*
- *Choose rendering hints use case*
- *Choose colors, erase item use case*

The *Insert Applet Use Case* Diagram is as follows:

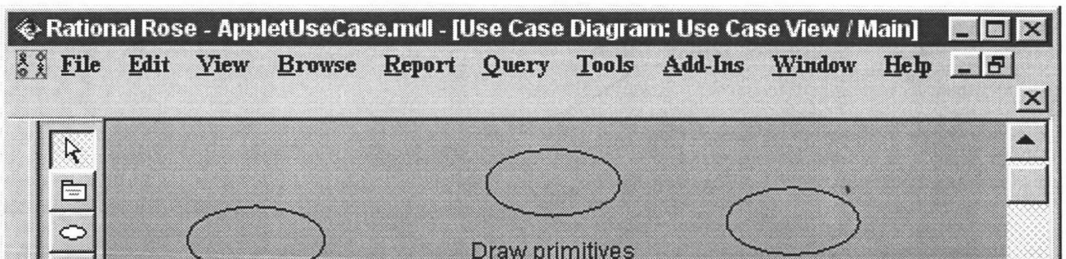


Fig.3.3 *Insert Applet Use Case* Diagram.

## 3.2 The HDWE's Architecture

In this section, a Class Diagram is created using Rational Rose to capture the structure of the classes that form the HDWE's architecture, and to show the common roles and responsibilities of the entities that provide the HDWE's behavior.

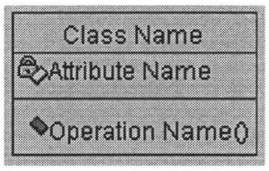
A Class diagrams contains icons representing classes, and their relationships. The next section describes the main notations and the concepts.

### 3.2.1 Notations and Concepts.

**Class**

A class captures the common structure and common behavior of a set of objects. A class is an abstraction of real-world items.

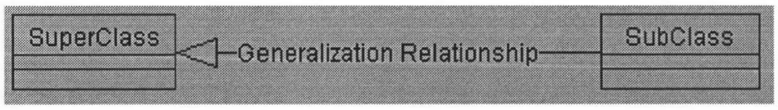
Graphical Depiction: A class icon is drawn as a 3-part box, with the class name in the top part, a list of attributes in the middle part, and a list of operations in the bottom part.



**Generalize/Inherits Relationship**

A generalize relationship between classes shows that the subclass shares the structure or behavior defined in one or more superclasses. The generalize relationship is used to show a "is-a" relationship between classes.

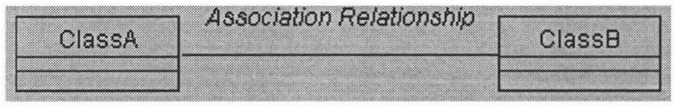
Graphical Depiction : A generalize relationship is a solid line with an arrowhead pointing to the superclass:



**Association Relationship**

An association represents a semantic connection between two classes, or between a class and an interface. Associations are bi-directional; they are the most general of all relationships and the most semantically weak. For example, an association between the **HandDrawnPanel** class and the **EditorPanel** class means that objects in the **HandDrawnPanel** class are connected to objects in the **EditorPanel** class.

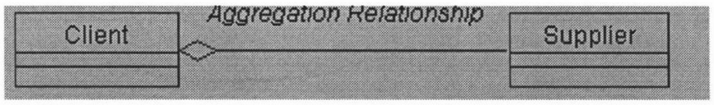
Graphical Depiction: An association relationship is an oblique or orthogonal line:



**Aggregate Relationship**

Use the aggregate relationship to show a whole and part relationship between two classes. The class at the client end of the aggregate relationship is sometimes called the aggregate class. An instance of the aggregate class is an aggregate object. The class at the supplier end of the aggregate relationship is the part whose instances are contained or owned by the aggregate object. Use the aggregate relationship to show that the aggregate object is physically constructed from other objects or that it logically contains another object. The aggregate object has ownership of its parts.

Graphical Depiction: An aggregate relationship is a solid line with a diamond at one end: The diamond end designates the client class.



### 3.2.2 HDWE's Classes and Relationships

In this section, all of the classes of the HDWE that has been designed by the author will be introduced. The classes from the JDK API, which have been heavily used in the HDWE system, for instance, the classes come from Swing text package and Java 2D package, will be discussed in section 3.3.

Class **Desktop** is a container class that used to create a multiple-document interface or a virtual desktop. HDWE creates **EditorFrame** objects and add them to the **Desktop**. A **Desktop** object uses an **EditorUICreator** object to create its user interface.

Class **EditorFrame** is a lightweight object that provides many of the features of a native frame, including dragging, closing, becoming an icon, resizing, title display, and support for a menu bar. An **EditorFrame** object is created and added to a **Desktop** object. The **EditorFrame's** contentPane is where the HDWE adds child components, such as an **EditorPanel** or a **HandDrawnPanel** object. A **HandDrawnPanel** object is associated with an **EditorPanel** object, because a **HandDrawnPanel** object can only be created by a specific **EditorPanel** object.

The class **EditorPanel** contains a **JEditorPane** which is used to handle the Html documents. An **EditorPanel** object is created and added to an **EditorFrame** object. An **EditorPanel** object is associated with a set of actions classes to provide the responce to the user's actions. The actions include : **NewAction, OpenAction, PrintAction, ReloadAction, SaveAction, SaveAsAction, ViewSourceAction, CloseAction, CloseAllAction, CloseAllWithSaveAllAction, CloseAllWithoutSaveAllAction, UndoHandler, AppletAction,** and **ExitAction.**

A **HandDrawnPanel** object contains a **DrawPanel** object which is used to handle the HDP drawing documents. A **HandDrawnPanel** object uses a **DrawPanelUICreator** class to create its own user interface. It also cantains a **ColorPalette** class to provide a user with a user interface to control the foreground and background of the draw panel of the HDP. A **HandDrawnPanel** object is created and added to an **EditorFrame** object.

Both **EditorUICreator** and **DrawPanelUICreator** classes are subclasses of the **UICreator** class, which is a general class for creating user interface for an application.

A **ColorPalette** object consists of a **BasicColorPanel** object which contains a set of **ColorButton** objects, each of which represents a basic color. A **ColorPalette** also has an other button which is used to provide the user with an interface to choose a color that is not a basic color.

The **DrawPanel** class is one of the most importance classes in HDP. It provides the user a canvas to edit an HDP drawing object. Such as hand drawn curves, lines, etc. An items (a member variable of the **DrawPanel** object, which is a vector) contains the content (represented by the **Item** class) of the **DrawPanel**.

One of shapes implemented by HDWE is the **Curve** class, which includes **GeneralPath** class as a member variable. The purpose of the **Curve** class is to extend the **GeneralPath** class, so that the user can get and set the points of the curve.

HDWE also provides an **AppletReader** class to read the hand drawn document. This is the class that will be sent to the browser with the hand drawn file, so that the browser can display the viewer the drawing applet using **AppletReader** class.

## 3.2.3. Class Diagram

A class diagram has been drawn using Rational Rose for HDWE as in Fig.3.8. This Class Diagram shows the classes and their relationships in the logical design of the HDWE system. The class diagram represents the entire class structure of the HDWE system.

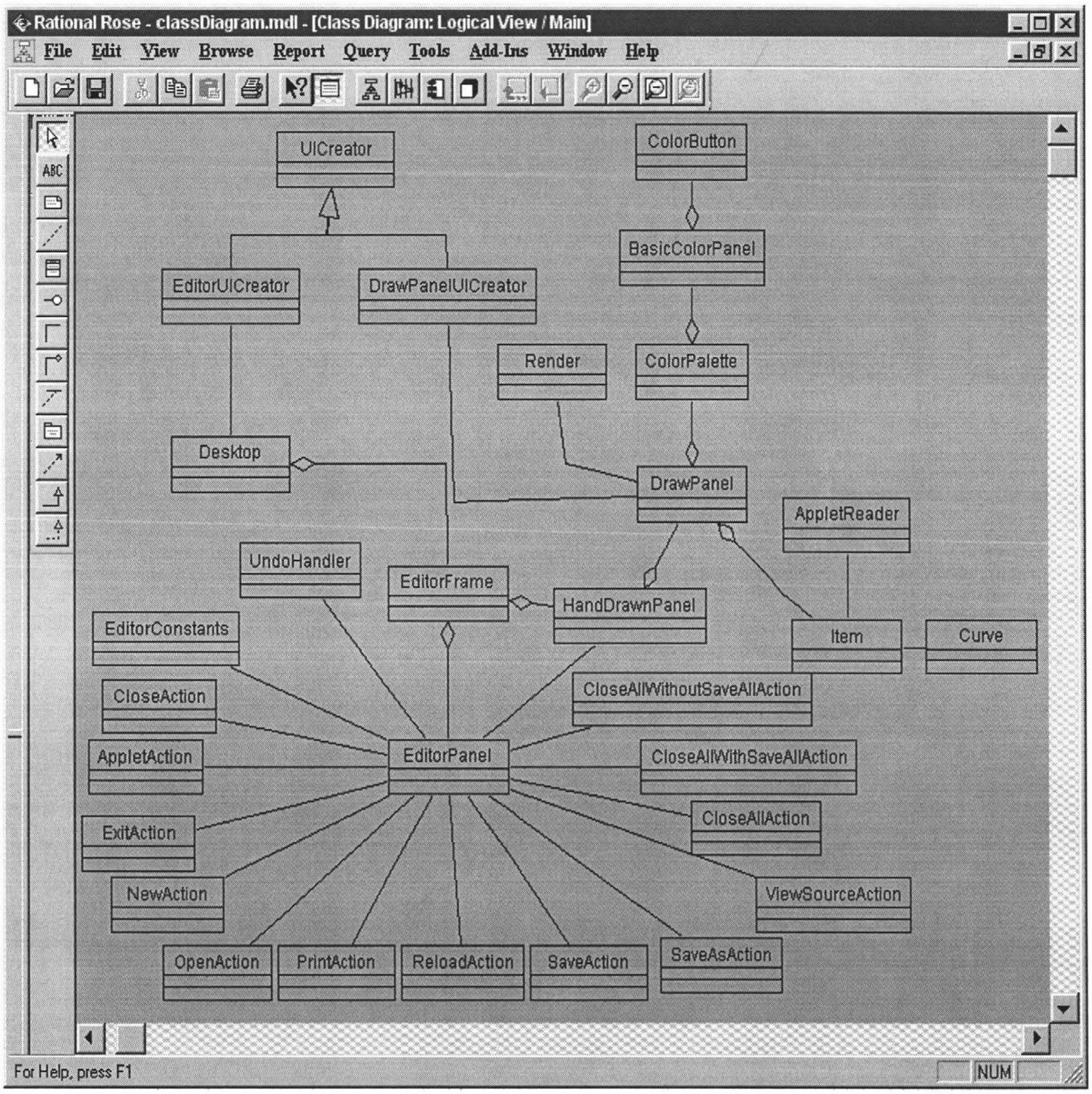


Fig.3.4 Class Diagram of HDWE.

## 3.3 The Html and Java 2D Package

Both the Javax.swing.text.html package and the Java 2D package of Java APIs have been heavily used in developing HDWE. This section describes important features built into these two packages.

### 3.3.1 Swing Text Package

#### 3.3.1.1 The Swing text-component hierarchy

The Swing text package includes a comprehensive set of text-related classes to support various text related activities. These classes get combined into Swing's text components, with the pieces replaceable with alternative implementations.

The text classes provided in the Swing package subclass from a root class named JTextComponent. Fig.3.5 shows the Swing text-class hierarchy.

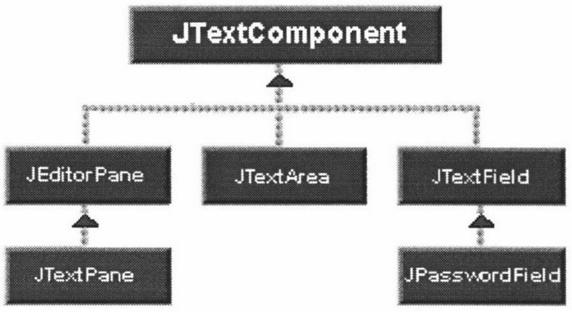


Fig.3.5 Swing Text-Class Hierarchy.

The JEditorPane component is the extensible component that provides a convenient way to morph into different kinds of support. JEditorPane provides a registration mechanism that can easily extend the set of text types that it knows how to deal with. By default, the JEditorPane registration mechanism provides some level of support for plain text, HTML, and RTF.

*Delegation of functionality*

Swing's text components delegate most all of their duties to other objects. Consequently, developers usually can not directly extend a text component in order to change its behavior. Instead, developers must change the objects that are being delegated to.

### 3.3.1.2 Text Commands

In the Swing text API, the text components export most of their capabilities as commands. Commands are implementations of the Swing Action interface. Commands are typically bound to keyboard actions, and can also be bound to other UI mechanisms such as toolbar buttons, menu items, and the mouse.

The Swing text package implements text commands as subclasses of the TextAction class. When an application executes a text-related command, the TextAction class tries to find a text component to operate on by examining the value returned by a method named ActionEvent.getSource(). If this value is an instance of a JTextComponent, the component becomes the target of the operation. The action may also attempt to decode additional information needed from the command string associated with the ActionEvent.

If this attempt fails, the TextAction class tries to target the currently (or most recently) focused component. If that attempt also fails, no action is taken.

### 3.3.1.3 Undo-redo support

Swing's text components do not directly support undo-redo capabilities as that would tie them to a specific undo-redo policy. But they do support undo/redo indirectly by supplying their support in Swing's text-component model. Because the view portion of a text component reflects the state of the component's model, changing the model with undo/redo causes the views to follow automatically.

*Documents and undo-redo actions*

An object called a Document, defined in Swing's Document interface, is the model for text components. A document can be shared by multiple components. A document communicates with its views via a DocumentEvent, as shown in the following diagram. The illustration shows how a document might communicate with two differentJTextComponent objects, each of which contains a view.

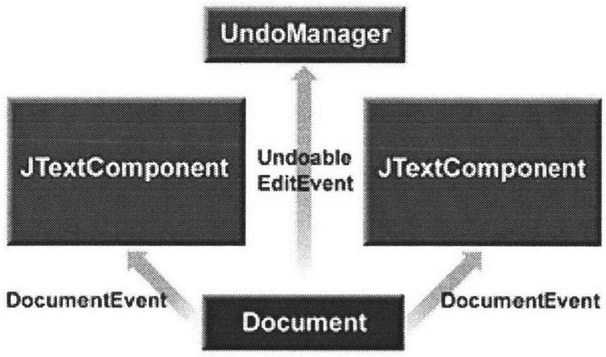


Fig.3.6 A document communivates with its views via a documentEvent.

Referring to this diagram, suppose that the component shown on the left mutates the document object. The document responds by dispatching a DocumentEvent to both component views and sends an UndoableEditEvent to the listening logic, which maintains a history buffer. If the history buffer rolls back, another DocumentEvent is sent to both views, causing them to reflect the undone mutation to the document -- that is, the removal of the left component's mutation.

#### 3.3.1.4 PL&F

Swing's text components -- like all other Swing components -- have a PL&F. But it is more difficult to build PL&F capabilities into text components. The problem is that there is a certain tension between the content represented inside a PL&F text component and the user interface that determines the component's overall look and feel. The user interface is likely to dictate some elements of the component's behavior and appearance, such as its set of key bindings (largely a matter of feel), its set of colors, its interaction-related behavior when it is selected, and so on. These features and functionality are somewhat orthogonal to the look of the component, which is largely determined by the kind of content that it represents.

The standard UI delegate used by other kinds of Swing components is not sufficient. A text component needs a second delegate to handle the kind of content that it displays.

*EditorKit*

In the Swing text-component package, this second delegate is an implementation of a class named EditorKit, as illustrated in the following diagram:

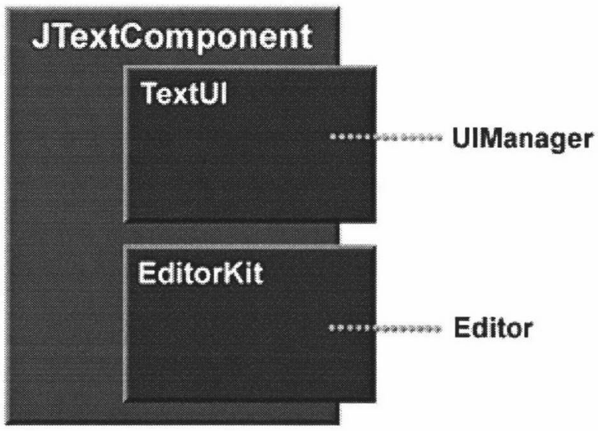


Fig.3.7 JTextComponent delegate EditorKit to handle the Content.

The EditorKit, is the primary mechanism for extending Swing text components. Architecturally, the EditorKit is a bundling of the capabilities needed by a text component for dealing with a particular type of content.

The EditorKit has the following responsibilities:

1. It creates a model (see next heading). Because Swing's EditorKit implementation knows what kind of content it is describing, it can provide a model that has a particular set of policies for describing that kind of content.
2. It provides a factory for building views. Rather than providing a view directly, the EditorKit provides a way of creating views.
3. It provides commands for editing the content. Because the EditorKit encapsulates functionality that knows how to deal with a particular content type, it knows what kinds of operations can be performed on that particular content type.
4. It saves data to a stream. Because different kinds of content can have their own individual storage formats, the EditorKit is an ideal mechanism for streaming data.

5. It reads from a stream. Again, because different kinds of content can have their own individual storage formats, the EditorKit is an ideal mechanism for streaming data.

_The TextUI object_

The UIManager used by a text component is called a TextUI object. When a TextUI object is installed into a JTextComponent, its duties are to make sure that a default set of properties for the type of UI has been plugged into the JTextComponent. The duties of a TextUI object consist of the following:

1. It sets the various colors used in a text component, such as the caret color, the selection background color, the selection foreground color, and the like.
2. It installs a Caret implementation. In a text component, the caret is an object used for navigating through the text component. It is responsible for the component's selection policy.
3. It installs a suitable set of key bindings (a Keymap) for the UI.
4. It builds a view of the model. In simple components, this task can be performed directly. In more complex components it can be delegated to an EditorKit.

#### 3.3.1.5 The Text-Component Model

At the simplest level, text can be modeled as a linear sequence of characters. To support internationalization, the Swing text model uses unicode characters. The sequence of characters displayed in a text component is generally referred to as the component's content. The interface for the object that holds the content is the Document interface. When a change is made to the content, the component's view (or views) needs to be notified of the change.

Because the content of a text component can be quite large and cannot be efficiently deduced, this notification must include fairly detailed information about whatever change has taken place. The interface that Swing uses to describe text-change information is called a DocumentEvent.

*The coordinate system*

To convey information about changes to a text view (or text views) in a proper manner, a document must have a coordinate system, as shown in the following diagram:

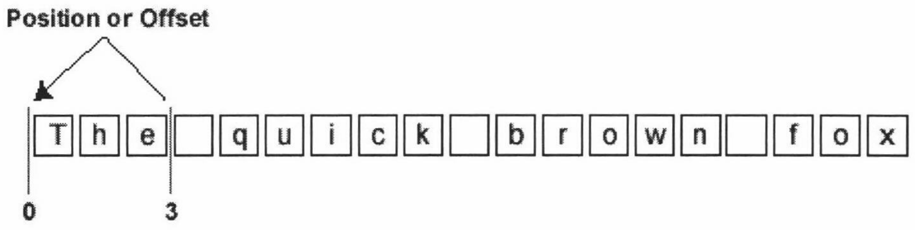


Fig.3.8 Coordinate System of a Document

As the diagram shows, a location in a text document can be referred to as a position, or an offset. This position is zero-based. It can be used to specify the character's position between any other two characters.

*The Element interface*

Structure imposed upon the content is typically associated with some additional information. To model the structure in this way, Swing uses an object called an Element to represent a fragment of structure.

As noted earlier, a Document is a text container that supports editing and provides notification of changes. In other words, it serves as the model in a MVC relationship. Support is provided to mark up the text in a document with a structure that tracks changes. The unit of structure used in this construct is called an Element.

In Swing text components, views are typically built from an element structure. An arbitrary set of attributes can be associated with an element. The interface itself is intended to be free of any policy for structure that is provided, as the nature of the document structure should be determined by the implementation.

*The AbstractDocument*

The Swing text package provides an abstract implementation of the Document interface. This implementation is called the AbstractDocument.

A unique feature of the AbstractDocument is that the text-content storage is separate from the structural modeling of the document. This feature allows a developer to establish how a document is stored independently of what the document is modeling. For example, an application can create a model structure designed to support an HTML document and then independently plug in storage mechanisms based upon different algorithms.

These text content storage plug-ins must implement the Content (AbstractDocument.Content) interface. This interface requires that the following three kinds of behavior be provided:

1. General editing features: text insertion, text removal, and text retrieval.
2. Support for marks that represent a position between characters in storage, that tracks changes made to the text storage. These are provided as an implementation of the Position interface.
3. Provide objects that represent the changes made to the storage so that they can efficiently participate in undo/redo operations. These are provided as an implementation of the UndoableEdit interface.

#### 3.3.1.6 The View Class

A very important part of the text package is the View class. As its name suggests, the View class represents a view of the text model. It is this class that is responsible for the look of the text component.

By default, a view is very light. It contains a reference to the parent view from which it can fetch many things without holding state, and it contains a reference to a portion of the model (the Element). A view does not have to exactly represent an element in the model; that is simply a typical, and therefore convenient, mapping. Alternatively, a view can maintain a couple of Position objects to keep track of its location in the model. This is typically the result of formatting where views have been broken down into pieces.

The fact that views are closely related to Elements makes it easier to use factories to produce views. For instance, the inner class HTMLFactory of the HTMLEditorKit class creates views from elements as follows:

| **Elements** | **Views** |
|---|---|
| HTML.Tag.CONTENT | InlineView |
| HTML.Tag.P | javax.swing.text.html.ParagraphView |
| HTML.Tag.H1 | javax.swing.text.html.ParagraphView |
| HTML.Tag.UL | ListView |
| HTML.Tag.BODY | BlockView |
| HTML.Tag.IMG | ImageView |
| HTML.Tag.HR | HRuleView |
| HTML.Tag.BR | BRView |
| HTML.Tag.TABLE | javax.swing.text.html.TableView |
| HTML.Tag.FRAME | FrameView |

This is intended to enable customization of how views get mapped over a document model. In turn, using factories to produce views makes it easier to keep track of the various parts of views that must be changed when their corresponding models change.

A view has the following responsibilities:

1. It participates in layout.
2. It renders a portion of the model.
3. It translates between the model and view coordinate systems.
4. It responds to changes from the model.

In HDWE, JEditorPane text component has been used to edit and browse Html documents. The JEditorPane has exported its capabilities as commands. These commands have been bound to menus, tool buttons of HDWE. JEditorPane has delegated its content handling capabilities to an EditorKit class, in HDWE, this is HTMLEditorKit class. It creates a model - HTMLDocument, which provides general editing features, such as text insertion, removal and retrieval, which also supports for marks, and undo/redo operations. HTMLEditorKit also provides a factory - HTMLFactory class for building views, commands for editing the content. It also saves and reads data to or from a stream. The default TextUI class sets various colors used in HDWE, installs a Caret, and installs a suitable set of key bindings.

## 3.3.2 Java 2D

### 3.3.2.1 Overview of the Java 2D API

The Java 2D API provides enhanced two-dimensional graphics, text, and imaging capabilities for Java programs through extensions to the Abstract Windowing Toolkit (AWT). This comprehensive rendering package supports line art, text, and images in a flexible, full-featured framework for developing richer user interfaces, sophisticated drawing programs and image editors.

The Java 2D API provides

- A uniform rendering model for display devices and printers
- A wide range of geometric primitives, such as curves, rectangles, and ellipses and a mechanism for rendering virtually any geometric shape
- Mechanisms for performing hit detection on shapes, text, and images
- A compositing model that provides control over how overlapping objects are rendered
- Enhanced color support that facilitates color management
- Support for printing complex documents

*Java 2D Rendering*

The basic rendering mechanism is that the drawing system controls when and how programs can draw. When a component needs to be displayed, its paint or update method is automatically invoked with an appropriate Graphics context.

The Java 2D API introduces java.awt.Graphics2D, a new type of Graphics object. Graphics2D extends the Graphics class to provide access to the enhanced graphics and rendering features of the Java 2D API.

*Graphics2D Rendering Context*

The collection of state attributes associated with a Graphics2D object is referred to as the Graphics2D rendering context. To display text, shapes, or images, a user can set up the Graphics2D rendering context and then call one of the Graphics2D rendering methods, such as draw or fill. The Graphics2D rendering context contains several attributes as follows:

- The pen style that is applied to the outline of a shape. This stroke attribute enables user to draw lines with any point size and dashing pattern and to apply end-cap and join decorations to a line.
- The fill style that is applied to a shape's interior. This paint attribute enables one to fill shapes with solid colors, gradients, and patterns.
- The compositing style that is used when rendered objects overlap existing objects.
- The transform that is applied during rendering to convert the rendered object from user space to device-space coordinates. Optional translation, rotation, scaling, or shearing transforms can also be applied through this attribute.
- The clip, which restricts rendering to the area within the outline of the Shape used to define the clipping path. Any Shape can be used to define the clip.
- The font used to convert text strings to glyphs.
- Rendering hints that specify preferences in the trade-offs between speed and quality. For example, one can specify whether antialiasing should be used, if it's available.

*Graphics2D Rendering Methods*

The following are general rendering methods that can be used to draw any geometry primitive, text, or image:

- draw--renders the outline of any geometry primitive, using the stroke and paint attributes.
- fill--renders any geometry primitive by filling its interior with the color or pattern specified by the paint attribute.
- drawString--renders any text string. The font attribute is used to convert the string to glyphs, which are then filled with the color or pattern specified by the paint attribute.
- drawImage--renders the specified image.

In addition, Graphics2D supports the Graphics rendering methods for particular shapes, such as drawOval and fillRect.

#### 3.3.2.2 Coordinate Systems

The Java 2D system maintains two coordinate spaces.

- User space is the space in which graphics primitives are specified.
- Device space is the coordinate system of an output device, such as a screen, window, or a printer.

User space is a device-independent logical coordinate system: the coordinate space that your program uses. All geometries passed into Java 2D rendering routines are specified in user-space coordinates.

When the default transformation from user space to device space is used, the origin of user space is the upper-left corner of the component's drawing area. The x coordinate increases to the right, and the y coordinate increases downward.

Device space is a device-dependent coordinate system that varies according to the target rendering device. Although the coordinate system for a window or the screen might be very different from that of a printer, these differences are invisible to Java programs. The necessary conversions between user space and device space are performed automatically during rendering.

#### 3.3.2.3 Shapes

The classes in the java.awt.geom package define common graphics primitives, such as points, lines, curves, arcs, rectangles, and ellipses.

With these classes one can create virtually any geometric shape and render it through Graphics2D by calling the draw method or the fill method.

*Rectangular Shapes*

The Rectangle2D, RoundRectangle2D, Arc2D, and Ellipse2D primitives are all derived from RectangularShape, which defines methods for Shape objects that can be described by a rectangular bounding box. The geometry of a RectangularShape can be extrapolated from a rectangle that completely encloses the outline of the Shape.

*GeneralPath*

The GeneralPath class enables user to construct an arbitrary shape by specifying a series of positions along the shape's boundary. These positions can be connected by line segments, quadratic curves, or cubic (Bézier) curves.

*Areas*

With the Area class a user can perform boolean operations, such as union, intersection, and subtraction, on any two Shape objects. This technique, often referred to as constructive area geometry, enables the user to quickly create complex Shape objects without having to describe each line segment or curve.

#### 3.3.2.4 Text

When a developer needs to display text, he can use one of the text-oriented components, such as the Swing label or text components. When the developer uses a text component, a lot of the work is done for him--for example, JTextComponent objects provide built-in support for hit testing and displaying international text.

If the developer just wants to draw a static text string, he can render it directly through Graphics2D by using the drawString method. To specify the font, one can use the Graphics2D setFont method.

*Text Layout*

Before text can be displayed, it must be laid out so that the characters are represented by the appropriate glyphs in the proper positions. If the developer is using Swing, he can let JLabel or JTextComponent manage text layout for him. JTextComponent supports bidirectional text and is designed to handle the needs of most international applications. On the other hand, if he is not using a Swing text component to automatically display text, he can use one of two Java 2D mechanisms for managing text layout.

#### 3.3.2.5 Images

The Java 2D API implements a new imaging model that supports the manipulation of fixed-resolution images stored in memory. A new Image class in the java.awt.image package, BufferedImage, can be used to hold and to manipulate image data retrieved from a file or a URL. For example, a BufferedImage can be used to implement double buffering--the graphic elements are rendered off-screen to the BufferedImage and are then copied to the screen through a call to Graphics2D drawImage. The classes BufferedImage and BufferedImageOp also enable you to perform a variety of image-filtering operations, such as blur and sharpen. The producer/consumer imaging model provided in previous versions of the JDK is supported for backward compatibility.

In HDP, Graphics2D has been used to access to the enhanced graphics and rendering features of the Java 2D API. The HDP provides a user with a user interface to set the rendering context, such as pen styles, colors, and rendering hints. HDP has also used two kinds of rendering method, they are draw - which renders the outline of any geometry primitive, and fill - which renders any geometry primitive by filling its interior with the color or pattern. HDP has also used some common graphics primitives defined in java.awt.geom package, such as, lines, curves, rectangles, and ellipses.

# Chapter 4

# Implementation and Result

## 4.1 Problems Encountered With the Solutions

During implementing the HDWE system, a number of problems have been encountered, some of these problems and their solutions are discussed in this chapter.

### 4.1.1 Internationalization

**Internationalization Problem**

The HDWE needs to display its messages, menu labels, or tool tips in an appropriate language for people from different countries. Unfortunately the developer of the HDWE is not multilingual, so translators who can translate these messages into all kinds of different language are needed. Since the translators are not programmers, the developer has to move the messages, labels and tips out of the source code and into text files so that the translators can edit them. The HDWE must be flexible enough so that it can display the messages in other languages. Therefore, the HDWE needs to be internationalized.

**Internationalization Concepts**

Internationalization is the process of designing an application so that it can be adapted to various languages and regions without engineering changes.

An internationalized program has the following characteristics:

- With the addition of localized data, the same executable can run worldwide.
- Textual elements, such as status messages and the GUI component labels, are not hard-coded in the program. Instead they are stored outside the source code and retrieved dynamically.
- Support for new languages does not require recompilation.

- Culturally-dependent data, such as dates and currencies, appear in formats that conform to the end user's region and language.
- It can be localized quickly.

Localization is the process of adapting software for a specific region or language by adding locale-specific components and translating text. Usually, the most time-consuming portion of the localization phase is the translation of text. Other types of data, such as sounds and images, may require localization if they are culturally sensitive. Localizers also verify that the formatting of dates, numbers, and currencies conforms to local requirements.

## **Internationalization Procedures**

Because the messages are no longer hard-coded and because the language code is specified at run time, the same executable can be distributed worldwide. No recompilation is required for localization. The procedures to internationalize the HDWE is as follows:

### *1. Create the Properties Files*

A properties file stores the translatable text of the messages to be displayed, it can be created with any text editor. The default properties file is the English version. Now that the messages are in the properties file, they can be translated into various languages. No changes to the source code are required. The French translator may create a French version of the properties file which contains the same items as the English version, but where the values associated with each of the keys have been changed. The keys themselves must not be changed, because they will be referenced when HDWE fetches the translated text.

### *2. Define the Locale*

The Locale object identifies a particular language and country. It represents a specific geographical, political, or cultural region. An operation that requires a Locale to perform its task is called locale-sensitive and uses the Locale to tailor information for the user. For example, displaying a number is a locale-sensitive operation--the number should be formatted according to the customs or conventions of the user's native country, region, or culture. The following statement defines a Locale for which the language is English and the country is the United States:

```
        aLocale = new Locale("en","US");
```

Locale objects are only identifiers. After defining a Locale, it can be passed to other objects that perform useful tasks, such as formatting dates and numbers. These objects are locale-sensitive because their behavior varies according to Locale. A ResourceBundle is an example of a locale-sensitive object.

*3. Create a ResourceBundle*

Resource bundles contain locale-specific objects. When a program needs a locale-specific resource, a String for example, that program can load it from the resource bundle that is appropriate for the current user's locale. In this way, a developer can write program code that is largely independent of the user's locale isolating most of the locale-specific information in resource bundles. The ResourceBundle in HDWE is created as follows:

```
Private static ResourceBundle resources;
static
   { try
      { resources = ResourceBundle.getBundle("Editor",Locale.getDefault());
      }
    catch (MissingResourceException mre)
     { System.err.println("Editor.properties not found");
       System.exit(0);
     }
}
```

The arguments passed to the getBundle method identify which properties file will be accessed. The first argument, Editor, refers to the Editor.properties file in HDWE system.

The Locale, which is the second argument of getBundle, specifies which of the Editor files is chosen. When the Locale was created, the language code and the country code were passed to its constructor.

Now all the developer has to do is get the translated messages from the ResourceBundle.

*4. Fetch the Text from the ResourceBundle*

The properties files contain key-value pairs. The values consist of the translated text that the program will display. The developer specifies the keys when fetching the translated messages from the ResourceBundle with the *getString* method. For example, to retrieve the objects identified by the String nm key, the developer invokes *getResourceString* method, which is as follows:

```
public String getResourceString(String nm)
  {  String str = null;
     try
     {      str = resources.getString(nm);
     }
     catch (MissingResourceException mre)
     {  system.out.println("MissingResourceException"+mre.);
     }
     return str;
  }
```

The above source code uses the key because it reflects the content of the message. The key is hard-coded in the program and it must be present in the properties files.

## 4.1.2 UICreator Class

**Problem**

Both the Desktop class and the HandDrawnPanel class of the HDWE need to create a user interface. They both need methods to create the menus, menu items, tool bars, tool buttons, and also need a member of variables to store the name of menu items, actions, and commands. A class that generalizes these common methods and attributes should be implemented to create a common user interface for all kinds of applications. As a result, a general class - **UICreator** has been implemented in HDWE.

**Solution**

When a developer constructs a new **UICreator** object, he needs to provide a **ResourceBundle** which is the locolazation resource for the HDWE, it cantains all menu item, toolbar button, and tool tips keys and values. The HDWE can fetch each of them and create the corresponding menu items and buttons. The developer also needs to provide the actions that are supported by the HDWE and HDP. The Actions supported by the HDWE are:

```
private Action[] getActions()
  { Action[] defaultActions =
      { new NewAction(this),
        new OpenAction(this),
        new SaveAction(this),
        new SaveAsAction(this),
        new CloseAction(this),
        new ReloadAction(this),
        new PrintAction(this),
        new ViewSourceAction(this),
        new AppletAction(this),
        new ExitAction(this),
        new CloseAllAction(this),
        new CloseAllWithoutSaveAllAction(this),
        new CloseAllWithSaveAllAction(this),
        undoHandler.new UndoAction(),
        UndoHandler.new RedoAction()
      };
    HTMLEditorKit htmlKit = new HTMLEditorKit();
    return TextAction.augmentList(htmlKit.getActions(), defaultActions);
    }
```

The Actions supported by the HDP are:

```
private Action[] defaultActions =
{ new NewDrawingAction(),
  new OpenDrawingAction(),
  new SaveDrawingAction(),
  new SaveDrawingAsAction(),
  new CloseDrawingAction(),
  new PrintDrawingAction(),
  new InsertAction(),
  new ExitDrawingAction(),
  new UndoAction,
  new RedoAction
  new LineAction(),
  new CurveAction(),
  new EraserAction(),
  new CircleAction(),
        new FilledCircleAction(),
        new EllipseAction(),
        new FilledEllipseAction(),
        new SquareAction(),
        new FilledSquareAction(),
        new RectangleAction(),
        new FilledRectangleAction(),
      };
```

The main methods included in this class are as follows:

- *CreateMenubar*

  Creates the menubar for the application. By default this pulls the definition of the menu from the associated resource file.

- *CreateMenu*

  Creates a menu for the application. By default this pulls the definition of the menu from the associated resource file.

- *CreateMenuItem*

  This is the hook through which all menu items are created. It registers the result with the menu item hashtable so that it can be fetched with *getMenuItem*().

- *CreateSubMenu*

  Creates subMenu for the application.

- *CreateToolbar*

  Creates the toolbar for the application. By default this reads the resource file for the definition of the toolbar.

- *CreateToolbarButton*

  Creates a button to go inside of the toolbar. This is the hook through which every toolbar item is created. By default this will load an image resource. In HDWE system, the image file is located in the images\ directory. Images\ directory is in the same directory as the **UICreator** class.

- *GetAction*

  Fetches the list of actions supported by the HDWE or HDP. This method is implemented to return the list of actions supported by the embedded **JEditorPane** or **HandDrawnPanel** augmented with the actions defined locally.

- *GetResource*

  Fetches the directory of the class.

```
protected URL getResource(String key)
{  String name = getResourceString(key);
   if (name != null)
   {  URL url = this.getClass().getResource(name);
      return url;
   }
   return null;
}
```

- *GetResourceString*

  Fetch the locale value of a specific key item.

- *Tokenize*

  Take the given string and chop it up into a series of strings on white space boundaries. This is useful for trying to get an array of strings out of the resource file.

The **UICreator** class is a general class, as it can be used by any application to create it's own user interface.

In the HDWE system, both the **Desktop** class and **HandDrawnPanel** class need extra methods to create their own specific user interface, as a result, the **UICreator** class has been extended to a **EditorUICreator** class and a **DrawpanelUICreator** class.

### 4.1.3 Serialization

**Problem**

The Serializability of a class is enabled by the class that implementing the **java.io.Serializable** interface. The serialization interface has no methods or fields and serves only to identify the semantics of being serializable. Classes that do not implement this interface will not have any of their state serialized or deserialized. All subtypes of a serializable class are themselves serializable.

In HDWE, after a user finishes drawing on the HDP, the drawing object needs to be saved on the hard drive, so that it can be retrieved at a later stage, or can be transmitted to the remote web browser to be displayed as an applet. This means the drawing objects on the HDP need be serialized. These drawing objects are lines, curves, and primitives, all of them are implemented using the Shape interface, which provides definitions for objects that represent some form of geometric shape. The Stroke interface allows a Graphics2D object to trace a shape's outline with a marking pen of the appropriate size and shape. The Paint interface defines how color patterns can be generated for Graphics2D operations. The RenderingHints class contains rendering hints that can be used by the Graphics2D class.

The problem comes from the fact that all these classes, that is, Shape, Stroke, Paint, and the RenderingHints, can not be serialized.

**Constriction**

Java API provides a mechanism to solve this problem, which is to subclass the non-serializable classes to be serialized. The precondition for this mechanism is that, the subtype may assume responsibility for saving and restoring the state of the supertype's public, protected, and package fields. The subtype may assume this responsibility only if the class it extends has an accessible no-argument constructor to initialize the class's state. It is an error to declare a class Serializable if this is not the case. During deserialization, the fields of non-serializable classes will be initialized using the public or protected no-argument constructor of the class. A no-argument constructor must be accessible to the subclass that is serializable. The fields of serializable subclasses will be restored from the stream.

However, most of Java 2D classes do not have an accessible no-argument constructor. As a result, these Java 2D classes can not be subclassed in order to implement serializattion.

**Solution**

In HDWE, an **Item** object is used to store the information about an item on the draw panel. An item of the draw panel is an object that has been drawn on the draw panel, it starts from the point the mouse is pressed until the point where the mouse is released. The information included in the **Item** class is a shape, rendering context, the coordinate of the shape, background and foreground color of the shape.

However, all of the classes above listed are not implemented in the Serialization interface. We need to replace these classes with the classes that can be serialized so that the state of the items on the draw panel can be stored permanently on hard driver and can be restored at a later date.

In order to make the **Item** class serializable, all non-serializable member variables have been replaced by the serializable objects as follows:

1. Shape has been replaced by a variable of type **integer**, this is encoded to represent:

| | | |
|---|---|---|
| 1 | : | LINES |
| 2 | : | CURVES |
| 3 | : | ELLIPSES(including circle) |
| 4 | : | RECTANGLES(including Square) |

2. A **Vector** class has been used to store the points for each shape. This is because the Vector class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items

after the Vector has been created. And most of all, the Vector class is serializable.

3. Stroke has been replaced by:

```
Float        width;
Int          cap, join;
```

4. Paint has been replaced by:

```
int          red, green, blue;
```

5. RenderingHints has been replaced by:

```
int          anti, rend, dith;
```

Two methods are used to write and read a class's state: The *writeObject* method is responsible for writing the state of the object for its particular class so that the corresponding and the *readObject* method is responsible for reading from the stream and restoring the class fields.

This resolved the serialization problem of the **Item** class of HDWE.

### 4.1.4 Content type of the JEditorPane

**Problem**

The **JEditorPane** is a text component for editing various kinds of content. The following content types can be manipulated by **JEditorPane**:

- text/plain

  Plain text, which is the default the type when the type given is not recognized. The kit used in this case is an extension of DefaultEditorKit that produces a wrapped plain text view.
- text/html

  HTML text. The kit used in this case is the class javax.swing.text.html.HTMLEditorKit which provides html 3.2 support.

- text/rtf

  RTF text. The kit used in this case is the class javax.swing.text.rtf.RTFEditorKit which provides a limited support for the Rich Text Format.

The **JEditorPane** class uses implementations of the **EditorKit** to accomplish its behavior. It needs to effectively morph into the proper kind of text editor for the kind of content it is given. The content type that editor is bound to at any given time is determined by the **EditorKit** currently installed. If the content is set to a new URL, its type is needed to determine the **EditorKit** that should be used to load the content.

**Solution**

There are several ways to load content into **JEditorPane**.

1. The *setText* method can be used to initialize the component from a string. In this case the current **EditorKit** will be used, and the content type will be expected to be of this type.
2. The *read* method can be used to initialize the component from a **Reader**. If the content type is html, relative references (e.g. for things like images) can not be resolved unless the <base> tag is used or the Base property on **HTMLDocument** is set. In this case the current **EditorKit** will be used, and the content type will be expected to be of this type.
3. The *setPage* method can be used to initialize the component from a URL. In this case, the content type will be determined from the URL, and the registered **EditorKit** for that content type will be set.

In HDWE, the *setPage* method has been used to load the document. At first, the *getContentType* method is invoked to get content type of the file that is to be opened, and then set the appropriate content type for the **JEditorPane**.

In HDWE, the parameter URL of the *getContentType* method is the full path directory name of the file to be opened. This method returns the string that is the name of the contentType. In the following source code, *url.openConnection()* returns a URLConnection object that represents a connection to the remote object referred to by the *url,* and the *getContentType()* method returns the value of the content-type header field. The value of the string might be one of: "text / plain", " text / html ", " text / rft ". The source code is as follows:

```
public String getContentType(URL url)
{     String contentType = null;

            try
            {       contentType = url.openConnection().getContentType();
            }
            catch (IOException ioe)
            {       System.out.println("Can not open the connection" +
                    ioe.getMessage());
            }

            return contentType;
      }
```

After getting the content type of the URL, the *setContentType* method of **JEditorPane** can be used to set the correct content type for the **JEditorPane**. It performs this task by two steps as follows:

1. calls *getEditorKitForContentType*

   *getEditorKitForContentType* method fetches the editor kit to use for the given type of content. This is called when a type is requested that does not match the currently installed type. If the component does not have an EditorKit registered for the given type, it will try to create an EditorKit from the default EditorKit registry. If that fails, a **PlainEditorKit** is used on the assumption that all text documents can be represented as plain text.

2. calls *setEditorKit* if an editor kit can be successfully located.

   *SetEditorKit s*ets the currently installed kit for handling content, it establishes the content type of the editor. Any old kit is first deinstalled, then if kit is non-null, the new kit is installed, and a default document created for it.

In this way, the **JEditorKit** can set the appropriate content type at run time.

### 4.1.5 Applet Insertion

**Problem**

When a user edits an Html document, he can insert a drawing object as an applet into that Html document. The common syntax for the APPLET tag is as follows:

```
  <APPLET
        CODEBASE = codebaseURL
        ARCHIVE = archiveList
        CODE = appletFile ...or... OBJECT = serializedApplet
        ALT = alternateText
        WIDTH = pixels  HEIGHT = pixels>
  <PARAM NAME = appletAttribute1 VALUE = value>
</APPLET>
```

The mandatory attributes are CODE, WIDTH, and HEIGHT. Some of these attributes are described as follows:

*CODE = applet File*

This REQUIRED attribute gives the name of the file that contains the applet's compiled Applet subclass.

*ALT = alternate text*

This OPTIONAL attribute specifies any text that should be displayed if the browser understands the APPLET tag but can not run Java applets.

*WIDTH = pixels, HEIGHT = pixels*

These REQUIRED attributes give the initial width and height (in pixels) of the applet display area, not counting any windows or dialogs that the applet brings up.

*<PARAM NAME = appletAttribute1 VALUE = value> . . .*

This tag is the only way to specify an applet-specific attribute. Applets access their attributes with the *getParameter()* method.

The HDP should be able to set the right values for these attributes of the Applet tag, and should be able to insert the Applet tag with the attributes into the Html document.

## Solution

*Set the attributes for the Applet tag*

In HDWE, the value of the CODE attribute is **AppletReader.class,** which can be used to render a drawing document as an applet in a Html document. In HDP, two methods : *getAppletwidth* and *getAppletHeight* are used to get the values for the WIDTH and HEIGHT attributes, and the PARAM attribute is used to contain a drawing document file's name, which can be fetched by *getFileName()* method.

*Insert the Applet tag*

The default support for Html version 3.2 is provided by a *HTMLEditorKit* class in Java. *InsertHTMLTextAction* is a static inner class of *HTMLEditorKit.* It can be used to insert an arbitrary string of Html into current Html document. A user can choose the Insert Applet menu item only from an opened html document. At least two HTML.Tags need to be supplied. The first Tag, parentTag, identifies the parent in the document to add the elements to. The second tag, addTag, identifies the first tag that should be added to the document as seen in the Html string. For example, if the developer wants to create an action to insert a table into the body. The parentTag would be HTML.Tag.BODY, addTag would be HTML.Tag.TABLE, and the string could be something like <table><tr><td></td></tr></table>.

The source code of InsertAction of the HDP is as follows, it is used to insert the current drawings into the Html document from which the Insert Applet menu item has been chosen.

```
class InsertAction extends AbstractAction
   { public InsertAction()
      { super(EditorConstants.insertAction);
      }

      public void actionPerformed(ActionEvent e)
      { if ( fileName != null)
         { String htmlInsert = "<applet code=AppletReader.class"+
            " width="+getAppletWidth()+
            " height="+getAppletHeight()+">"+
            "<param name=fileName value="+getFileName()+">"+
            "alt="+"Your browser understands the &lt;APPLET&gt;"+
            "tag but isn't running the applet, for some reason."+
    "Your browser is completely ignoring the &lt;APPLET&gt; tag!"+ "</applet>";

            HTMLEditorKit.InsertHTMLTextAction embed =
            new  HTMLEditorKit.InsertHTMLTextAction
            ("Embed",htmlInsert,HTML.Tag.BODY, HTML.Tag.APPLET);

            ActionEvent actionEvent = new ActionEvent(targetPane,1001,"Embed");
            embed.actionPerformed(actionEvent);
        }
        else
            System.out.println("No such a file available.");
    }
}
```

As above code shows, the **InsertAction** class can get the applet width, height and the drawing file name as the value of the Applet tag's attributes WIDTH, HEIGHT and PARAM. And the drawing file can be inserted into the Html document. As a result, the browser can display the right drawing document with the right size.

### 4.1.6 How to check if a drawing file has been modified or not

**Problem**

When a user tries to create a new drawing document, open a drawing document, close an opened drawing document, or exit from the HDP, the system should be able to check to see if the current opened drawing document has been changed since the last time it was saved. If it has been changed, the user should be asked if he or she wants to save the changes.

**Solution**

A member variable - *changeFlag* in **DrawPanel** class is used to store this information. The mechanism to determine whether a change has happened or not is by checking if the mouse has been dragged in the draw panel.

The *MouseDragged* method of **DrawPanel** class is invoked when a mouse button is pressed on a component and then dragged. Mouse drag events will continue to be delivered to the component where the first originated until the mouse button is released (regardless of whether the mouse position is within the bounds of the component). In HDWE, this means that the user has modified the content of the **DrawPanel**.

If a *mouseDragged* event has happened, the system will check to see if the *changeFlag's* value is true or not, if not, set it to true:

```
public void mouseDragged(MouseEvent e)
{       if( !changeFlag )
        changeFlag = true;
```

The implementation of the Close Action is as follows:

```
if drawPanel has been changed, prompt the user to save the current document;
        if user choose yes button, save the document and then clear the drawPanel;
        if user choose no button,  clear drawPanel;
        if user choose cancel button, the system will do nothing;
if drawPanel has not yet been changed, clear drawPanel directly.
Finaly, set ChangeFlag to false.
```

The source code for it is as follows:

```
  class CloseDrawingAction extends AbstractAction
  { CloseDrawingAction()
      {  super(EditorConstants.closeDrawingAction);
         }

    public void actionPerformed(ActionEvent e)
    { if (drawPanel.isChanged())
      {int result = JOptionPane.showConfirmDialog(editorFrame, "Save it?");


             if (result == JOptionPane.YES_OPTION)
             {        if(fileName != null)
                      {        SaveDrawingAction sa = new SaveDrawingAction();
                               sa.actionPerformed(e);
                      }
                      else
```

```
                    {SaveDrawingAsAction saa = new SaveDrawingAsAction();
                            saa.actionPerformed(e);
                    }
                }

                if(result != JOptionPane.CANCEL_OPTION)
                    drawPanel.clearDrawPanel();
                }
                else
                    drawPanel.clearDrawPanel();

              editorFrame.setTitle("Untitiled New File.");
                fileName = null;
    }
  }
```

## 4.1.7 The Curve class

**Problem**

In HDP, a member variable of the **DrawPanel** class - *items* is used to store all of the drawing items of the **DrawPanel**. The element of the *items* vector is an object of the **Item** class. An **Item** object stores information about a geometric object's coordinate points, background color, foreground color, and rendering hints. As discussed in section 4.1.3, the Java 2D primitives do not support serialization, all these geometric objects have to be replaced by the shape codes and the points that construct them. The constructing points of the geometric object need to be saved in order to initialize the geometric object from a stream. One of these geometric objects can be an object of as a **GeneralPath** class, which represents a geometric path constructed from straight lines, it can contain multiple subpaths. The problem is that the **GeneralPath** class does not provide any method to set or get the points that constructed the **GeneralPath**. To remedy this problem, a subclass of the **GeneralPath** class was considered. It was found however that the **GeneralPath** class is a final class, it can not be extended.

**Solution**

In order to extend the **GeneralPath** class to handle points, a new class named **Curve** is implemented to contain **GeneralPath** as a member variable. The **Curve** class has a

Vector class to store the points of the **GeneralPath** class, and the setter and the getter methods to set or get those points. Some methods of the **GeneralPath** class such as *lineTo*, *moveTo* have been re-written to handle the points.

The part of the source code of the **Curve** class is as follows:

```
//----------Member Vairable -------------
      private Vector        points;
      private GeneralPath curve;

//---------- Constructors ---------------
      public Curve()
      {     points = new Vector();
            curve = new GeneralPath();
      }

      public Curve(Vector p)
   {  this();
      setPoints(p);
      curve.moveTo(((Double)points.elementAt(0)).floatValue(),
               ((Double)points.elementAt(1)).floatValue());

      for (int i=2;i < p.size();i+=2)
            curve.lineTo(((Double)points.elementAt(i)).floatValue(),
                  ((Double)points.elementAt(i+1)).floatValue());
   }
```

The *lineTo* methods of the **GeneralPath** class has been re-written as follows:

```
   public void lineTo(float x,float y)
   {  curve.lineTo(x,y);
      points.addElement(new Double(x));
      points.addElement(new Double(y));
   }
```

The *moveTo* methods of the **GeneralPath** class has been re-written as follows:

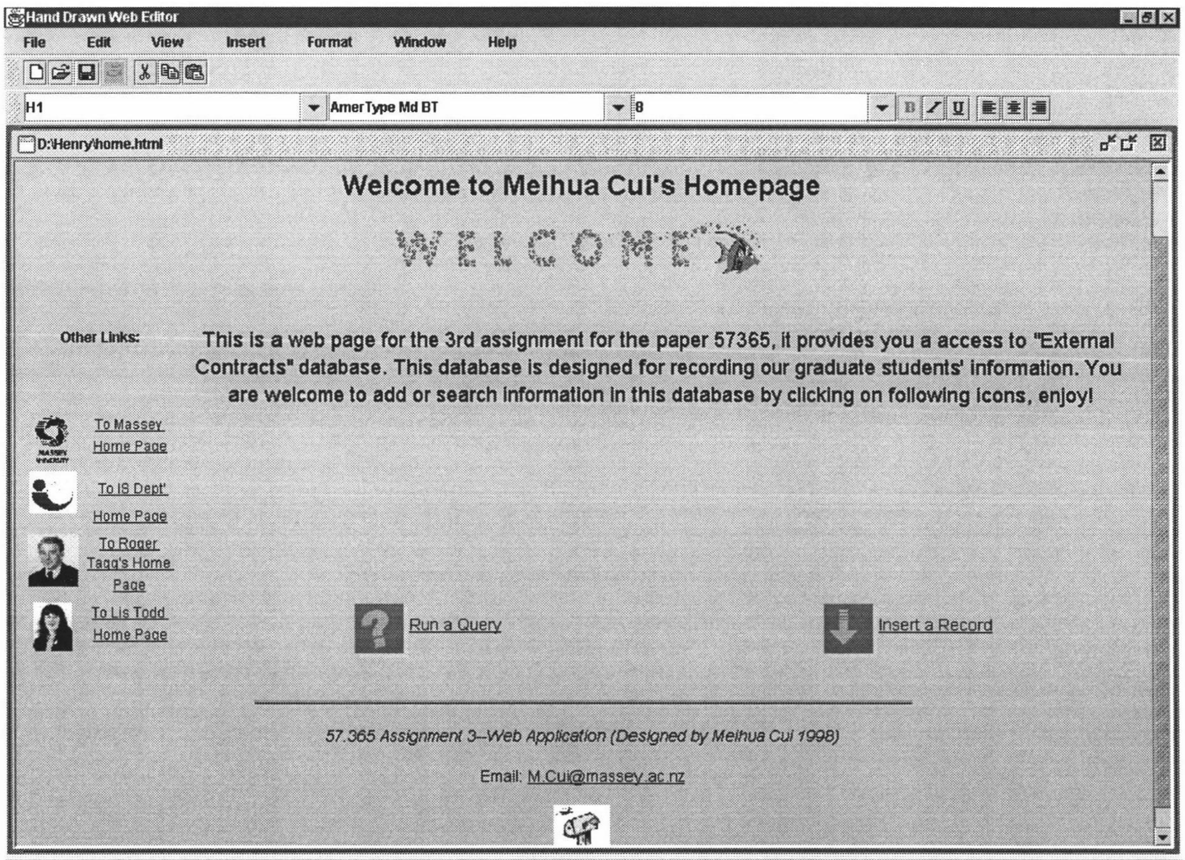
```
   public void moveTo(float x,float y)
   {  curve.moveTo(x,y);
      points.addElement(new Double(x));
      points.addElement(new Double(y));
   }
```

## 4.2 Examples of Using of the HDWE System

In this section, some examples of the use of the HDWE and HDP have been given.

### 4.2.1 HDWE

The Fig. 4.1 shows the HDWE has opened an existing Html document. The file name is shown on the Editor Frame. A user can edit, insert, save, or close this document.



Welcome to Meihua Cui's Homepage

This is a web page for the 3rd assignment for the paper 57365, it provides you a access to "External Contracts" database. This database is designed for recording our graduate students' information. You are welcome to add or search information in this database by clicking on following icons, enjoy!

*57.365 Assignment 3--Web Application (Designed by Meihua Cui 1998)*

Email: M.Cui@massey.ac.nz

Fig.4.1 HDWE opened an Html document.

## 4.2.2 HDP

An snapshot of HDP is shown in Fig.4.2, some different primitives have been drawn using different colors and line styles.

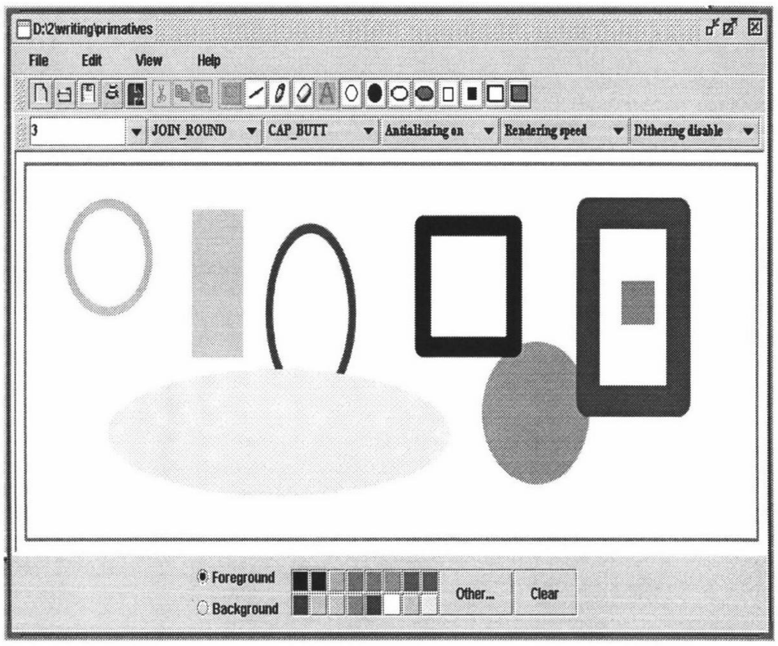


Fig.4.2 All kinds of primitives.

As the following snapshot (Fig.4-3.) shows, when a user edits an Html document, he or she may want to insert an arbitrary hand drawing object into that document. The user can do so by selecting the Insert Applet menu item from the Insert Menu, as a result, a HDP will be activated to let the user hand draw curves or primitives. After finishes the drawing, the user can embed this drawing object by clicking on the Insert Applet tool button on the HDP, or selecting the Insert Applet menu item from the File menu of HDP.

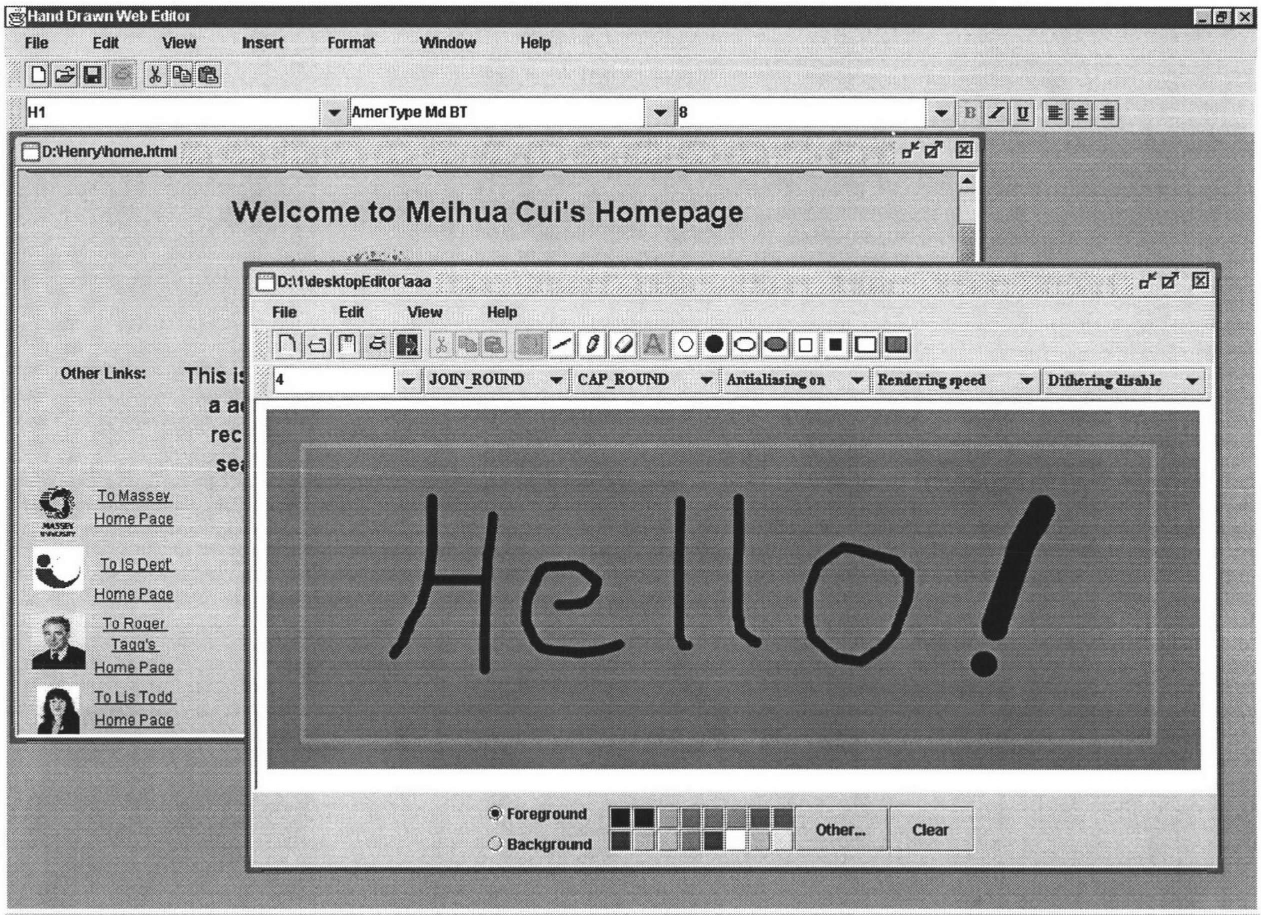
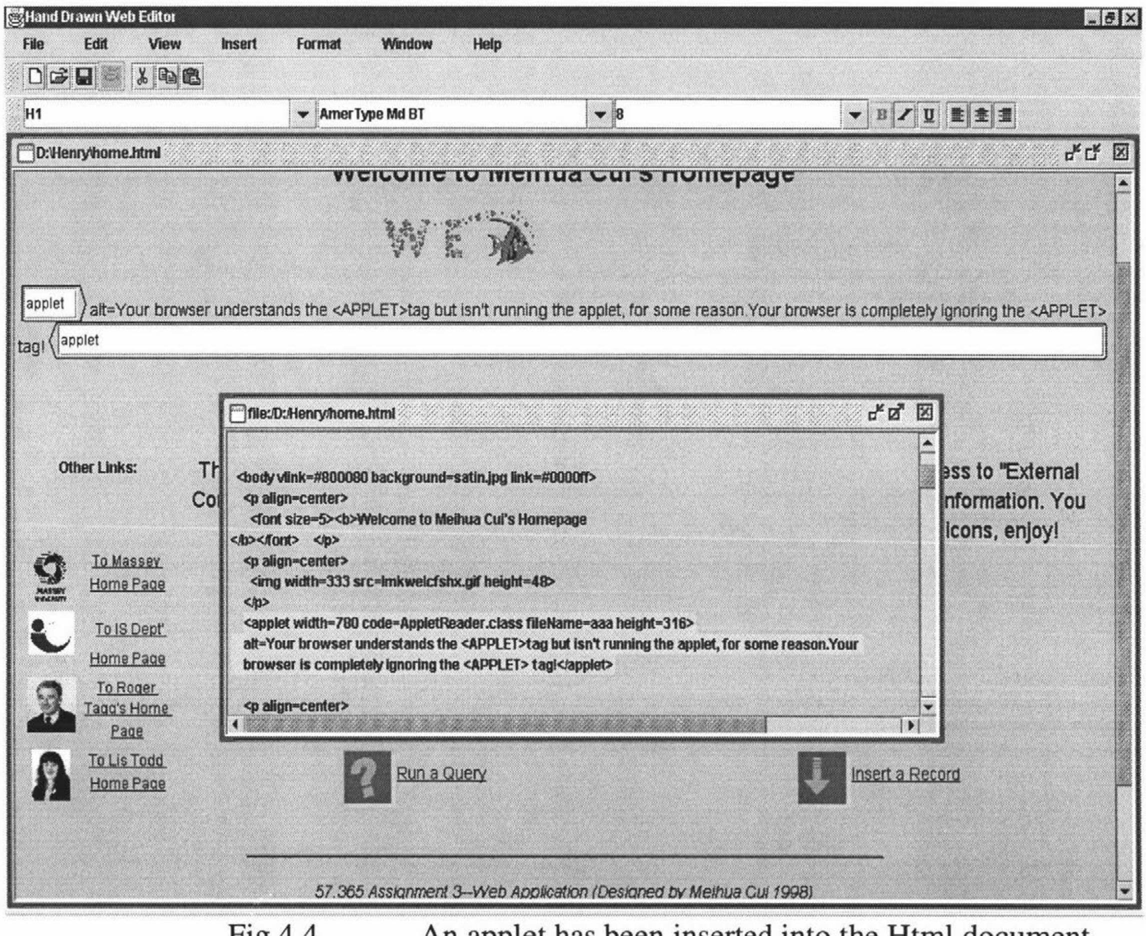


Fig.4.3 An arbitrary hand drawing document.

The following snapshot ( Fig.4-4) shows that an applet has been inserted into the Html document. However, neither Netscape nor Internet Explorer supports Java 2, and the **HtmlEditorKit** class in Java has not yet implemented the AppletView class to display an Applet tag. As a result, the HDWE can not yet run the applet, it only shows the alternative text of the applet tag. The source file of the Html document clearly shows that the Applet tag has been successfully inserted into the Html document.



alt=Your browser understands the <APPLET>tag but isn't running the applet, for some reason.Your browser is completely ignoring the <APPLET> tag!

```
 <body vlink=#800080 background=satin.jpg link=#0000ff>
  <p align=center>
   <font size=5><b>Welcome to Meihua Cui's Homepage
</b></font>    </p>
  <p align=center>
   <img width=333 src=lmkwelcfshx.gif height=48>
  </p>
  <applet width=780 code=AppletReader.class fileName=aaa height=316>
  alt=Your browser understands the <APPLET>tag but isn't running the applet, for some reason.Your
  browser is completely ignoring the <APPLET> tag!</applet>

  <p align=center>
```

Fig.4.4 An applet has been inserted into the Html document

A full user manual of the HDWE and HDP is given in the Appendix A.

# Chapter 5

# Conclusions and Further Work

## 5.1 Comments on using Java API Framework

The Java API framework has been heavily used in developing the HDWE system. The pros and cons of using a framework are discussed in this section.

**Definition**

A framework is an object-oriented class hierarchy plus a built-in model of interaction that defines how the objects derived from the class hierarchy interact with one another.

This rather simple definition belies the power of frameworks. In practical terms, the framework approach leverages capital-intensive software investment through reuse, and provides a much higher-level application programming interface, so that applications can be developed faster.

A framework is more than a class hierarchy. It is a miniature application complete with dynamic as well as static structure. It is a generic application a developer can reuse as the basis of many other applications. A frameworks is also specialized for a narrow range of application, because each model of interaction is domain-specific, e.g., designed to solve a narrow set of problems. A framework is the product of many iterations in design and evolves over long periods of time.

**Cons**

The frameworks is designed for a general purpose, which result in the major limitation of the frameworks - the interactions among object cannot be specified. It is necessary for the developer to create domain-specific classed by hand, this can be done by deriving subclasses from these classes.

A second problem is the separation between framework, editor, and resource. Unless the developer is knowledgeable in these three areas, it is unlikely that the developer will use the frameworks effectively. Frameworks are not suitable for novice or inexperienced experts, it is an extremely complex structure, and needs a long learning curve for a developer to get master it. Due to these deficiencies, enormous effort would be attached to the application implementation.

Frameworks also possess their own forms of complexity. The fragile base class problem, the lack of dynamic interfaces, and the confusion brought on by buzzwords are a few of the intrinsic complexities of frameworks.

**Pros**

Reuse through inheritance yields dramatic programmer productivity, and saves considerable programming time and effort.

Building blocks: objects are in fact self-contained building blocks; hence they are ideal for a unified theory of composition.

Reuse: frameworks are in fact reusable. And if the developers have a sufficient supply of application-specific frameworks, then they can build 80% of the world's application with only 20% of the effort required when starting from scratch.

Testing: components of a framework are assumed to be highly reliable because of exhaustive testing. After all, they have been reused many times.

## 5.2 Achievement

A fully working HDWE with the facility to input and display Html document and hand drawn document has been implemented. The following section describes the functionality of HDWE and HDP respectively.

### 5.2.1 HDWE

The HDWE has been implemented to create a new Html document, open an existing document, save the current document, save the current document with a new name, closes the current document, close all of the opened documents at once, and exit from the HDWE.

Using the Edit menu, a user can edit the current document, such as cut, copy, paste, select all, undo, and redo. The user can also use the View Source menu item to switch between the source (shows tags) and the target (hide tags) documents, and use the Reload menu item to close the previous copy of the document without saving it, and re-open a new copy of that document.

Using the Insert Menu, a user can insert applets, images, forms, URLs, tables, frames, and lists into the current Html document.

### 5.2.2 HDP

HDP has been implemented to create a new drawing document, open an existing drawing document, save the current drawing document, save the current document with a new name, close the current document, close all of the opened documents at once, insert the current drawing document from HDP into the current html document, and can exit from the HDP.

Using the tool buttons provided by HDP, a user can select, draw a line, draw a curve, erase an item, draw a circle or a filled circle, draw an ellipse or a filled ellipse, draw a square or a filled square, or draw a rectangle or a filled rectangle.

A user can use the line style comboBoxes to select the line width, join style, and the endcap style, or use rendering comboBoxes to control rendering quality, such as rendering the object with antialiasing, whether the objects to be rendered as quickly as possible, or the rendering quality be as high as possible, or whether or not the dithering is enabled.

A user can also use color palette to control the foreground and background color of the Drawn Panel. The user can choose a commonly used color, or a specific color from other color panels by clicking on the Other button. It provides the user with three color choser panels:

- Swatches -- for choosing a color from a collection of swatches.
- HSB -- for choosing a color using the Hue-Saturation-Brightness color model.
- RGB -- for choosing a color using the Red-Green-Blue color model.

The HDWE system uses the Windows Multiple Document Interface (MDI), so that a user can have several documents opened for editing simultaneously.

## 5.3 Further Work

Even though the HDWE and HDP have been implemented to edit and display Html documents and hand drawing documents, there are still many features that could have been included in both the HDWE and the HDP editing component.

For HDWE, the further work could includes:

- Implement AppletView class
- Completely resolve the concurrency problem.
- Completely provide undo redo support for the whole system.
- Implement all of the insert menu items:
    - Insert Images
    - Insert Forms
    - Insert URLs
    - Insert Frames

- Context sensitive help

For the HDP, the further work could include:

- Extend the function of the HDP

In following sections, each aspect of this further work will be discussed in detail.

## 5.3.1 Implement AppletView Class

An applet is a little application program, which can perform some simple tasks without having to send a user request back to the server. Java applets can be included in Html documents, much in the same way an image is included. When the user uses a Java-compatible browser to view a page that contains a Java applet, the applet's code is transferred to user system and executed by the browser.

Here is an example of a simple APPLET tag:

<applet code="AppletReader.class" width=100 height=140></applet>

This tells the viewer or browser to load the applet whose compiled code is in AppletReader.class (in the same directory as the current HTML document), and to set the initial size of the applet to 100 pixels wide and 140 pixels high.

The default support is provided by the **HTMLEditorKit** class. But the <applet> tag has not been supported yet. In order to view a html document including an applet element, HDWE needs to implement the **AppletView** class.

The mechanisms to implement the **AppletView** class is as follows:

The **AppletView** class needs to implement the view interface for <applet> elements. This view needs to load the class specified by the classid attribute. If possible, the Classloader used to load the associated Document can be used. This would typically be the same as the ClassLoader used to load the **EditorKit**. If the documents ClassLoader is null, Class.forName can be used instead. If the class can successfully be loaded, an attempt will be made to create an instance of it by calling Class.newInstance. An attempt can also be made to narrow the instance to type java.awt.Component to display the object.

### 5.3.2 Completely Resolve the Concurrency Problem.

The multiple document interface (MDI) is a specification that defines a user interface for applications that enable the user to work with more than one document at the same time. Each document in an MDI application is displayed in a separate child window within the client area of the application's main window.

The concurrency problem with the MDI is actually the problem of the readers and writers, which can be described as follows:

> A file is accessible by a number of user processes. On some occasions the file is to be read, which implies that it is not changed, and on other occasions the file is to be written, i.e. changed. The ground rules of the problem are that since the resource does not change state whilst being read, it may be read by any number of processes simultaneously. If any process is in the act of changing the state of the resource, then that process must have exclusive access to the resource, i.e. it would not be appropriate for another process either to be writing at the same time, or indeed reading while the writing was in progress.

In this problem, the use of a monitor is required to grant (or perhaps deny) access to the shared resource - a file. In this case it is necessary for the user processes to indicate not only that they wish to use the resource, but also the mode in which they wish to do so.

A boolean is required to indicate whether or not the resource has been allocated exclusively to a process. In addition, a variable is required to indicate whether or not the resource is being used for reading. Since more than one process may be in the act of reading the resource, this variable will have to be a counter rather than just a boolean. Because there are two conditions under which a process may be delayed, namely an attempt to read and an attempt to write to the resource, it is necessary for the monitor to include two condition variables that can be named okToRead and okToWrite.

In designing a monitor to solve the readers and writers problem, a strategic decision has to be made. This may be stated as - If a number of processes are reading the resource and a request arrives to write to the resource, this request will be delayed. The question of strategy is to decide whether or not this request should take priority over subsequent read requests, which could in principle start immediately, since other reads are already in progress. The argument for allowing these requests to go ahead is that it maximises concurrency, or equivalently minimises the total delay. The disadvantage of this approach is that, in a heavily loaded system, the write operation may be delayed indefinitely. An additional disadvantage is that a write request heralds the updating of the information held by the resource, and that delaying the write operation will mean that more reading of un-updated information will take place. The correct decision as to which of the two approaches to take will depend on the precise nature of the information held by the resource, and the frequency with which the resource is accessed.

### 5.3.3 Undo Redo Support

HDWE should be able to provide the Undo command in the Edit menu that allows the effect of the last operation to be undone. Typing text, and any command that changes the content of the document, should be undoable. But, following actions that cannot be undone:

- Scrolling and windowing commands
- Text selection
- Undo itself (it can be undone with Redo)
- Any actions performed prior to the last time the document was saved cannot be undone.

If a user executes several Undo commands in a row, he or she will undo the most recent action, and then undo the second most recent action, and so forth. By default, the user should be able to undo the last 30 actions. The user can also be allowed to change the default undo limit. To reverse an Undo, the user can use Redo. If the user has performed several Undos, he or she can reverse each of them by performing an equal number of Redos. If the user perform one or more Undos, and then perform an undoable action, the user will no longer be able to redo any of the Undos.

### 5.3.4 Context Sensitive Help

HDWE should be able to provide a user with help facilities that are context sensitive. Enabling the user to select from those topics that are relevant to the actions currently being performed. Obviously, this reduces the time required for the user to obtain help and increases the "friendliness" of the interface.

As the user moves from program to program, from field to field, he or she may need help in understanding what he or she should enter in the current field, or even what the purpose of the current program is. The HDWE system should be able to provide the user with the appropriate manual to the section that deals with the question currently being prompted on the screen. The user should be able to physically do this on the keyboard by hitting the key that has been designated as the HELP hot key, usually the F1 function key.

This should gives the user sufficient information to resolve problems without having to resort to the hard copy manuals, which are generated directly from this online help text. Once the user has read enough to resolve his or her problem, the user can simply press the ESC key to return to the exact point in the program that the user was at prior to activating the help processor.

Every field in every program should have context sensitive help associated with it, and the entire program too.

This context sensitive help of HDWE should also be supplemented by the HOWDOI process, which provides a search engine to specific procedures to solve common problems and answer typical questions.

### 5.3.5 Implement all of the Insert Menu Items.

**Insert Images**

A user should be able to insert an image file into the current Html document. The procedures could be:

1 Position the insertion point where the user want to insert a picture or an image.

2 On the Insert menu, choose the Image menu item, a file chooser interface will be activated so that the user can choose an image file.

3 Locate the file that contains the image the user wants to insert.

4 Double click that image file.

**Insert Forms**

Form is used to present a fill-out form to be used for the user actions such as registration, ordering, or queries. The forms can contain a wide range of HTML markup including several kinds of form fields such as single and multi-line text fields, radio button groups, checkboxes, and menus.

The entire form is a single element enclosed by <FORM> and </FORM>. The <FORM> tag takes two important attributes, ACTION and METHOD. The HDWE should be able to provide the user with a user interface to fill in the values of these attributes.

**Insert URLs**

Hypertext links are defined with the <A> Anchor element. The HDWE should be able to provide the user with an interface to insert a hyperlink as follows:

1 Select the text or drawing object the user wants to display as the hyperlink, and then click the URL menu item in the Insert menu. A user interface should be able to be activated.

2. If the user has any unsaved changes, HDWE should be able to prompt the user to save the file.

3 In the Link to file or URL box provided by the user interface, enter the path of the file the user wants the hyperlink to jump to, or click Browse to select from a list of files.

4 In the Named location in file box, enter the sub-address.

5. Click ok to change selected text or drawing object to a Hyperlink.

**Insert Frames**

*Creating frames*

A user should be able to create frame areas in several different ways. For example, clicking on the certain button will split the currently selected frame horizontally into two equal frames, and clicking on the other button will split the selected frame vertically. If a user right-click on the graphical area of the frame editor, a pop-up menu should be appear, from which the user can choose Split horizontally or vertically. To create custom-width frame areas, HDWE should provide the user with a user interface to:

- Move the cursor to the edge of any graphical area in the frame editor.
- Hold down the left mouse key. The cursor changes to a double-headed arrow.
- Drag away from the edge. This will 'pull' the edge of the frame with the mouse, creating two different frame areas.

## 5.3.6 Extend the Function of the HDP

The HDP needs to be extended to allows a user to:

1. Select a rectangular or an irregularly shaped area of a drawing object. To remove the selection box, the user can click outside the box.
2. Copy part of a drawing object by selecting the area to be copied and then dragging the cursor to define the area to paste.
3. Change the size of a drawing object. If the current picture is bigger than the new size, it is cut from the right side and bottom to fit within the smaller area.
4. Zoom in or out of a picture.
5. Rotate a picture. A user may select a free-form shape to be rotated.
6. Stretch or skew an item by entering amount into the stretching or skewing option form.

# Appendix A HDWE User Manual

## A.1 Introducing the HDWE

The HDWE is a stand-alone electronic publishing application that can help a user to design pages for the World Wide Web (WWW). Web pages are written in Hyper Text Markup Language (HTML) and Cascaded Style Sheet.

HDWE offers many ways to make creating HTML documents easier. If a user is an experienced HTML author, he can type all the formatting tags directly, or select them from menus and pop-up lists. If a user is new to HTML, HDWE has screens to let him insert images, formatting, and hypertext links into the document without having to learn the HTML language.

## A.2 HDWE Features

Supports HTML 3.2 and CSS1.

Windows 95-style interface, Macintosh interface, or cross platform interface.

Provides the Applet draw panel on which a user can draw anything using a pen input device.

## A.3 HDWE Menus

### A.3.1 File Menu

| | |
|---|---|
| New : | Opens a new window for editing a new html document. The new document will be referred to as "Untitled" on the editor frame. |
| Open: | Opens an existing file for editing. The file will appear in its own window, and its name will be listed on the editor frame. |
| Save : | Saves the current document. If a user has not saved this document |

before, this option has the same effect as choosing the Save As menu item from the File menu.

Save As : Saves the current document with a new name.

Close: Closes the current document. If the document has been changed since the last time it was saved, the user will be asked if he or she wants to save the changes.

Preview: Launches the default browser so that the user can see what the document will be look like to a viewer.

Print: Prints the current document to the default printer.

Print Setup: Lets the user change the default printer, and change the setup of the printer.

Close All: Provides a user with a quick way to close all of the opened documents. If a document has been changed since the last time it was saved, the user will be asked if he or she wants to save the changes.

Close & Save All: Saves all the current documents without prompting the user and then closes all of them.

Close All Without Save All : Closes all opened documents at once without saving or prompting the user to save.

Exit : Leaves HDWE. If any document has been changed since the last time it was saved, the user will be asked if he or she wants to save the changes.

### A.3.2 Edit Menu

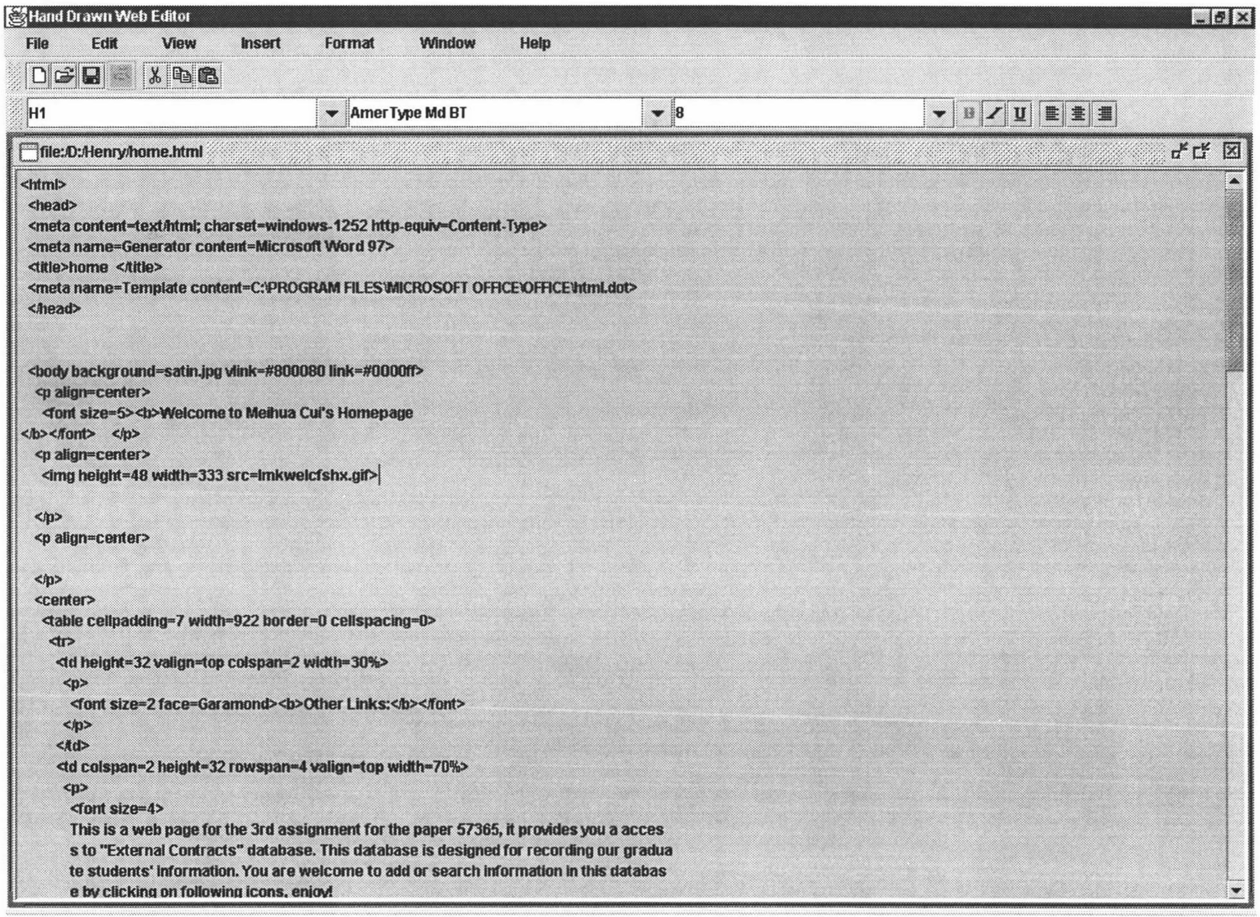
| | |
|---|---|
| Cut : | Removes highlighted text from the user's document and puts it in the Clipboard . The text may then be pasted from the Clipboard. |
| Copy : | Takes a copy of highlighted text in the user's document and puts it in the Clipboard . The text may then be pasted from the Clipboard. |
| Paste : | Inserts the contents of the Clipboard in the user's document. The information remains in the clipboard, so he or she can use Paste to insert the same information repeatedly. |
| Select All: | Selects the whole content of current document at once. |
| Undo : | Reverses the last action. For example, if the user accidentally deleted some text, he or she could use Undo to get it back. |
| Redo : | Reverses the last action. For example, if the user have undone a action by choosing Undo menu item, the redo can re-do that action. |

### A.3.3 View Menu

| | |
|---|---|
| Tool Bar : | Provides fast access to commonly-used HDWE functions. It can be hidden from the View menu. |
| Format Bar : | Provides fast access to commonly-used HDWE font and tag functions. It can be hidden from the View menu. |
| Status Bar : | HDWE uses the Status bar at the bottom of the screen to tell the user what it is doing. |
| View Source : | Provides a convenient way to switch between the Source(shows tags) and target(hide tags) documents. |

Reload : Closes the previous copy of the document without saving it, and re-opens a new copy of that document.

The following figure shows the source content of the html document being opened by Fig. 4-1.



```
<html>
 <head>
 <meta content=text/html; charset=windows-1252 http-equiv=Content-Type>
 <meta name=Generator content=Microsoft Word 97>
 <title>home  </title>
 <meta name=Template content=C:\PROGRAM FILES\MICROSOFT OFFICE\OFFICE\html.dot>
 </head>


 <body background=satin.jpg vlink=#800080 link=#0000ff>
  <p align=center>
   <font size=5><b>Welcome to Meihua Cui's Homepage
</b></font>   </p>
  <p align=center>
   <img height=48 width=333 src=Imkwelcfshx.gif>

  </p>
  <p align=center>

  </p>
  <center>
   <table cellpadding=7 width=922 border=0 cellspacing=0>
    <tr>
     <td height=32 valign=top colspan=2 width=30%>
      <p>
       <font size=2 face=Garamond><b>Other Links:</b></font>
      </p>
     </td>
     <td colspan=2 height=32 rowspan=4 valign=top width=70%>
      <p>
       <font size=4>
       This is a web page for the 3rd assignment for the paper 57365, it provides you a acces
       s to "External Contracts" database. This database is designed for recording our gradua
       te students' information. You are welcome to add or search information in this databas
       e by clicking on following icons. enjoy!
```

Fig.A.1 Source of the document opened in Fig.A.1.

### A.3.4 Insert Menu

Applet : Inserts an Applet into the Html document, the applet is resizable and acts like a white board. The user can draw or erase different styled and colored lines, curves, primitives, text and images.

Image : Inserts an image into the current document. This will usually be in GIF or JPG format. The image file can be in the same directory as the current document.

| | |
|---|---|
| Form : | Inserts a form into the current document. Form is used to present a fill-out form to be used for the user actions such as registration. It takes two important attributes, Action and Method. The HDWE provides the user with an interface to fill in these attributes. |
| URL : | Hypertext Targets let the user jump to other specific locations. The user can jump to a target in the same document, or another document. Targets are most often used in long a document which is divided into several sections. |
| Tables : | Inserts Table, Table Row or Table Data Cell into the current document. A user can do so by selecting corresponding sub menu item from the Table Menu Item. |
| Frame : | Inserts a frame or frame set by right-clicking, and then a pop-up menu will allow the user to choose to split horizontally or vertically. |
| Lists : | Inserts ordered list, ordered list item, unordered list, unordered list item into the current Html document by selecting corresponding sub menu item from the List menu Item. |

### A.3.5 Format Menu

| | |
|---|---|
| Alignment: | Aligns highlighted portion of content to left, right or center. |
| Bold : | Changes selected portion of content to the bold style. |
| Italics : | Changes selected portion of content to the italics style. |
| Underline : | Changes selected portion of content to the underline style. |

### A.3.6 Window Menu

HDWE uses the Windows Multiple Document Interface (MDI). This means that a user can have several documents opened for editing simultaneously. The Window Menu provides some easy ways to manage these documents.

Cascade : Overlaps all opened documents from the top left to the bottom right of the screen, so that the title bars of all documents are visible.

Tile : Arranges all open documents from top to bottom across the screen. The height of each document window will be reduced so they all fit in the screen.

Window List: This lists all the documents a user currently has opened.

### A.3.8 Help Menu

| | |
|---|---|
| Content : | Provides the user all the content of help topics. |
| Search : | Provides the user with an index so that the user can find a specific help topic quickly. |
| About HDWE: | Describes the general information of the HDWE. |

## A.4 Hand Drawn Panel Menus

### A.4.1 Menu Bar

#### *A.4.1.1 File Menu*

New Drawing: Opens a new window for drawing. The new document will be referred to as "Untitled" on the editor frame.

Open Drawing:Opens an existing drawing document for editing. The file will be appeared in its own window, and its name will be listed on the editor frame.

Save Drawing:Saves the current document. If a user has not saved this document before, this option has the same effect as choosing the Save As from the File menu.

Save As : Saves the current document with a new name.

Close : Closes the current document. If the document has been changed since the last time it was saved, the user will be asked if he or she want to save the changes.

Print : Prints the current document to the default printer.

Insert Drawing:Inserts the drawing from HDP into the html document from where the insert applet menu item has been chosen.

Exit : Leaves HDP. If the drawing document has been changed since the last time it was saved, the user will be asked if he or she wants to save the changes.

#### *A.4.1.2 Edit Menu*

Cut : Removes highlighted portion of drawing from the user's drawing object and puts it in the Clipboard . It may then be pasted from the Clipboard.

Copy : Takes a copy of highlighted portion of drawing from the user's drawing object and puts it in the Clipboard .It may then be pasted from the Clipboard.

Paste :Inserts the contents of the Clipboard into user's drawing object. The information remains in the clipboard, so that the user can use Paste to insert the same information repeatedly.

Select All :Selects the whole drawing on the Hand Drawn Panel.

Undo :Reverses the last action.

| | |
|---|---|
| Redo | :Reverses the last undo action. For example, if a user have undone a action, he can redo that action by choosing the Undo menu item from the Edit Menu. |

*A.4.1.3 View Menu*

| | |
|---|---|
| Tool Bar | :Provides fast access to commonly-used Hand Drawn Panel functions. It can be hidden from the View menu. |

*A.4.1.4 Help Menu*

| | |
|---|---|
| Content | :Provides a user with all of the help content of HDP. |
| Search | :Provides a user with an index so that the user can find a specific help topic quickly. |
| About HDP | :Describes the general information of the HDP. |

### A.4.2 Tool Bar

The screen snapshot of Tool bar of Hand Drawn Panel is as follows:

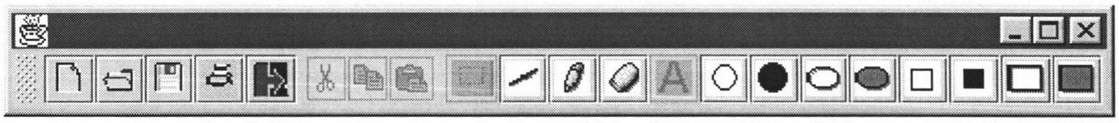


Fig.A.2 Screen snapshot of Tool Bar of Hand Drawn Panel.

The tool buttons are divided into two sets, the first 8 buttons' functionality has been described in A.4.1. The rest of the tool buttons provide the user the functionality as follows:

- Select
- Draw a line
- Draw a curve
- Erase an item
- Enter Text
- Draw a Circle
- Draw a Filled circle
- Draw an ellipse
- Draw a filled ellipse
- Draw a square
- Draw a filled square
- Draw a rectangle
- Draw a filled rectangle

### A.4.3 Rendering Hint ComboBox

As follows is the snapshot of Rendering Hint ComboBoxes of HDP:

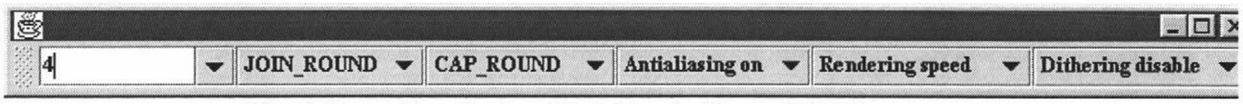


Fig.A.3 Rendering Hint ComboBox of HDP.

The first three comboBox is about Line Styles, a user can select a line width, a join style, and a endcap style from these three comboBoxes:

1. Line width : is the thickness of the line measured perpendicular to its trajectory.

2. Join Style : is the decoration that is applied where two line segments meet. There are three join styles: Join_Bevel, Join_Miter, and Join_Round.

3. Endcap Style : is the decoration that is applied where a line segment, there are three endcap styles: Cap-Butt, Cap_Round, and Rap_Square.

The last three comboBoxes let a user to control Rendering Quality. The user's preferences are specified as hints through the *RenderingHints* attribute in the *Graphics 2D* context. Not all platforms support modification of the rendering mode so specifying rendering hints does not guarantee that they will be used. When a hint is set to default, the platform rendering default is used is used.

4.Antialiasing : is a technique used to render objects with smoother-appearing edges. It can be set to default, on, or off.

| | |
|---|---|
| On | -- rendering is done with antialiasing |
| Off | -- rendering is done without antialiasing |
| Default | -- rendering is done with the platform default antialiasing mode. |

5. Rendering : lets a user to choose whether the objects to be rendered as quickly as possible, or that the rendering quality be as high as possible. It can be

set to speed, quality and default.

Speed -- Appropriate rendering algorithms are chosen with a preference for output speed.

Quality-- Appropriate rendering algorithms are chosen with a preference for output quality.

Default -- The platform default rendering algorithms are chosen.

6. Dithering : lets a user to choose whether or not to dither when rendering the objects. It can be set to disable, enable and default.

Disable -- do not dither when rendering

Enable -- dither when rendering, if needed

Default -- use the platform default for dithering

### A.4.4. Color Palette

As follows is the snapshot of Color Palette of HDP:

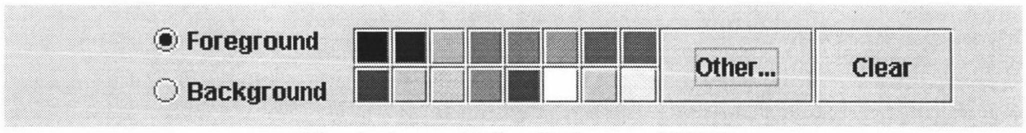


Fig.A.4 Color Palette of HDP.

The Color Palette of HDP is used to control the foreground and background color of the Drawn Panel. As shown above, each square box represent a commonly used color. A user can choose one of these colors by simply click it. If a specific color a user wants is not among these color boxes, the user can choose it from other color panels by clicking on the Other button. It provides the user with a palette of colors to choose from. As Fig. 5,6,7 show, the default color chooser provides three chooser panels:

- Swatches -- for choosing a color from a collection of swatches.

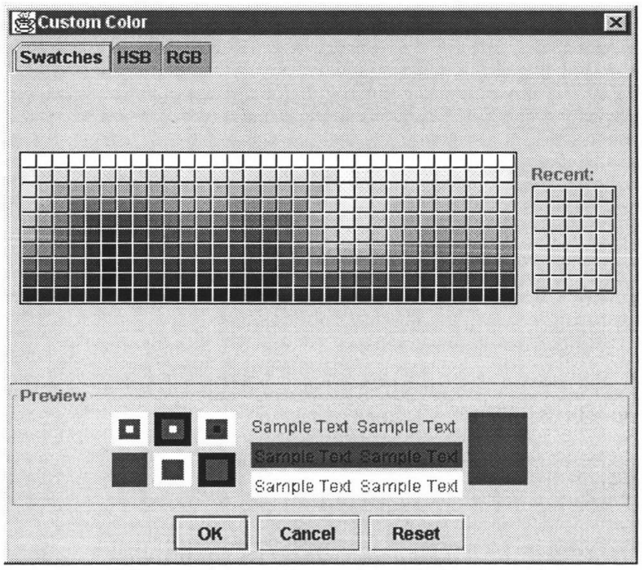
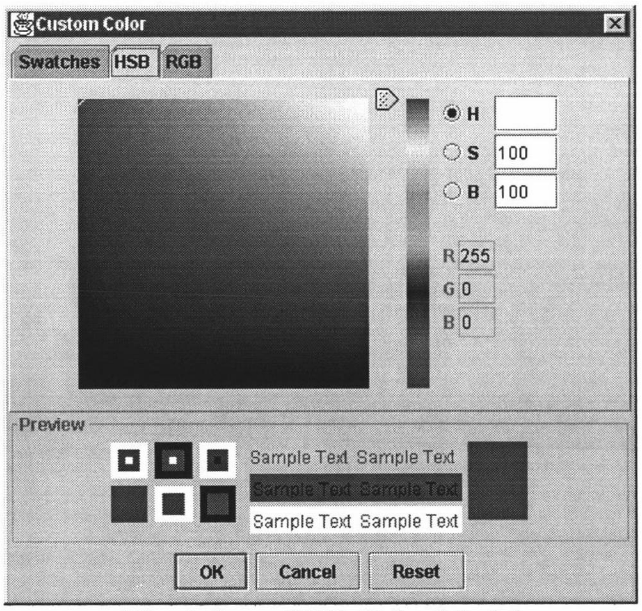


Fig.A.5 Swatches Color Model.

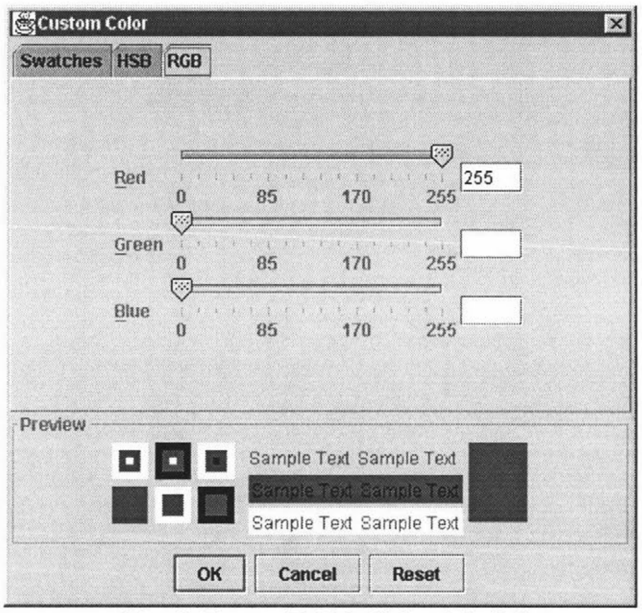
- HSB -- for choosing a color using the Hue-Saturation-Brightness color model.



H
S 100
B 100
R 255
G 0
B 0

Fig.A.6 HSD Color Model.

- RGB -- for choosing a color using the Red-Green-Blue color model.



Red: 255

Fig.A.7 RGB Color Model.

# Bibliography

## Books

1. Ian S.Graham, *HTML Stylesheet Sourcebook*, Wiley Computer Publishing,1997.

2. Craig Larman, *Applying UML and Patterns,* Prentice Hall, 1998.

3. Michael morrison, et al, *Java 1.1 unleashed,* Sams.net Publishing, 1997.

4. E.Stephen Mack and Janan Platt, *HTML 4.0,* SYBEX Inc., 1997.

5. Terry Quatrani, *Visual Modeling With Rational Rose And UML,* Addison Wesley Longman,Inc.,1998

6. Martin Fowler, Kerndall Scott, *UML Distilled, Applying The Standard Object Modeling Language,* Addison Wesley Longman,Inc.,1998

7. James J. Odell, *Advanced Object-Oriented Analysis & Design Using UML,* SIGS Books & Multimedia.,1998

8. Roger S.Pressman,*Software Engineering A Practitioner's Approach*, Third Edition, McGRAW-Hill International Editions,.1992.

9. Roger S.Pressman,*Software Engineering A Practitioner's Approach*, Fourth Edition, McGRAW-Hill International Editions,.1992.

10. Laura Lemay, Charles L.Perkins, *Teach Yourself JAVA 1.1 in 21 Days*, Second Edition, Sams.net Publishing,.1997

11. Russel Winder,Graham Roberts, *Developing Java Software*,John Wiley & Sons Ltd,.1998

12. Nancy M.Wilkinson, *Using CRC Cards, An Informal Approach to Object-Oriented Development*, SIGS Books & Multimedia.,1995

13. Richard Light, *Presenting XML*, Sams.net Publishing,.1997

14. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Company,.1994

15. Matthias Felleisen, Daniel P.Friedman, *A Little Java, A Few Patterns*, Massachusetts Institute of Technology Press,.1998

16. Grady Booch: *Object-Oriented Analysis and Design with Applications.*, Benjamin/Cummings, 1994

17. Grady Booch: *Object Solutions: Managing the Object-Oriented Project.* Addison-Wesley, 1995

18. Peter Coad and Edward Yourden: *Object-Oriented Analysis.* Yourdon, 1991

19. Peter Coad and Edward Yourden: *Object-Oriented Design.* Yourdon, 1991

20. James Rumbaugh,Michael Blaha, William Premerlani, Frederick Eddy, and William Lorenzen: *Object-Oriented Modeling and Design.* Prentice Hall, 1991

21. Ted Lewis, Larry Rosenstein, Wolfgang Pree, Andre Weinand, Erich Gamma, Paul Calder, Glenn Andert, John Vlissides & Kurt Shumucker, *Softbound,Object Oriented Application Frameworks.,1995*

22. Beck, Kent. "Patterns and Software Development." Dr. Dobb's Journal 19, no. 2 (February 1994): 18.

23. Beck, Kent and Johnson, Ralph. "Patterns Generate Architectures," European Conference on Object-Oriented Programming (1994).

24. Birrer, Andreas and Eggenschwiler, Thomas. "Frameworks in the Financial Engineering Domain: An Experience Report," European Conference on Object-Oriented Programming (1993): 21-35.

25. Booch, Grady. "Designing an Application Framework," Dr. Dobb's Journal 19, no. 2 (February 1994): 24.

26. Booch, Grady. Object-Oriented Analysis and Design With Applications. Redwood City, CA: Benjamin/Cummings, 1994.

27. Campbell, Roy; Islam, Nayeem; Raila, David; and Madany, Peter. "Designing and Implementing "CHOICES": an Object-Oriented System in C++," Communications of the ACM 36, no. 9 (September 1993): 117.

28. Coad, Peter. "Object-Oriented Patterns," Communications of the ACM 35, no. 9 (1992): 152.

29. Eggenschwiler, Thomas and Gamma, Erich. "ET++ SwapsManager: Using Object Technology in the Financial Engineering Domain," OOPSLA '92 Conference Proceedings, ACM SIG Notices 27, no. 10 (1992): 166.

30. Frameworks: The Journal of Software Development Using Object Technology. Software Frameworks Association.

31. Gamma, Erich; Helm, Richard; Johnson, Ralph; and Vlissades, John. "Design Patterns: Abstraction and Reuse of Object-Oriented Design," European Conference on Object-Oriented Programming (1993): 406-431.

32. Gamma, Erich; Helm, Richard; Johnson, Ralph; and Vlissades, John. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Forthcoming.

33. Goldstein, Neal, and Jeff Alger. Developing Object-Oriented Software for the Macintosh. Reading, MA: Addison-Wesley, 1992.

34. Johnson, Ralph. "How to Design Frameworks," OOPSLA '93 Tutorial Notes, 1993.

35. Mallory, Jim. "TI Software Speeds Semiconductor Production," Newsbytes NEW07200011 (July 1993).

36. Nelson, Carl. "A Forum for Fitting the Task," IEEE Computer 27, no. 3 (March 1994): 104.

37. Shelton, Robert. "The Distributed Enterprise," The Distributed Computing Monitor 8, no. 10 (October 1993): 3.

38. Stroustrup, Bjarne. The C++ Programming Language. 2d ed. Reading, MA: Addison-Wesley, 1991.

39. Wilson, Dave. "Designing Object-Oriented Frameworks." Personal Concepts, Palo Alto, CA 1994.

40. Wong, William. Plug & Play Programming, An Object-Oriented Construction Kit. M&T Books, 1993.

41. C. R. Snow, *Concurrent Programming*, Cambridge Computer Science Texts, 1993.

## Web Sites

42. http://fims-www.massey.ac.nz/~crjessho/html/Net_tools.html

43. http://www.javaworld.com/javaworld/jw-07-1998/jw-07-java-win32.html

44. http://www.oz.nthu.edu.tw/~u840901/proj/node14.html

45. http://www.oz.nthu.edu.tw/~u840901/proj/node19.html

46. http://www.relisoft.com/java/c_java.html

47. http://www.sm.go.dlr.de/~bob/Docu-intern/JavaPG/noMoreC/index.html

48. http://www.students.uiuc.edu/~gjkaiser/serious/acmarticle.html

49. http://kia.etel.ru/books/javainnet/appe.htm

50. http://www.soft-design.com/softinfo/objects.html

51. http://iamwww.unibe.ch/~scg/OOinfo/FAQ/

52. http://www.clark.net/pub/howie/OO/ooterms.html

53. http://java.sun.com/about.html

54. http://www.javaworld.com/javaworld/jw-07-1998/jw-07-java-win32.html

55. http://lcs.www.media.mit.edu/~moux/papers/PAAM96/node7.html

56. http://java.sun.com/products/jfc/tsc/getting_started/getting_started.html
#swing_comp_architecture

57. http://www.andromeda.com/people/ddyer/java/Reviews.html#kawa

58. http://www.tek-tools.com/kawa/

59. http://www.tek-tools.com/kawa/reviews.htm

60. http://www.cjug.org/news/kawa.html

61. http://java.sun.com/products/jfc/tsc/swingdoc-arch.htm

62. http://www.htw-dresden.de/~beck/JAVA11/SWING/mvc.html

63. http://java.sun.com/features/1998/11/jdk.html

64. http://www.symantec.com/domain/cafe/vcafese30/index.html

65. http://java.sun.com/pr/1998/12/pr981208-04.html

66. http://java.sun.com/products/jfc/tsc/index.html

67. http://java.sun.com/products/jfc/tsc/Text/text/text.html

68. http://java.sun.com/docs/books/tutorial/index.html

69. http://java.sun.com/docs/books/tutorial/2d/index.html

70. http://java.sun.com/docs/books/tutorial/2d/overview/index.html

71. http://java.sun.com/docs/books/tutorial/2d/overview/rendering.html

72. http://java.sun.com/docs/books/tutorial/2d/overview/coordinate.html

73. http://java.sun.com/docs/books/tutorial/2d/overview/shapes.html

74. http://java.sun.com/docs/books/tutorial/2d/overview/text.html

75. http://java.sun.com/docs/books/tutorial/2d/overview/images.html

76. http://java.sun.com/docs/books/tutorial/2d/overview/printing.html

77. http://bmccarty.apu.edu:8080/curricula/cs501/Lecture02/index.htm

78. http://www.uwa.edu.au/cwis/howto/forms.html

79. http://www.hut.fi/u/jkorpela/forms/

80. http://www.hut.fi/~jkorpela/HTML3.2/5.25.html

81. http://www.hut.fi/~jkorpela/HTML3.2/

82. http://www.manning.com/Lewis2/index.html

83. http://www.manning.com/Lewis2/Contents.html

84. http://www.manning.com/Lewis2/Preface.html

85. http://www.manning.com/Lewis2/Chapter1-1.html

86. http://www.developer.ibm.com:8080/library/aixpert/feb95
/aixpert_feb95_boof.html

87. http://www.developer.ibm.com:8080/library/aixpert/feb95
/aixpert_feb95_boofframe.html

88. http://hpsalo.cern.ch/TaligentDocs/TaligentOnline/DocumentRoot/1.0/Docs
/books/DF/DF_2.html

89. http://hpsalo.cern.ch/TaligentDocs/TaligentOnline/DocumentRoot/1.0/Docs
/books/DF/DF_133.html#HEADING195

90. http://www.mvblind.uni-linz.ac.at/aster/node104.html

91. http://simon.cs.cornell.edu/Info/People/raman/current/phd-thesis
/html/node18.html#SECTION00210000000000000000

92. http://www.w3.mag.keio.ac.jp/TR/NOTE-amaya

93. http://kb.indiana.edu/data/adnl.html

94. http://www.w3.org/Style/

95. http://www.w3.org/TR/REC-html40/struct/dirlang.html#h-8.2.1

96. http://server.htmlhelp.org/reference/html40/special/bdo.html

97. http://www.mcp.com/sites/1-56830/1-56830-306-8/links.html

98. http://www.eiffel.com/doc/manuals/technology/oo_comparison/

99. http://java.sun.com/docs/books/tutorial/ui/swingComponents /colorchooser.html

100. http://www.staminasoftware.com/context.htm

101. http://www.vtiscan.com/jdk1.2beta2/docs/guide/security/spec /security-spec.doc1.html

102. http://www.swiss.ai.mit.edu/~jbank/javapaper/javapaper.html

103. http://ei.cs.vt.edu/~wwwbtb/book/chap14/index.html

104. http://www.cs.princeton.edu/sip/java-faq.html

105. http://www.nikos.com/javatoys/deep/security/

106. http://www.disordered.org/Java-JIT.html

107. http://www.omnisoft.se/java_performance.html

108. http://www.inside-java.com/articles/perf/index.htm

109. http://java.sun.com/sfaq/