國立交通大學

資訊工程系

碩 士 論 文

Java AWT 移植到.NET Windows Forms 的可行性之研究

The Study of Portability of Java AWT to .NET Windows Forms

研 究 生：朱俊欣

指導教授：吳毅成　教授

中 華 民 國 九 十 三 年 六 月

Java AWT 移植到.NET Windows Forms 的可行性之研究
The Study of Portability of Java AWT to .NET Windows Forms

研 究 生：朱俊欣　　Student：Chun-Hsin Chu

指導教授：吳毅成　　Advisor：I-Chen Wu

國 立 交 通 大 學
資 訊 工 程 系
碩 士 論 文

A Thesis
Submitted to Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science and Information Engineering

June 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年六月

Java AWT 移植到.NET Windows Forms 的可行性之研究

研究生：朱俊欣　　指導教授：吳毅成

國立交通大學 資訊工程學系

摘要

Java AWT 和.NET Windows Forms 分別是在 Java 平台與.NET Framework 平台開發圖形使用者介面程式的兩個套件。AWT 套件受限於 Java 平台的 write once, run anywhere 特徵，想要呼叫作業系統提供的原生 API 十分困難，因此一些圖形使用者介面上的應用較難完成。Windows Forms 套件則提供更強大的圖形使用者介面開發功能，以及和 Windows 作業系統整合能力。如果可以使用 Windows Forms 開發程式，原本圖形使用者介面的許多需求更容易達成。

本論文探討從 Java AWT 移植到.NET Windows Forms 的可行性。先研究 Java AWT 套件和.NET Windows Forms 套件的差異，然後提出一個移植的解決方案，探討移植的可行性。最後以個案研究來探討實際的移植過程。

The Study of Portability of Java AWT to .NET Windows Forms

Student: Chun-Hsin Chu Advisor: I-Chen Wu

Department of Computer Science and Information Engineering

National Chiao Tung University

ABSTRACT

Java AWT is a package of building graphics user interface in Java. .NET Windows Forms is also a package of building graphics user interface in .NET Framework. Because of the feature “write once, run anywhere.” of Java, the AWT library is difficult to call the native library of the operating system. Therefore, some applications of graphics user interface are hard to implement. The Windows Forms library has much powerful API and integrates easily with Windows operating system. If we use Windows Forms to build graphics user interface, we can implement applications easily.

In this thesis, we study the portability of Java AWT to .NET Windows Forms. First, we introduce the difference between AWT and Windows Forms. Then, we propose a solution to port AWT to Windows Forms and analyze the portability. Finally, we use a real porting case to study and discuss the portability for the case.

誌謝

首先要感謝我的指導教授，吳毅成博士，由於他的細心指導，並提供許多寶貴的意見與指正，這篇論文才得以順利完成。

特別要感謝的就是實驗室的同學陳家齊、陳智文、林義欽及所有學長學弟們在研究及生活上的幫助。同時，也要感謝許許多多的同學及朋友，在我的研究期間，給了我相當多的關懷與鼓勵，陪我渡過這段最值得回憶的學生生活。

最後要感謝我親愛的父母，在我的求學生涯中給了我最大的支持和照顧。謹以此論文，獻給我最摯愛的家人。

目錄

表目錄

圖目錄

第一章　緒論

在本章之中，會先做個簡單的背景介紹，然後討論為什麼我們需要將使用 Java AWT 開發的程式，移轉到.NET Windows Forms。最後是本文大綱。

1.1 背景說明

圖形使用者介面(Graphics User Interface)是目前開發視窗應用程式重要部分，有一個好的圖形使用者介面，使用者才可以方便地使用程式，程式也會受到歡迎。

目前使用 Java 語言開發的程式已十分普遍。雖然 Java 語言發展的歷史不長，不過由於 Java 語言易於上手和跨平台的特性，使得 Java 語言越來越受歡迎。

使用 Java 語言開發的程式，雖然享受到跨平台的特性，但是也因此對於想要直接使用作業系統原生的 API，變成十分麻煩。比如說甲公司使用 Java applet 發展一個網路橋牌遊戲，讓使用者可以透過網路互相對戰(圖 1-1，畫面取自 CYC 遊戲大聯盟網站[16])。另外有一家乙公司在網頁上開發了 Avatar 系統(圖 1-2，畫面取自 CYC 遊戲大聯盟網站[16])，使用者透過瀏覽器進入 Avatar 系統，更換各種造型。

圖 1-1、甲公司的 JAVA APPLET 橋牌遊戲

圖 1-2、乙公司的網頁 AVATAR 系統

這兩家公司想要透過彼此合作，提升自己的競爭能力。於是乙公司希望能在甲公司的遊戲畫面中顯示 Avatar 的圖片(圖 1-3)，如此乙公司可以推廣自己的系統；而甲公司也因為遊戲畫面更加漂亮，可以吸引更多使用者加入遊戲。

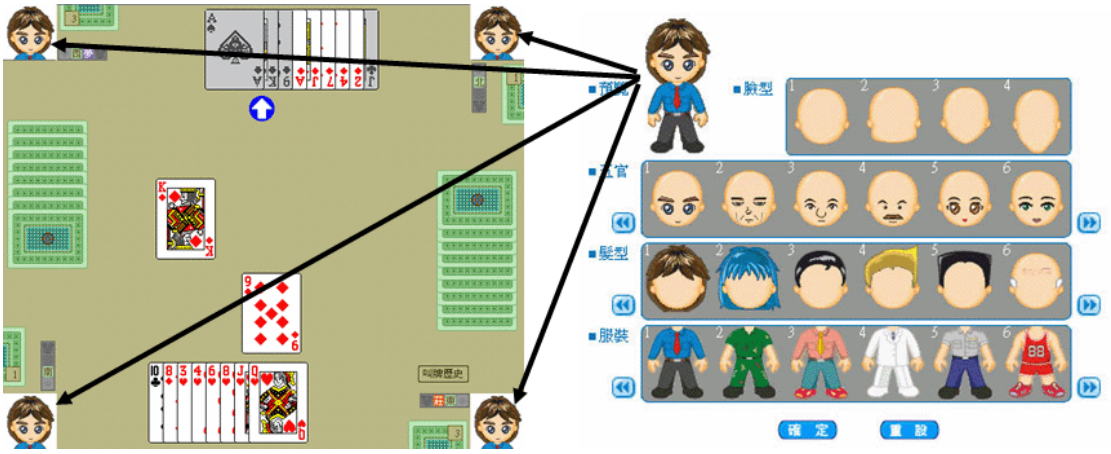


圖 1-3、甲公司和乙公司的合作

但是甲公司發現，使用 Java applet 開發的遊戲，在提供圖形使用者介面功能的 AWT 套件中，並不能直接顯示網頁[1]。正好執行遊戲的 Windows 作業系統，有提供一個 ActiveX 控制項，可以用來顯示網頁。不過必須要透過 JNI，經過非常複雜的過程，才可以使用此控制項。而且 JNI 的撰寫上也有許多限制[4]，像是錯誤檢查(Error checking)、傳送錯誤的參數給 JNI 的函式(Passing invalid arguments to JNI functions)、違反程式接觸控制的規範(Violating access control rules)等等。需要尋找其他的解決方案。

Microsoft 在 2000 年提出新的程式開發平台：.NET Framework [7]，提供許多 API、跨語言的開發環境、方便的部署方式、輕鬆地與 Windows 作業系統環境的整合。其中 Windows Forms 是負責開發圖形使用者介面的套件，可以解決上述 Java AWT 遇上的問題。因此，本論文將探討 Java AWT 移植到.NET Windows Forms 的可行性。

1.2 Java AWT 介紹

AWT 的全名是 Abstract Window Toolkit，是在 Java 平台上開發圖形使用者介面的套件[1]。AWT 提供了許多程式庫讓程式設計師可以完成建立視窗、使用控制項、繪圖、影像處理等等的工作。

1.2.1 Java 平台介紹

Java 平台是由昇陽(SUN)公司在 1995 年所提出的[5]。起初是為了解決消費性電子產品上開發程式的不便，而發展一套全新的開發平台。由於消費性電子產品的硬體平台差異十分廣泛，故 Java 平台在設計時特別重視跨硬體平台的設計。

根據 Java 的設計特點，因此 Java 的架構和以往 C/C++架構有所差異，以下就是 Java 的平台的架構圖[11]：

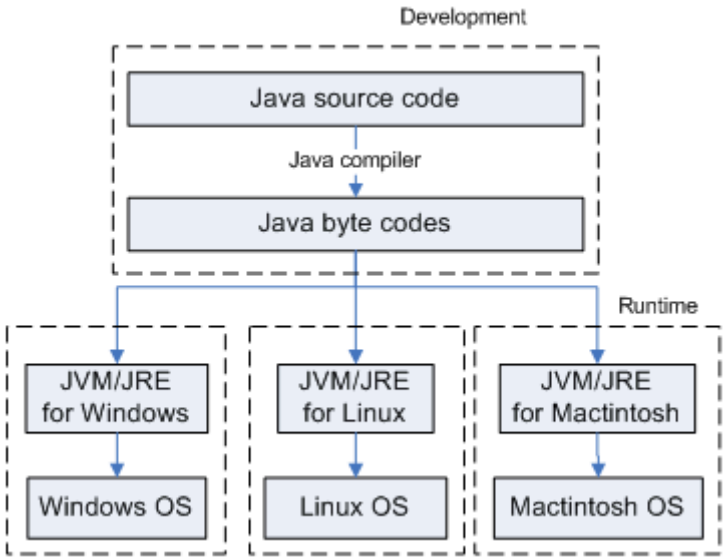


圖 1-4、JAVA 平台的架構圖

程式設計師使用 Java 語言開發 Java 程式，經過 Java 編譯器編譯後，產生 Byte codes 中繼程式碼。Byte codes 透過不同平台上所安裝的 Java 虛擬機器(Virtual machine)解譯後執行。因此 Java 可以達到跨平台目標。

目前 Java 平台在網際網路上的 Web 服務（Web services）和專業解決方案（Enterprise solution），以及小型可攜帶裝置的應用上十分熱門。

1.2.2 Java AWT 概要

AWT 提供了一套抽象的程式庫來建立圖形使用者介面，讓 Java 跨平台的目標可以實現。在 AWT 的幫忙下，程式設計師不需要管不同作業系統所提供的圖形使用者介面原生程式庫如何使用，因為這些工作都交由 AWT 來處理。

AWT 從 JDK 1.0 就已經出現，但是在 JDK 1.1 時有大幅改善，因此目前使用 AWT 開發的程式，大部分都是使用 JDK 1.1 來開發。AWT 所提供的功能，可分成以下幾類[1]：

- Component：
  Component 是構成使用者介面的最基本元件，它實作使用者介面的某個部分，並且可以重複使用。由許多不同種類的 Component 來組成使用者介面。常見的 Component 有 Button、Label、Menu、Window 等等。另外，程式設計師也可以根據 Component 來擴充功能，建立自己的控制項。

- Layout：
  Layout 是用來排列螢幕上的元件。如果沒有 Layout 來管理元件的話，必須要使用絕對座標來決定位置。這點和 Java 平台的跨平台特性有所衝突，因為不同平台的元件的樣子多少有差異，使得程式在各個平台"看"起來不一樣。Layout 就是用來統一管理元件大小與位置的方法。

- Container：
  Container 是一種 Component，它提供一個矩形區域來放置其他的 Component，而這些 Component 會受到 Container 所屬的 Layout 來管理。由於 Container 也是 Component，所以 Container 中可以放置另外一個 Container。這種遞迴架構讓程式設計師可以做到封裝細節的動作，建立更高層級物件。

- Event：
  AWT 採用事件驅動(Event-driven)模式。當事件(像是滑鼠點選、視窗改變大小)發生時，AWT 會幫你處理好相關的動作，自動把事件傳給對事件有興趣的 Handler 來處理。程式設計師只需要跟 AWT 註冊 Handler 對那些事件有興趣就可以了。

- Drawing and Graphic：
  AWT 提供許多基本的繪圖函式(例如：畫線、畫矩形、畫圖等等)，也提供簡單的影像處理函式(縮放、旋轉等等)。

介紹完 Java AWT 的功能後，我們再看看.NET Windows Forms 有那些功能。

1.3 .NET Windows Forms 介紹

Windows Forms 是在.NET Framework 平台上開發圖形使用者介面的套件[9]。Windows Forms 取代了以前在 Windows 作業系統上開發圖形使用者介面套件(MFC 或 Windows SDK)，提供了更多、更方便的功能給程式設計師使用。在 Windows Forms 的幫助下，程式設計師可以完成建立視窗、使用控制項、繪圖、影像處理等等的工作。

1.3.1 .NET framework 平台介紹

.NET Framework 平台是由 Microsoft 公司在 2000 年所提出的，是新一代在 Windows 上開發程式的標準。.NET Framework 平台具有下列幾項特點[7][8]：

- 跨程式語言的開發環境。
- 更方便地移植應用程式到其他平台。
- 強大地與現有 Windows 程式(dll 和 COM 元件)的整合能力。

- 記憶體自動管理。
- 多執行序。
- 例外處理。
- 物件導向程式設計。
- 安全性。

我們再針對前三項在 Java 中沒有的特點加以說明。

首先，.NET Framework 支援許多的程式語言來開發應用程式，像是 C#、VB.NET、J#、C++等等，因此程式設計師可以使用自己所熟悉的語言來開發程式，不需要再學習新的語言。

此外，由於.NET Framework 將程式編譯成中繼碼，執行時再透過虛擬機器來編譯的特性，故只要在非 Windows 的作業系統也安裝 CLR 的話，那麼用.NET Framework 開發的程式一樣可以執行。

因為.NET Framework 本來就是針對 Windows 所設計的，為了讓原先程式設計師開發的程式能繼續被重複使用，所以提供很方便的方式直接呼叫 dll 和 COM 元件所提供的函式。

接下來，來看看.NET Framework 的架構[11]：

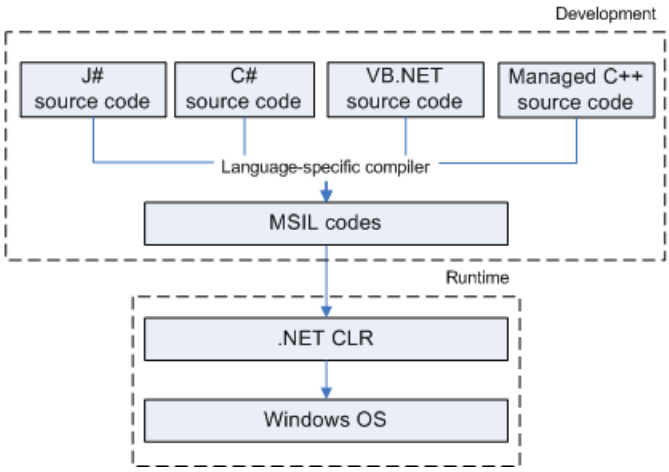


圖 1-5、.NET 平台的架構圖

在這裡可以發現，.NET Framework 平台的架構和 Java 平台十分類似。程式設計師使用 J#、C#等等語言開發程式，經過不同程式的編譯器編譯後，產生 MSIL codes 中繼程式碼。MSIL codes 在執行時透過.NET CLR 利用 JIT(Just-In-Time)編譯技術即時編譯成機器碼後執行。

1.3.2 .NET Windows Forms 概要

Windows Forms 以 Windows 作業系統所提供 Win32 API 基礎，在上面架上一層抽象層，讓程式設計師易於開發圖形使用者介面，不需要瞭解 Win32 API 如何運作。雖然之前已經有 MFC、Visual Basic 等類似的程式庫出現，但是它們各自有各自的開發語言，難以整合。而 Windows Forms 允許與不同語言來開發，讓程式的相容性更高。

Windows Forms 所提供的功能，可分成以下幾類[9][12]：

- Control：
  Control 類似 AWT 中的 Component，是構成使用者介面的最基本元件。Windows Forms 中常見的 Control 有 Button、Label、Form、Panels 等。

- Container：
  Container 類似 AWT 中的 Container，可在其中放置 Control，達到封裝的效果。不過在 Windows Forms 中，Container 不使用 Layout 管理放置在裡面的 Control，而是以絕對座標來定位。

- Event：
  Windows Forms 和 AWT 一樣採用事件驅動模式。當事件發生時，Windows Forms 會把事件傳給對此事件有興趣的程式來執行。

- Drawing and Graphics：
  Windows Forms 中提供 GDI+程式庫，讓程式設計師可以繪製許多基本圖形(線、矩形、多邊形)，以及做影像處理(縮放、變形)的工作。

- Data Binding：
  Control 中的每個屬性(大小、位置)可透過資料繫結的方式設定，資料的來源可以是值的陣列或是存放在資料庫中的資料。Windows Forms 會自動從資料來源抓取資料，設定該 Control 的屬性值。

1.4 本文大綱

本論文的第一章為緒論，簡單介紹背景及 Java AWT 和.NET Windows Forms。在第二章，比較 Java 平台和.NET Framework 平台的差異，並且加以分析。在第三章正式開始探討 Java AWT 移植到.NET Windows Forms 的可行性。在第四章，介紹一個實際從 Java AWT 移植到.NET Windows Forms 的實際個案。提出在移植過程時的問題，並且加以分析。第五章說明結論和未來發展。

第二章　Java 平台和.NET Framework 平台的比較

在本章之中，對於 Java 平台和.NET Framework 平台的相似與相異性，做詳細的介紹與分析。以兩平台的整體架構，分成程式語言(Language)、中繼碼(Intermediate code)和虛擬機器(Virtual machine)三個層次來探討。最後探討圖形使用者介面的可移植性。

2.1 程式語言層的差異

在 Java 平台開發程式，只有 Java 程式語言一種可以選擇；但是在.NET 平台下，可以使用 C#、VB.NET、C++、J#等等的程式語言開發程式。其中 J#程式語言符合 Java 程式語言的語法和語意，並且支援大部分 JDK 1.1 提供的 API[8][15]，因此我們以 Java 和 J#來探討這兩個程式語言有什麼差異。

對於 Java 和 J#的差異，主要討論的要點有三項：

- API(Application Program Interface)，可以使用的程式庫。
- Package，管理類別的方式。
- Javadoc，程式碼註解方式。

我們將在下三節中分別介紹。

2.1.1 API 的討論

如前面提到的，J#支援大部分 JDK 1.1 的 API，以及 Windows Foundation Class (WFC) 程式庫和許多 JDK 1.2 java.util package

內的類別[15](表 2-1)。

AbstractCollection	AbstractList
AbstractMap	AbstractSequentialList
AbstractSet	ArrayList
Collection	Collections
Comparator	ConcurrentModificationException
HashMap	HashSet
Iterator	LinkedList
List	ListIterator
Map	Map.Entry
Set	SortedMap
SortedSet	TreeMap
TreeSet	

表 2-1 、J#支援的 JDK1.2 的 java.util 套件

但是 J#不支援 JDK 1.1[15]中的：

- JNI(Java Native Interface)技術。
- RMI (Remote Method Invocation)技術。
- sun.* 開頭的套件。
- netscape.* 開頭的套件。
- java.lang.SecurityManager。

J#也不支援一些 JDK 1.2 和所有 JDK 1.3 與 JDK 1.4 所支援的 API。因此使用不支援 API 的程式需要經過轉換後，才可以將 Java 平台的程式移植到.NET 平台。

另外，J#可以使用.NET 平台本身所提供的程式庫，因此上述一些不支援的 API 也可以找到相對應的程式來使用。像是不支援的 JNI 技

術可由.NET 平台中的 P/Invoke 技術取代；而 RMI 技術則是由.NET 平台中的.NET Remoting 技術取代。

2.1.2 Package 的討論

Package 是 Java 程式語言中，用來管理相關類別(class)和介面(interface)的管理方法，透過 Package 的管理，程式開發者可以尋找類別、避免名稱衝突和控制可使用的類別。

在 J#中，與 Package 管理方法相同的是 Namespace，Namespace 一樣有上述功能，但是 Namespace 並沒有代表類別檔案實際上放置的路徑[15]。下面舉個例子：

在 Java 的 Package 中的 com.cyc.Hello 這個類別，Hello.java 實際上的存放路徑要是：com\cyc\Hello.java；而 J#中的 Namespace 中的 com.cyc.Hello 這個類別，Hello.java 實際上的存放路徑就不受限制，只要放在程式找得到路徑就可以。

2.1.3 Javadoc 的討論

J#也支援 Java 語言中 Javadoc 的註解格式，以及產生出註解網頁的功能。

2.2 中繼碼層的比較

Java 程式經過編譯後，產生 Byte codes。Byte codes 屬於中繼碼，無法直接執行，要透過 Java 虛擬機器解譯後才能執行。因為有這種設計，才能讓 Java 達成 Write once, run anywhere 的目標。.NET framework 平台同樣採用此種設計概念，J#程式經過編譯之後，不是直接產生執行檔，而是產生 MSIL(Microsoft intermediate language)

中繼檔。

Byte codes 和 MSIL 雖然語法上有差異，不過程式開發者多半不會直接撰寫中繼碼來發展程式，因此可以忽略不談。

Byte codes 和 MSIL 在特徵上很相像，都具備下列項目[5][10]：

- 一組中央處理器(CPU)獨立的指令。
- 堆疊式(Stack-based)程式語言。
- 中繼碼驗證。

Java 平台和.NET Frmaework 平台，存放編譯好的中繼碼的方式不同。在 Java 平台中，Java 語言的存放中繼碼的最小部署單位是 Class，用 Java 開發的程式經過編譯後，會產生 Class 檔案。

而在.NET Framework 平台中，J#語言的存放中繼碼的最小部署單位是 Assembly[7]，用 J#開發的程式經過編譯後，會產生 Exe、Dll 等檔案，而所謂的組件(Assembly)，就是這些一個或多個 Exe 或 Dll 的集合，裡面包含應用程式會用到的程式碼和資源檔，如圖 2-1 所示：

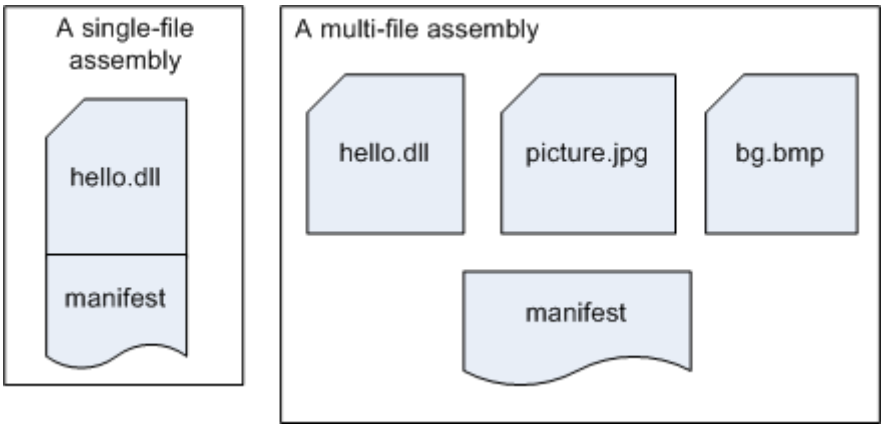


圖 2-1、ASSEMBLY 的類別

因為存放方式的不同，使得兩平台尋找程式庫的方式也不同，將

在下一小節討論。

2.2.1 尋找程式庫方法的討論

在 Java 語言中，CLASSPATH 環境變數用來告訴應用程式要如何尋找程式庫的方法。Java 平台會去 CLASSPATH 所指定的路徑中，找出應用程式會用到的程式庫。

在 J#語言中不支援 CLASSPATH 環境變數，而是採用新的尋找程式庫的方法[15]，分成四個步驟：

1. 檢查組態檔：
   在組件的組態檔中會定義版本、文化等資訊，讓其他應用程式分辨此組件是不是所要尋找的目標。此外，也可以直接定義要在那些路徑尋找組件。

2. 檢查先前參考的組件：
   如果被要求的組件已經在先前的呼叫中尋找過，則此組件已經被載入了，直接使用就可以了。

3. 檢查全域組件快取：
   全域組件快取(Global assembly cache)是.NET 平台用來存放共享組件的快取空間。因此在電腦上執行的.NET 程式，都可以使用全域組件快取內的組件。

4. 透過基礎碼或探查找出組件：
   基礎碼是由組態檔中的<codeBase>項目來提供，<codeBase>指出組件要在那個路徑搜尋的網址(URL)。如果組態檔中沒有基礎碼，則會透過探查(Probing)的方式來尋找。探查的方式會根據以下四個原則[15]：(1)應用程式基底(application base)，為應用程式執行時所在的根位置。(2)文化特性，為參考中組件的文

化特性屬性 (Attribute)。(3)名稱(assembly name)，為受參考組件的名稱。(4)私用 binpath，為根位置之下使用者定義的子目錄清單。

雖然 Java 平台和.NET Framework 平台尋找程式庫的過程不同，但不影響兩平台間的移植性。根據本節的介紹可知，Java 平台和.NET 平台在中繼碼層的差異，不影響移植的可行性。

2.3 虛擬機器層的比較

虛擬機器的功能是用來解譯中繼檔並執行。Java 平台的虛擬機器是 Java virtual machine，而.NET 平台的虛擬機器是 Common language runtime。兩平台的虛擬機器有著相同的特徵[10][11]：

- 載入物件類別(Class loading)。
- 資源回收處理(Garbage collection)。
- 記憶體資料形別安全檢查(Memory type safety checking)。
- 安全性檢查(Security checking)。
- 序列化(Serialization)。

因為虛擬機器層的目的在於執行程式，因此跟移植性比較無關係，故虛擬機器層的移植性不在本論文的討論範圍。

2.4 圖形使用者介面的可移植性

Java AWT 程式庫在 J#中依然被支援，因此原本使用 AWT 開發圖形使用者介面的程式，移轉到 J#後可以直接使用。

不過 J#中的 AWT 元件，與 Windows Forms 的元件不能共用。如圖 2-2 所示，Windows Forms 中的 TreeView 控制項，不能直接加入 AWT

的 Frame 容器中。

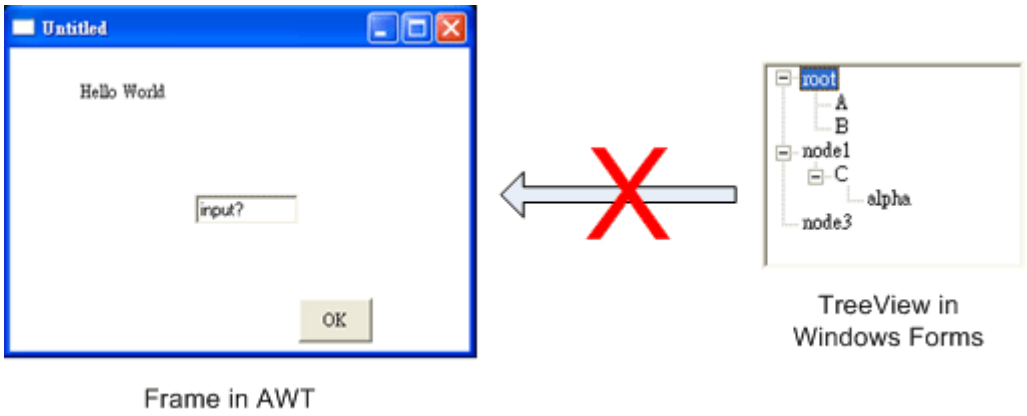


圖 2-2、WINDOWS FORMS 控制項無法加入 AWT 中

因此 AWT 雖然可以直接在 J#中使用，可是無法享受到移轉到 J#後，由 Windows Forms 所提供的各種好處。

一種解決方式是改寫圖形使用者介面的程式，從 AWT 改成使用 Windows Forms。此種解決方式需花費許多時間來完成，並不是一個好方案。

我們希望能儘量減少程式碼的變動，來達成移植的目的。於是本論文設計了一個 Wrapper(命名為 GuiWrapper)，它使用 Windows Forms 程式庫實作出 AWT 的功能。原本使用 AWT 的程式，只要改成使用本論文所提出來的 GuiWrapper，不需要修改許多程式碼，就可以使用 Windows Forms 所提供的功能。

在下一章中，本論文將介紹 GuiWrapper 的設計概念，以及探討可移植性。

第三章　Java AWT 移植到.NET Windows Forms 的可行性

在本章之中，先介紹本論文用來移植 Java AWT 到.NET Windows Forms 的 GuiWrapper，以及討論移植的可行性。再來是介紹 GuiWrapper 的設計，分成 Component 模式和 Event 模式兩部分介紹，並且各自討論該模式移植的可行性。

3.1 GuiWrapper 介紹

本論文為了讓 Java AWT 移植到.NET Windows Forms，發展出 GuiWrapper。GuiWrapper 是使用 Windows Forms 的技術，把 AWT 實作出來(圖 3-1)，原本使用 AWT 開發圖形使用者介面程式的程式設計師，只要改成使用 GuiWrapper 的程式庫，就可以使用 Windows Forms 的技術來顯示圖形使用者介面。

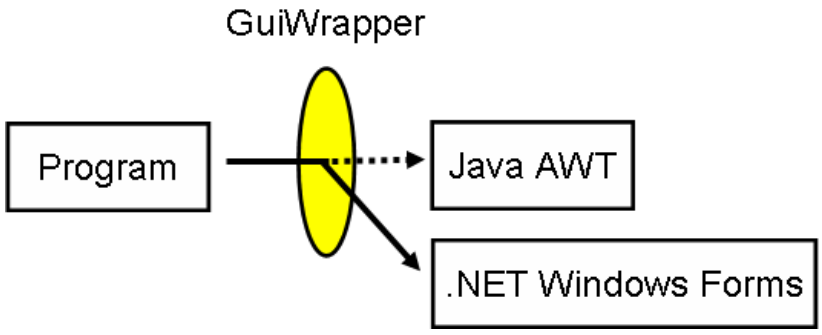


圖 3-1、GUIWRAPPER

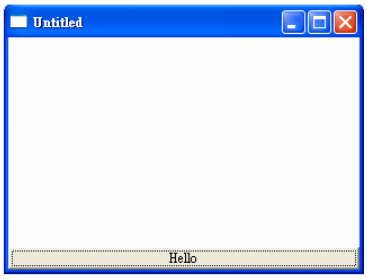
接下來，先介紹如何使用 GuiWrapper 的範例，再探討使用 GuiWrapper 移植的可行性。

3.1.1 GuiWrapper 範例

接下來，我們來看一個實際上使用 GuiWrapper 的範例。圖 3-2 是一個原本使用 AWT 的程式，執行後顯示 AWT 的視窗：

```
import java.awt.*;


public class MyFrame extends Frame {
   pubilc MyFrame() {
      Button b = new Button("Hello");
      add(b, BorderLayout.SOUTH);
      setSize(320, 240);
      show();
   }
   public static void main(String[] args) {
      new MyFrame();
   }
}
```



Java AWT Frame

圖 3-2、使用 AWT 的程式碼與執行結果

然後，程式設計師只需要將

```
import java.awt.*;
```

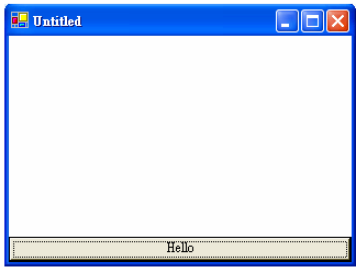
的敘述，改成

```
import com.cyc.GuiWrapper.awt.*;
```

後，原本使用 AWT 的程式，就會改成使用 GuiWrapper，呼叫 Windows Forms 來執行。圖 3-3 是修改過後的程式，執行後顯示 Windows Forms 的視窗：

```
//import java.awt.*;
import com.cyc.GuiWrapper.awt.*;

public class MyFrame extends Frame {
   pubilc MyFrame() {
      Button b = new Button("Hello");
      add(b, BorderLayout.SOUTH);
      setSize(320, 240);
      show();
   }
   public static void main(String[] args) {
      new MyFrame();
   }
}
```



.NET Windows Form

圖 3-3、使用 GuiWrapper 的 AWT 程式碼與執行結果

由上例可見，使用 GuiWrapper 可以非常簡單地將程式從 AWT 移植到 Windows Forms。

不過，不是每一個 AWT 的程式在移植時都如此順利，有些 AWT 程式使用 GuiWrapper 後，執行結果和原本有些差異；有些則是需要修改程式碼後，才能使用 GuiWrapper 移植。因此，本論文接著探討使用 GuiWrapper 的移植性。

3.1.2 移植的可行性

使用 GuiWrapper 可將 AWT 移植到 Windows Forms。但不是所有程式都能完美移植，根據每個程式的移植成果，我們把使用 GuiWrapper 的移植可行性分成四個層次：

1. 完全可移植(Completely portable)：
   原本使用 AWT 的程式，不需要修改程式碼，便可以使用 GuiWrapper 移植到 Windows Forms。移植後的執行結果與原本完全一樣。我們稱此程式為完全可移植。

2. 強可移植(Strongly portable)：
   原本使用 AWT 的程式，不需要修改程式碼，便可以使用 GuiWrapper 移植到 Windows Forms。但是移植後的執行結果與原本相似，比如說圖形使用者介面上的控制項功能和原本相同，但是控制項的外觀(大小、位置)可能會有所差異。我們稱此程式為強可移植。

3. 弱可移植(Weakly portable)：
   原本使用 AWT 的程式，必須遵循我們提出的指導方針，才可以直接使用 GuiWrapper 移植到 Windows Forms。否則需要修改程式碼才能使用 GuiWrapper 移植。我們稱此程式為弱可移植。

4. 不可移植(Not portable)：
   原本使用 AWT 的程式，不管如何修改程式，都無法使用 GuiWrapper 移植到 Windows Forms。我們稱此程式為不可移植。

3.2 GuiWrapper 設計

由於 GuiWrapper 需要把 AWT 的功能實作出來，於是我們先分析 AWT 提供了那些功能。AWT 所提供的功能，可分成下列三大類[1]：

- Component：
  各種構成圖形使用者介面的控制項，像是 Label、Button、TextFeild、Frame、Panel 等等。

- Graphics：
  各種繪圖與影像處理的功能，像是繪圖、繪製圖形(矩形、直線等等)和繪製文字。

- Event：
  各種程式執行時會發生的事件，像是 MouseEvent、KeyEvent、FocusEvent 等等。

GuiWrapper 根據上述功能，以 Component 模式和 Event 模式來設計整體架構。我們將在下兩節中，探討這兩個模式的移植性。

3.2.1 Component 模式

控制項是圖形使用者介面最基本的元件，我們在 Component 模式中討論和控制項有關的設計，一共分成 Component hierarchy、Layout manager 和 Graphics 三部分。接下來，我們分別探討這三部分的可移植性。

3.2.1.1 Component Hierarchy

在 AWT 中，控制項採用 Component 和 Container 的架構設計[1]。

Component 是構成使用者介面的最基本元件，它實作使用者介面的某個部分，並且可以重複使用。由許多不同種類的 Component 來組成使用者介面。

Container 是一種 Component，它提供一個矩形區域來放置其他的 Component，由於 Container 也是 Component，所以 Container 中可以放置另外一個 Container。這種遞迴架構讓程式設計師可以做到封裝細節的動作，建立更高層級物件。

AWT 採用 Component 和 Container 的架構(圖 3-4)，建立圖形使用者介面。常見的 Component 有 Label、Button、TextField 等等。另外，程式設計師也可以根據 Component 來擴充功能，建立自己的控制項。而常見的 Container 則有 Frame、Panel 等等。

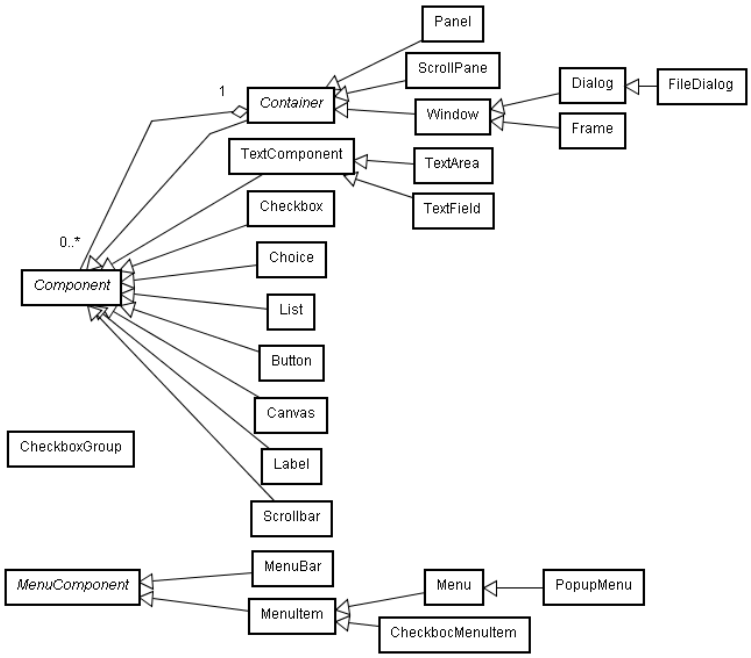


圖 3-4、AWT 的 COMPONENT HIERARCHY

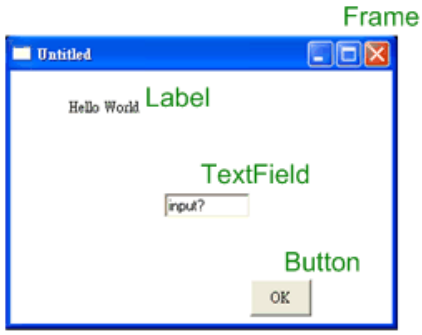


圖 3-5、使用 AWT 開發的圖形使用者介面

Windows Forms 一樣採用 Component 和 Container 的架構(圖 3-6)。在 Windows Forms 中的 Component 稱做 Control，常見的 Control 有 Label、Button、TextBox 等等。另外，程式設計師也可以根據 Control 來擴充功能，建立自己的控制項。而 Windows Forms 中的 Container 也是 Control，不過完整的功能是由 ContainerControl 提供，常見的 ContainerControl 有 Form、Panel 等等。

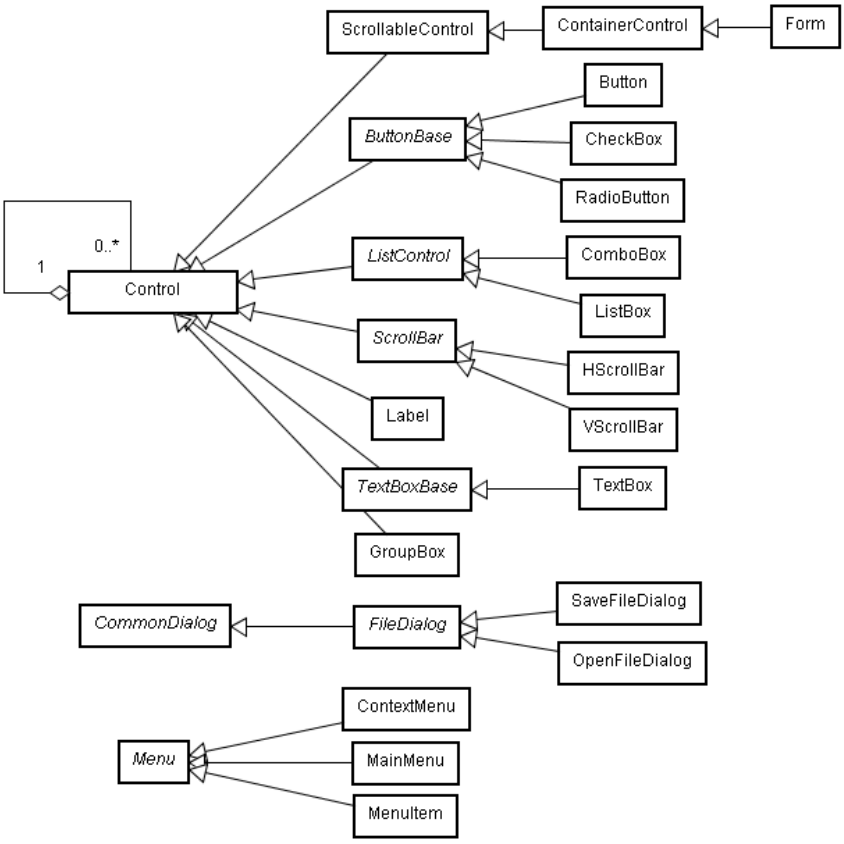


圖 3-6、WINDOWS FORMS 的 COMPONENT HIERARCHY

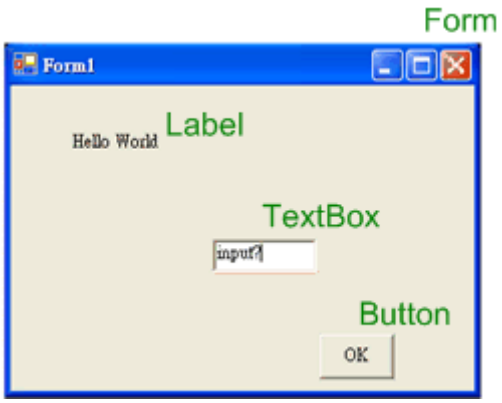


圖 3-7、使用 WINDOWS FORMS 開發的圖形使用者介面

由圖 3-4 和圖 3-6 可發現，AWT 中的控制項都可以在 Windows Forms 所提供的控制項中找到類似的。因此 GuiWrapper 在 Component hierarchy 的設計，就把 AWT 和 Windows Forms 的控制項作對應(圖 3-8)，用 Windows Forms 的控制項，把每個 AWT 的控制項實作出來。

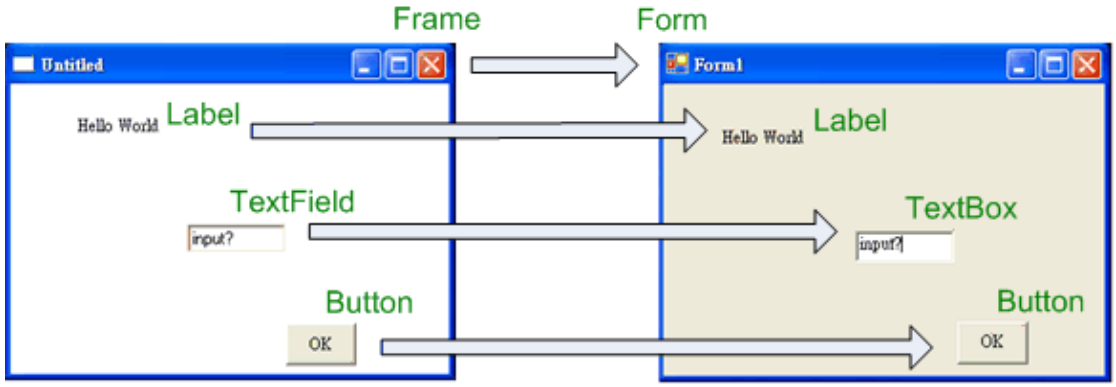


圖 3-8、AWT 和 WINDOWS FORMS 的控制項對應

根據上述的設計，GuiWrapper 的 Class diagram 如圖 3-9 所示，在 com.cyc.GuiWrapper.awt 套件中，仿造 AWT 的 Class diagram，宣告物件類別以及繼承關係。每個 Component 中，有一個變數存放該 Component 所對應的 Windows Forms 類別。每個 Component 所提供的方法，會使用 Windows Forms 提供的功能實作。

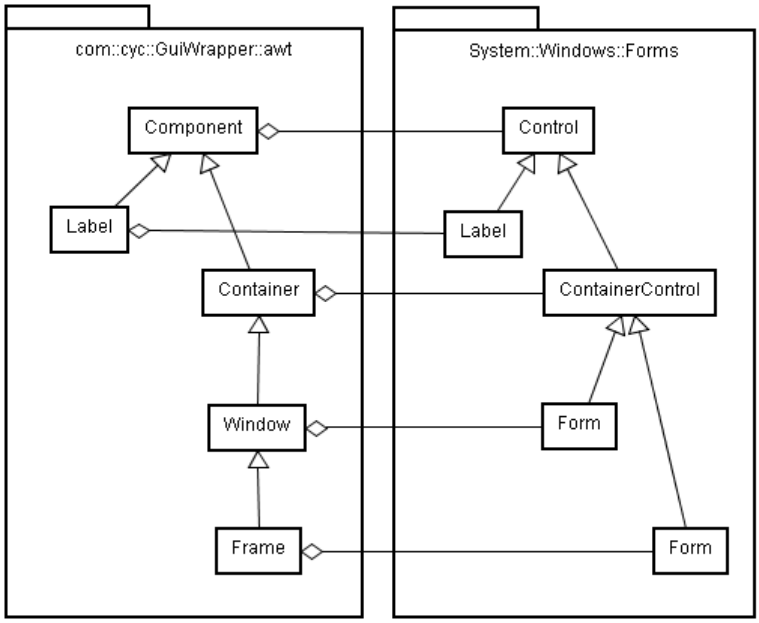


圖 3-9、GUIWRAPPER 的 CLASS DIAGRAM - 1

表 3-1 列出 AWT 和 Windows Forms 的控制項如何對應：

AWT	Windows Forms
Button	Button
Canvas	Control
Checkbox	CheckBox RadioButrton
CehckboxGroup	GroupBox
CheckboxMenuItem	MenuItem
Choice	ComboBox
Component	Control
Container	ContainerControl
Dialog	Form
FileDialog	SaveFileDialog OpenFileDialog
Frame	Form
Label	Label
List	ListBox
Menu	MenuItem
MenuBar	MainMenu

MenuItem	MenuItem
Panel	ContainerControl
PopupMenu	ContexMenu MenuItem
Scrollbar	VScrollBar HScrollBar
ScrollPane	ContainerControl
TextComponent	TextBox
TextField	TextBox
Window	Form

表 3-1 、AWT 和 Windows Forms 的控制項對應表

GuiWrapper 在 Component hierarchy 上，採用 Windows Forms 的控制項對應到 AWT 的控制項的設計，解決了 Component hierarchy 的問題。由於所有 AWT 的控制項，都可以被 Windows Forms 的控制項對應。因此，Component hierarchy 使用 GuiWrapper 移植的可行性是：完全可移植。

3.2.1.2 Layout Manager

放置在 Container 中的控制項會受到 Layout manager 的管理[1]，隨者 Container 大小的改變，跟著變更大小，讓圖形使用者介面的外觀看起來依然相同，不會改變原本設計的風格。有了 Layout manager 的幫助，圖形使用者介面便可有一致性。

AWT 中提供 BorderLayout、FlowLayout、GridLayout、CardLayout 和 GridBagLayout 五種 Layout manager[1]來管理 Container 中的控制項排列顯示。

Windows Forms 使用絕對位置排列顯示控制項，再加上 Docking 和 Anchoring 兩個屬性[9]。Docking 的功能類似 AWT 中的 BorderLayout，但是 Windows Forms 控制項沒有最佳大小的屬性，因

此無法直接使用 Docking 的功能取代 BorderLayout。Anchoring 的功能和 AWT 提供的五種 Layout manager 完全不同，因此無法使用。

為了解決這個問題，GuiWrapper 實作 AWT 中的五種 Layout manager(圖 3-10)，讓程式設計師可繼續使用 Layout manager。並且模擬 AWT 中驗證 Layout 的流程，讓 Layout manager 能正確運作。

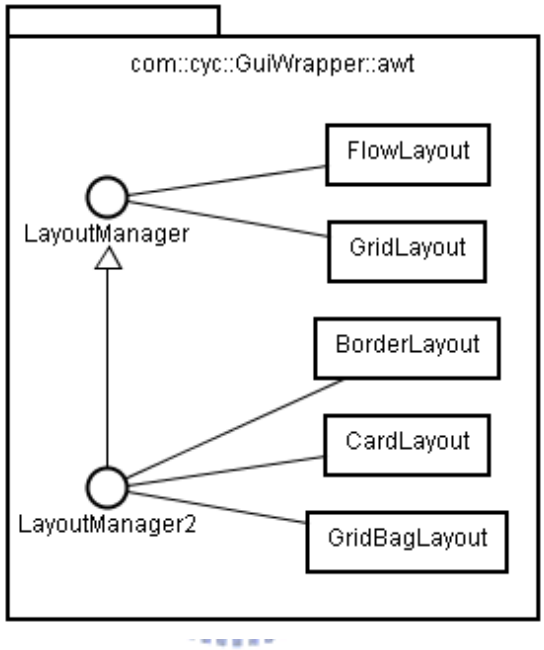


圖 3-10、GUIWRAPPER 的 CLASS DIAGRAM - 2

不過由於 Layout manager 在排列時，會詢問控制項的最佳大小，來設定新的大小。但 Windows Forms 的控制項的最佳大小和 AWT 的控制項的最佳大小，在有些控制項會有所不同，所以使用 GuiWrapper 移植後的圖形使用者介面在執行時可能會和之前有所不同。因此 Layout manager 使用 GuiWrapper 移植的可行性是：強可移植。

3.2.1.3 Graphics

Graphics 是控制項中用來繪圖的功能，例如繪製圖檔、繪製圖形(線、矩形等等)和繪製文字。此外，還提供一些影像處理的功能。

在 AWT 中，Graphics 的繪圖功能分散在 Image、Graphics、Font 中，而影像處理的功能則是在 java.awt.image 套件內[1]，程式設計師就使用這些功能來處理 Graphics。

在 Windows Forms 中，Graphics 的功能由 GDI+來提供，包含了繪圖和影像處理等功能[13]。GDI+所提供的功能都在 System.Drawing 套件中。

為了解決 Graphics 的移植問題，GuiWrapper 在設計上，使用 GDI+來把 AWT 提供的繪圖功能實作出來(圖 3-11)。至於影像處理功能，GuiWrapper 新增了 DotnetImageProducer 和 DotnetImageConsumer 兩個物件類別，這兩個類別使用原本 AWT 中處理影像的方式。因此影像處理的功能直接使用原本 AWT 提供的就可以了。另外，FontMetric 物件類別仍然使用原本 AWT 所提供的 FontMetric 物件類別。

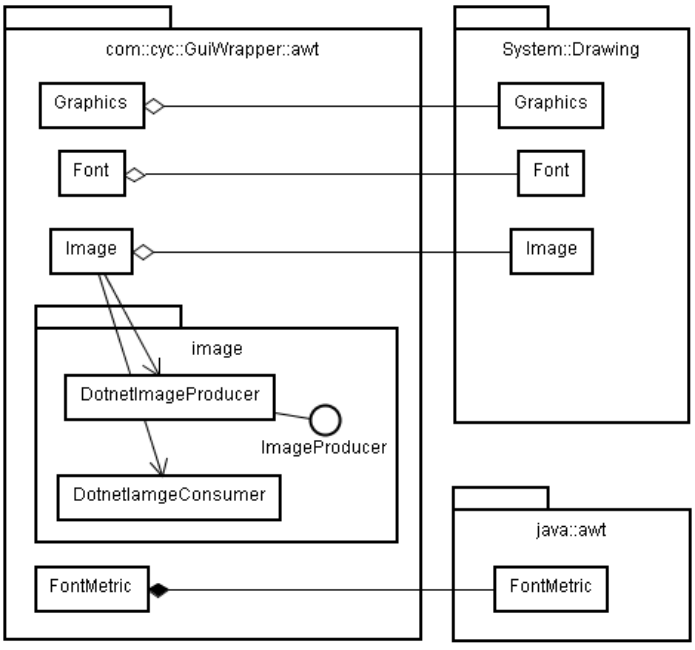


圖 3-11、GUIWRAPPER 的 CLASS DIAGRAM - 3

由於 AWT 所提供的 Graphics 功能可以使用 GDI+實作，因此 Graphics 使用 GuiWrapper 移植的可行性是：完全可移植。

3.2.2 Event 模式

開發圖形使用者介面的應用程式時，事件驅動是一個非常重要的觀念。當一個事件(像是滑鼠點選、鍵盤被按下一個鍵等等)發生，執行環境會產生一個事件，然後傳送給程式去處理，讓程式能和使用者互動。GuiWrapper 在 Event 模式的設計，一共分成通用 Event 模式和 Painting 模式兩部分。接下來，我們分別探討這兩部分的可移植性。

3.2.2.1 通用Event模式

在 AWT 中，一樣採用事件驅動模式來處理事件。不過 JDK 1.0 和 JDK 1.1 的事件驅動模式有所不同，根據第一章的介紹，我們的移植是以 JDK 1.1 為主，因此 AWT 的事件驅動模式以 1.1 為標準來移植。

AWT 1.1 中的事件驅動模式稱作 Delegation，是以 Observer 設計模式加上事件處理的設計[1]，如圖 3-12 所示。其中 Event source 會產生某種類型的事件(Event)，而對某個事件有興趣的物件稱作 Listener。

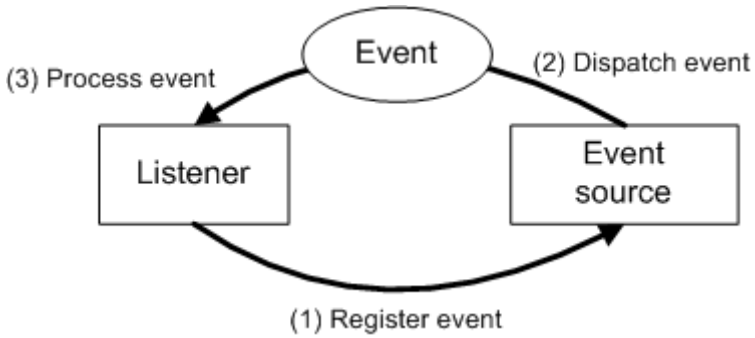


圖 3-12、AWT 1.1 的事件驅動模式

首先 Listener 會跟 Event source 註冊，表示對某個 Event 感到興趣。接者 Event source 發生事件，把事件傳送給對此事件有興趣

的 Listener。Listener 接到事件後，呼叫負責處理該事件 Handler 來處理。

圖 3-13 是 AWT 中處理按鍵控制項收到滑鼠點選事件的實際範例。首先，ActionListener 呼叫 Button 的 addActionListener()方法註冊，然後當 Button 接收到滑鼠點選的事件，產生出一個 ActionEvent 給 ActionListener，最後 ActionListener 的 actionPerformed()方法負責處理 ActionEvent 的相關後續動作。

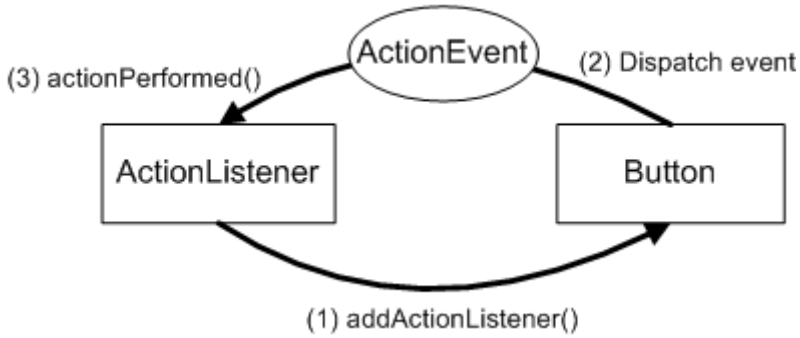


圖 3-13、AWT 處理事件的範例

瞭解 AWT 如何處理事件後，再來看 Windows Forms 如何來處理事件。

在 Windows Forms 中，和 AWT 一樣採用了 Delegation 方式處理事件。如圖 3-12，Windows Forms 中一樣有 Event source、Event 和 Listener 這三種類別來完成事件驅動的處理。

圖 3-14 是 Windows Forms 中處理按鍵控制項收到滑鼠點選事件的實際範例。首先 Form(也就是 Listener)呼叫 Button(也就是 Event source)的 add_Click()方法註冊，並且設定 Form 中的一個方法為 Callback 方法，然後當 Button 接受到滑鼠點選的事件，產生 EventArgs 物件(也就是 Event)來存放滑鼠點選的事件，最後 Form 中的 Callback 方法負責處理 EventArgs 的相關後續動作。

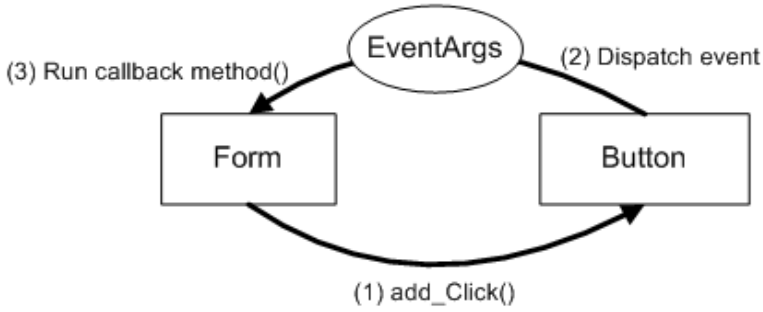


圖 3-14、WINDOWS FORMS 處理事件的範例

瞭解 AWT 和 Windows Forms 對於事件的運作模式之後，發現他們的處理方式十分相像。因此 GuiWrapper 在事件模式的移植上的設計，仿造 AWT 事件處理的方式，只是將註冊事件的動作重導給 Windows Forms 去註冊；並且當 Windows Forms 發生事件後，把 Windows Forms 的事件，重新組織成 AWT 的事件，傳送給 AWT 的 Listener 去執行。

圖 3-15 是 GuiWrapper 如何處理按鍵控制項收到滑鼠點選事件的範例。

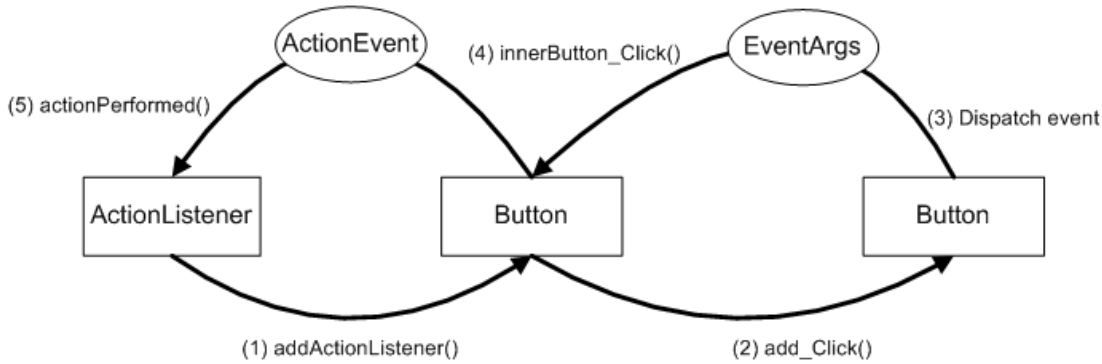


圖 3-15、GUIWRAPPER 處理事件的範例

跟圖 3-13 和圖 3-14 不同的地方在於，GuiWrpapper 中的 Button 會向 Windows Forms 的 Button 再註冊一次事件，讓 Windows Forms 知道 Button 需要滑鼠點選的事件。當 Windows Forms 收到滑鼠事件後，將事件傳給 GuiWrapper 的 Button，GuiWrapper 的 Button 再把事件包裝成 AWT 的事件，傳送給 AWT 的 Listener。

GuiWrapper 在處理事件的設計，以圖 3-16 的 Sequential diagram 表示，此 Diagram 為處理滑鼠點選按鍵控制項的處理。

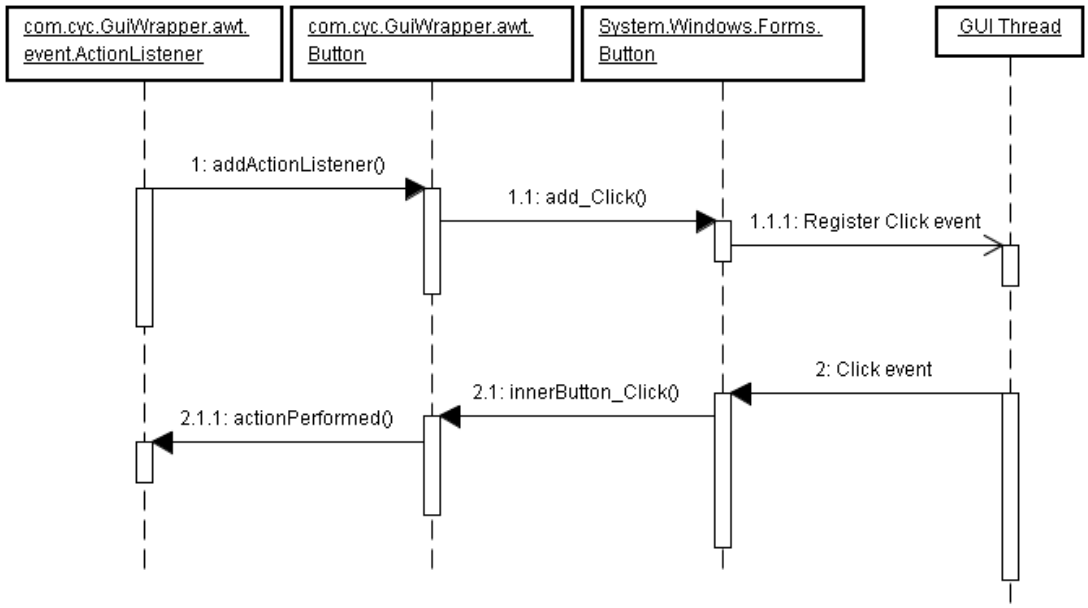


圖 3-16、GUIWRAPPER 處理事件的 SEQUENTIAL DIAGRAM

表 3-2 則是 GuiWrapper 如何對應 AWT 的事件註冊方法和 Windows Forms 事件註冊方法的列表。

AWT	Windows Forms
Button::addActionListener()	Button::add_Click()
Checkbox::addItemListener()	CheckBox::add_Click()
CheckboxMenuItem::addItemListener()	MenuItem::add_Click()
Choice::addItemListener()	ComboBox::add_SelectedIndexChanged()
Component::addComponentListener()	Control::add_Resize() Control::add_Move() Control::add_VisibleChanged()
Component::addFocusListener()	Control::add_GotFocus() Control::add_LostFocus()
Component::addKeyListener()	Control::add_KeyDown() Control::add_KeyUp() Control::add_KeyPress()
Component::addMouseListener()	Control::add_MouseDown()

	Control::add_MouseUp() Control::add_MouseEnter() Control::add_MouseLeave()
Component::addMouseMotionListener()	Control::add_MouseDown() Control::add_MouseUp() Control::add_MouseMove()
Container::addContainerListener()	ContainerControl::add_ControlAdded() ContainerControl::add_ControlRemoved()
List::addActionListener()	ListBox::add_DoubleClick()
List::addItemListener()	ListBox::add_SelectedIndexChanged()
MenuItem::addActionListener()	MenuItem::add_Click()
ScrollBar::addAdjustmentListener()	VScrollBar::add_Scroll() HScrollBar::add_Scroll()
TextComponent::addTextListener()	TextBox::add_TextChanged()
TextField::addActionListener()	TextBox::add_KeyDown()
Window::addWindowListener()	Form::add_Actived() Form::add_Closed() Form::add_Closing() Form::add_Deactive() Form::add_Load() Form::add_SizeChanged()

表 3-2、AWT 和 Windows Forms 的事件註冊方法對應表

GuiWrapper 在通用 Event 模式上，採用將 AWT 的事件對應到 Windows Forms 的事件註冊的設計，解決了通用 Event 模式的問題。由於所有 AWT 的事件(除了 Paint 事件之外，在下一節會探討)，都可以被 Windows Forms 的事件對應。因此，通用 Event 模式使用 GuiWrapper 移植的可行性是：完全可移植。

3.2.2.2 Painting模式

在GuiWrapper的設計下，AWT的事件模式可以透過Windows Forms的幫忙，得以正常運作。不過在 AWT 中，唯一有一種事件不是遵循之前的 Event-listener 的模式來處理的，那就是 Paint 事件[14]。我

們先介紹 AWT 中如何處理 Paint 事件的 Painting 模式，以及問題所在。

AWT 中的 Paint 事件分成兩種：

- 系統產生(System-triggered)：
  由系統發出的 Paint 事件，像是控制項第一次顯示在畫面以或控制項改變大小，此時該控制項都需要重新繪製。

- 應用程式產生(App-triggered)：
  由應用程式發出的 Paint 事件，當程式設計師需要改變控制項顯示資料時，發出此事件讓控制項重新繪製。

當程式在執行時，Paint 事件會由不同方式產生，存放在事件佇列內，等待被處理(圖 3-17)。

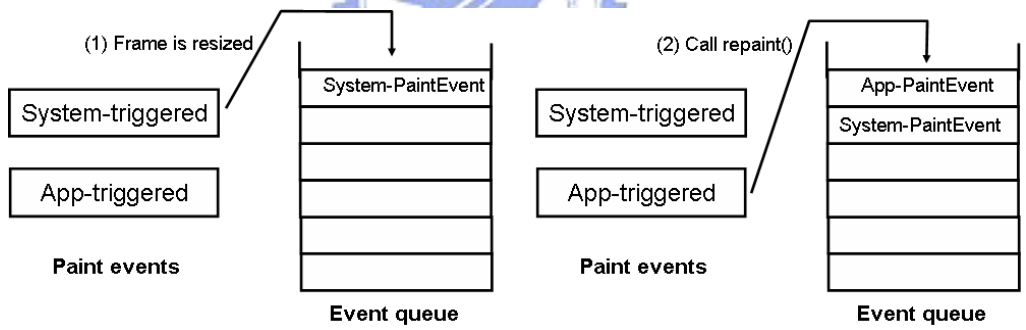


圖 3-17、產生 PAINT EVENT

UI 執行緒則會定時去事件佇列中取得事件來處理。由於 Paint 事件分成兩種來源，因此 UI 執行緒取得 Paint 事件時會根據來源，分別呼叫 Component 的 paint()方法或 Component 的 update()方法(圖 3-18)。

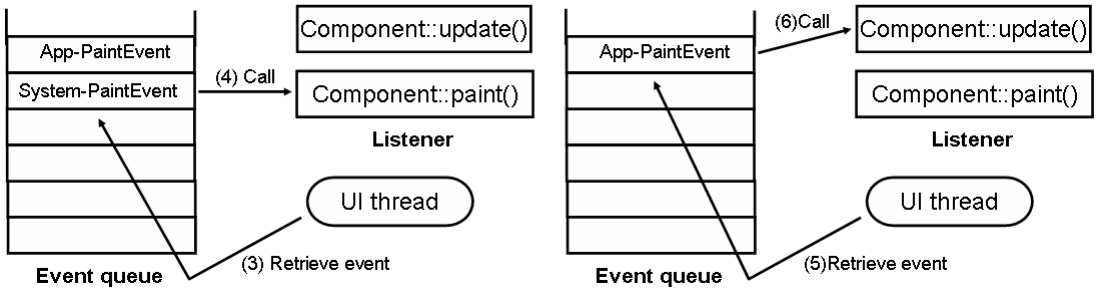


圖 3-18、處理 PAINT EVENT

因此程式中如果要處理 Paint 事件的話，必須覆寫 Component 中的 paint()或 update()這兩個方法，並沒有任何 Listener 可以用來註冊 Paint 事件。接著詳細介紹 paint()方法和 update()方法：

- paint()方法：
  負責繪製控制項。當發生系統產生的 Paint 事件時被呼叫。

- update()方法：
  負責更新控制項。當發生應用程式產生的 Paint 事件時被呼叫。預設的功能是先清除控制項背景，然後呼叫 paint()繪製控制項。

在 Windows Forms 中，要處理 Paint 事件有兩種方式。一種是遵循 Event-listener 模式，Listener 對控制項註冊發生 Paint 事件後要被呼叫的 Callback 方法，當發生 Paint 事件時呼叫 Callback 方法。不過此種方式無法取代控制項原本預設的繪圖動作。

另一種方式和 AWT 類似，當發生 Paint 事件時，Windows Forms 會先呼叫控制項的 OnPaintBackground()方法，接著呼叫 OnPaint()方法。接著詳細介紹 OnPaintBackground()方法和 OnPaint()方法：

- OnPaintBackground()方法：
  負責繪製控制項的背景。當發生 Paint 事件時首先被呼叫。

◆ OnPaint()方法：

負責繪製控制項。當發生 Paint 事件時接著 OnPaintBackground()被呼叫。

根據上面介紹，可以發現在 Windows Forms 中處理 Paint 事件的方法，在語意上來探討，OnPaintBackground()和 OnPaint()兩個方法加起來只有對應到 AWT 中的 paint()方法，並沒有任何一個方法對應到 update()方法。因此 GuiWrapper 設計時為了解決這個問題，提出了下列的解決方案(圖 3-19)：

◆ 覆寫 OnPaintBackground()方法：

讓 OnPaintBackground()方法架空，不作任何事情。避免控制項的背景重複繪製。

◆ 覆寫 OnPaint()方法：

讓 OnPaint()方法 呼叫 update()方法。

由於 update()方法在 AWT 中預設實作是呼叫 paint()方法，因此不管發生系統產生的 Paint 事件或是應用程式產生的 Paint 事件，update()方法和 paint()方法都會被依序呼叫。

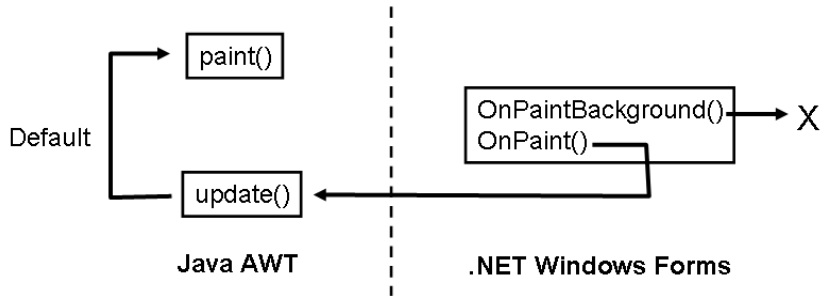


圖 3-19、GUIWRAPPER 在 PAINT EVENT 的解決方案

因為 GuiWrapper 採用此解決方案，因此原本使用 Java AWT 開發圖形使用者介面程式的程式設計師，必須遵守下列指導方針，才能讓

程式移植順利：

- 不要覆寫 update()方法。

GuiWrapper 在 Painting 模式上的設計，使得程式設計師需要遵守指導方針，才可以將 Painting 模式移植到 Windows Forms 之後，得到與之前處理的相似過程。由於程式設計師為了遵循指導方針，有可能會需要修改程式碼。因此，Painting 模式使用 GuiWrapper 移植的可行性是：弱可移植。

在本節的最後，把使用 GuiWrapper 的可移植性，做個總結：

- Component 模式：
  - Component hierarch 的可移植性是：完全可移植。
  - Layout manager 的可移植性是：強可移植。
  - Graphics 的可移植性是：完全可移植。
- Event 模式：
  - 通用 Event 模式的可移植性是：完全可移植。
  - Painting 模式的可移植性是：弱可移植。

第四章　實例探討

在本章之中，先介紹準備使用 GuiWrapper 來作移植的範例：本實驗室所開發的 CYC 遊戲平台客戶端的圖形使用者介面。接著討論使用 GuiWrapper 將 CYC 遊戲平台客戶端的圖形使用者介面從 Java AWT 移植到.NET Windows Forms 的可移植性。最後介紹 CYC 遊戲平台客戶端，從 Java 平台移植到.NET Framework 平台的其他問題。

4.1 介紹 CYC 遊戲平台客戶端的圖形使用者介面

CYC 遊戲平台[17][18][19]是本實驗室所開發，提供許多傳統的桌上遊戲(例如：象棋、跳棋、橋牌等等)，讓使用者可以透過網際網路，與其他使用者一起玩遊戲。

而 CYC 遊戲平台客戶端的圖形使用者介面(圖 1-1，畫面取自 CYC 遊戲大聯盟網站[16])，則是用 Java AWT 開發，以 Java applet 形式執行。使用者直接使用瀏覽器就可以透過 CYC 遊戲平台玩遊戲，省去下載程式安裝的動作。

目前為了繼續開發 CYC 遊戲平台客戶端的圖形使用者介面，遇上了三個需求：

1. 需要更換 Java 虛擬機器：
   CYC 遊戲平台客戶端是在微軟提供的 Java 虛擬機器上執行，而微軟的 Java 虛擬機器 2007 年 12 月 31 日終止支援，因此需要考慮更換 Java 虛擬機器的問題。

2. 需要內嵌網頁在圖形使用者介面中：
   CYC 遊戲平台因為是網際網路遊戲，和其他遊戲網站的合作自然密切。在客戶端的圖形使用者介面中，需要可以顯示網頁來增加

遊戲性(玩家人頭改成用 Avatar 系統)，或是替其他網站宣傳。

3. 需要增強繪圖效能：
   目前 CYC 遊戲平台客戶端的圖形使用者介面內，遊戲畫面以靜態為主，日後將增加越來越多動畫效果，因此繪圖效能需要增強，最好能使用 DirectX 技術。

而本論文所提出的 GuiWrapper，正好可解決 CYC 遊戲平台客戶端的圖形使用者介面的需求二和需求三，因此 CYC 遊戲平台客戶端就從 Java 平台移植到.NET Framework 平台，並且使用 GuiWrapper，將圖形使用者介面從 AWT 移植到 Windows Forms。

4.2 移植 CYC 遊戲平台客戶端的圖形使用者介面

這節中先討論 CYC 遊戲平台客戶端的圖形使用者介面使用 GuiWrapper 的可移植性，並且分析使用 GuiWrapper 的效率。最後探討 CYC 遊戲平台移植到.NET Framework 平台的其他問題。

4.2.1 使用 GuiWraper 移植圖形使用者介面的可行性

在第三章提到中使用 GuiWrapper 的可移植性，因此使用 GuiWrapper 移植 CYC 遊戲平台客戶端的圖形使用者介面，一共分成五項來探討：

- Component hierarchy：
  CYC 遊戲平台客戶端的圖形使用者介面使用的所有 AWT 控制項，GuiWrapper 都有支援，因此都可以透過 GuiWrapper 完整移植到 Windows Forms。故 CYC 遊戲平台客戶端的圖形使用者介面在 Component hierarchy 方面可以順利移植。

◆ Layout manager：

CYC 遊戲平台客戶端的圖形使用者介面有使用到 AWT 所提供的 Layout manager，因此圖形使用者介面使用 GuiWrapper 後，畫面上控制項的配置，和原本 AWT 時有些不同，但是實際上的差異不大。圖 4-1 是移植後的圖形使用者介面，跟移植前的圖形使用者介面(圖 1-1)的差異，只有左下角下拉式方塊的底色(顏色較深)，以及右下角的按鍵大小(尺寸較小，略擠)，其餘畫面皆相同。故 CYC 遊戲平台客戶端的圖形使用者介面在 Layout manager 方面可以順利移植。

圖 4-1、使用 GUIWRAPPER 移植後的圖形使用者介面(橋牌)

◆ Graphics：

CYC 遊戲平台客戶端的使用者介面所使用的繪圖與影像處理功能，GuiWrapper 都有支援。故 CYC 遊戲平台客戶端的圖形使用者介面在 Graphics 方面可以順利移植。

◆ 通用 Event 模式：

CYC 遊戲平台客戶端的使用者介面所使用的事件(除了 Paint 事件之外)，GuiWrapper 都有支援，並且處理流程和 AWT 時完全一樣。故 CYC 遊戲平台客戶端的圖形使用者介面在通用 Event 模式方面可以順利移植。

◆ Painting 模式：

CYC 遊戲平台客戶端的使用者介面遵循 GuiWrapper 的指導方針，在處理 Paint 事件上，雖然覆寫了 update()，但是 update()的功能只有呼叫 paint()，因此 Paint 事件處理流程跟 AWT 時一樣。故 CYC 遊戲平台客戶端的圖形使用者介面在 Painting 模式方面可以順利移植。

根據上面的探討，CYC 遊戲平台客戶端的圖形使用者介面，只要使用 GuiWrapper 就可以從 AWT 移植到 Windows Forms 上。

4.2.2 使用 GuiWraper 移植圖形使用者介面的效率

上一節中提到，使用 GuiWrapper，可以順利將 CYC 遊戲平台從 AWT 移植到 Windows Forms。本節中要來探討，使用 GuiWrapper 移植所花費的時間，比直接修改程式碼來移植，可以快多少。

CYC 遊戲平台客戶端的圖形使用者介面，以橋牌遊戲為例，跟畫面有關的物件類別，總共使用到 204 個 Java 檔案，79738 行的程式碼。這些程式碼，就是當移植到 Windows Forms 時，所需要修改的程式碼。

如果是採用直接修改程式碼來移植到 Windows Forms，至少需要把所有使用到 AWT 的物件類別，轉換成使用 Windows Forms。從程式碼分析，最少需要修改 14544 行。如果是採用 GuiWrapper 來移植到 Windows Forms，原本物件類別所 import 的套件，要從 AWT 改成 GuiWrapper。從程式碼分析，一共需要修改 308 行。

因此，使用 GuiWrapper 移植的效率比直接修改程式碼，還要快上 47.2 倍(14544/308)。

4.2.3 移植 CYC 遊戲平台的其他問題

在第二章中有介紹 J#非常適合用來將 Java 開發的程式，移植到.NET Framework 上。所以，CYC 遊戲平台客戶端使用 J#來作移植的工作。但是 J#中不支援一些原本 Java 可以使用的 API，使得 CYC 遊戲平台客戶端在移植到.NET Framework 平台遇到三個問題：JNI(Java Native Interface)、Resource 和 Javascript。在下面的小節中，我們將分別介紹如何處理這三個問題。

4.2.3.1 JNI

CYC 遊戲平台客戶端原本使用 JNI 技術，呼叫 Windows 作業系統提供的 Win32 API 來播放音效。移轉到.NET Framework 後，根據第二章的介紹，需要改用 P/Invoke 技術取代 JNI 技術。

因為 P/Invoke 可以直接呼叫 Win32 API，不必像 JNI 一樣得自己撰寫 C 語言的 DLL 後再呼叫 Win32 API，因此改用 P/Invoke 比使用 JNI 還方便呼叫 Windows 作業系統提供的功能。

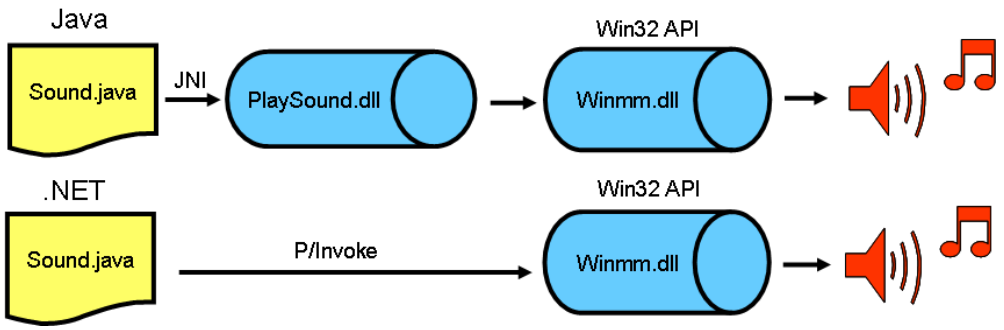


圖 4-2、使用 JNI 技術和 P/INVOKE 技術的比較

4.2.3.2 Resource

CYC 遊戲平台客戶端原本使用不同語言的資源檔，讓圖形使用者介面可以根據使用者習慣的語言，選擇顯示該語言的畫面。移轉到.NET Framework 後，資源檔的格式和 Java 使用的不一樣，需要作轉換。.NET Framework 本身就提供了轉換資源檔格式的程式碼，只要照著用就可以了。

4.2.3.3 Javascript

CYC 遊戲平台客戶端原本使用 Java applet 執行，因此可以使用 Javascript 和瀏覽器溝通。CYC 遊戲平台客戶端利用這項優點，開發了一些功能，像是使用 Javascript 另外開啟一個瀏覽器視窗，顯示使用者的聊天歷史。

移植到.NET Framework 後，CYC 遊戲平台客戶端是以應用程式的方式執行，無法使用 Javascript。CYC 遊戲平台客戶端的解決方法則是在圖形使用者介面中，內嵌了一個瀏覽器，Javascript 就交由內嵌的瀏覽去執行。此內嵌的瀏覽器，也讓 CYC 遊戲平台客戶端可以應用的功能更加豐富。

根據 4.1 和 4.2 節的探討，發現在 GuiWrapper 的幫助下，CYC 遊戲平台客戶端可以順利地從 Java AWT 移植到.NET Windows Forms。而且不需要修改太多的程式碼，便可以享受到移轉到.NET 平台後的好處。

第五章　結論與未來展望

本論文探討從 Java AWT 移植到.NET Windows Forms 的可移植性，並且提出 GuiWrapper 的解決方案。在 GuiWrapper 的幫助下，AWT 的程式，只需要修改 import 的套件，從原本的 java.awt 改成 com.cyc.GuiWrapper.awt，便可以移植到 Windows Forms。

使用 GuiWrapper 的可移植性，分成 Component 模式和 Event 模式兩部分探討：

- Component 模式：
  - Component hierarch 的可移植性是：完全可移植。
  - Layout manager 的可移植性是：強可移植。
  - Graphics 的可移植性是：完全可移植。

- Event 模式：
  - 通用 Event 模式的可移植性是：完全可移植。
  - Painting 模式的可移植性是：弱可移植。

目前 GuiWrapper 已經將大部分 AWT 的功能使用 Windows Forms 實作出來，本論文實際將 CYC 遊戲平台客戶端的圖形使用者介面從 Java AWT 移植到.NET Windows Forms，成果十分不錯。

未來 GuiWrapper 的研究方向，將針對目前不足的 Layout manager 和 Painting 模式問題，提出更好的解決方案。以及在 GuiWrapper 增加 Windows Forms 的特有功能(像是 AWT 中沒有的控制項)，讓程式設計師可以直接使用 GuiWrapper 來增強圖形使用者介面的功能。還有將 DirectX 技術整合在 GuiWrapper 的 Graphics 中，讓程式設計師不需要改寫程式碼，就可以使用 DirectX 技術增強圖形使用者介面的效能。

參考文獻

[1] John Zukowski, Java AWT Reference, O'Reilly, 1997.

[2] Mary Campione, Kathy Walrath, Alison Huml, Java Tutorial, Third Edition: A Short Course on the Basics, Addison Wesley, December 2000.

[3] David Flanagan, Java in Nutshell, Fourth Edition, O'Reilly, March 2002.

[4] Sheng Liang, The Java Native Interface, Addison Wesley, 1999.

[5] James Gosling, Henry McGilton, "The Java Language Environment", Sun Microsystems, 1996.

[6] David S. Platt, Introducing Microsoft .NET, Third Edition, Microsoft Press, 2003.

[7] Hoang Lam, Thuan L. Thai, .NET Framework Essentials, 3rd Edition, O'Reilly, August 2003.

[8] John Sharp, Andy Longshaw, Peter Roxburgh, Microsoft Visual J# .NET, Microsoft Press, September 2002.

[9] Matthew Adams, Ian Griffiths, .NET Windows Forms in a Nutshell, O'Reilly, March 2003.

[10] Jawahar Puvvala, Alok Pota, .NET for Java Developers: Migrating to C#, Addison Wesley, July 2003.

[11] Heng Ngee Mok, From Java to C#: A Developer's Guide, Addison Wesley, January 2003.

[12] Chris Sells, Windows Forms Programming in C#, Addison Wesley, August 2003.

[13] Mahesh Chand, Graphics Programming with GDI+, Addison Wesley, October 2003.

[14] Sun Microsystems, "Java Development Kit 1.1.8", available from http://java.sun.com/products/archive/jdk/1.1/index.html, 1999.

[15] Microsoft Corporation, "Visual J# in MSDN Library", available from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vjsharp/html/vjoriintroducingvisualjnet.asp, 2004.

[16] 群想網路科技, " CYC 遊戲大聯盟的橋牌遊戲", available from http://cycgame.com/, 2004.

[17] I-Chen Wu, "Internal Design Specification for the CYC System", Internal Document, December 1997.

[18] I-Chen Wu and Cheng-Da Shen, "The Game Developer Guide for the CYC system Version 2", Internal Document, December 1997.

[19] I-Chen Wu, "The CYC System", available from http://dist5.csie.nctu.edu.tw/cyc/index.html, 2004.