國立交通大學

資訊科學與工程研究所

碩士論文

適合嵌入式平台 Java ME CLDC 的 Java AWT 的設計

Design of Java AWT for Java ME CLDC Platforms

研 究 生：白家芸　　Student：Chia-Yun Pai

指導教授：蔡淳仁　　Advisor：Chun-Jen Tsai

中 華 民 國 101 年 7 月

中文摘要

就目前的趨勢，Java 嵌入式環境是未來的重要的發展。隨著數位化、自動化裝置越來越先進，圖形使用者介面的發展也要隨之跟進，讓科技能進步不會只限於在研究領域，更能讓一般大眾能享受到其中的便利與好處。結合這兩大方向，相信在 Java 嵌入式環境中開發 Graphics User Interface(GUI)系統到未來都會是不斷探討的主題。在分析 JavaME CDC/PBP 的 GUI 系統，Java AWT 中，我們參考並保留大部分原有的應用程式介面，設計並提出適合在 Java ME CLDC 環境中的 GUI 系統，命名為 MMES AWT。相較於 JavaME CDC/PBP 的 Java AWT，MMES AWT 的 class 集合較為簡化、需要系統 class 的支援也較少，native function 由 KNI 實作，需要 native graphics library 的支援較簡單。並提供 thread-safe 機制確保 Event handling 運作安全。目前支援基本的輸出輸入功能。實驗結果在畫面輸出上細微處差異，其他大致相同甚至精緻一點點。輸入的部分也可以正確偵測到滑鼠、鍵盤的訊息。MMES AWT 是一個較為簡化的 GUI 系統，方便移植、實作之外，之後也可以為基礎，深入的發展成其它的用途，例如 3D 圖形庫等等。

Abstract

As the trend, Java embedded system is more and more important evolvement. Graphics user interface(GUI) is accompanied with this trend. Advance technology is not common to the public, and it's not really convenient in practice. Combined above two directions, GUI system in Java embedded environment is a growing research topics in the future. According to Java AWT in JavaME CDC/PBP，we simplified the Java AWT classes, and remains most part of API, and designs a GUI system for JavaME CLDC, named MMES AWT. Compared with Java AWT in JavaME CDC/PBP, MMES AWT has less classes, less system classes supporting, KNI implementing native functions, native graphics library supporting simpler, and provides thread-safe mechanism. MMES AWT experiments two basic GUI system functions, displaying and accepting mouse, keyboard messages. In experiment results, two functions successfully implement and displaying almost looks same with PBP AWT. MMES AWT is a basic, simple GUI system. It's convenient to implementing in other way or porting to other platform. And it is also a basis for advance development, ex 3D GUI.

誌謝

在研究所這四年的時間裡，我非常感謝我的指導教授--蔡淳仁老師，在研究上教導與耐心的陪伴。這份論文能得以產生，是老師不斷給我機會，耐心教導與不放棄，才讓我經歷這麼多失敗與挫折後，能再度恢復信心完成。過程中體會到研究的精神，就是為了更多人的利益，不只是為了滿足自己的好奇心而已，還有心態對了，才會甚麼都對。

此外，也要謝謝交大電控所博班的學姐楊曉如學姊，在碩三一度念到恐慌症發作，幾乎快放棄學業時，帶我一起禪修禪行，從原本失望、對未來充滿絕望的深淵中，有勇氣能回來面對一切，重新開始，完成學業不只是為了學歷，更是很多人對我的鼓勵與陪伴，讓我才能有勇氣完成，最後還要謝謝實驗室其他同學的互相討論與幫忙，讓我在這四年的時間學到很多，不僅在學業上還有許多人生的智慧。最後還要謝謝我的家人在各方面對我的支持，沒有他們，我也不能這樣無後顧之憂的完成學業，太感恩這四年給我的一路的點點滴滴，總總的遭遇讓我能遇見人生最棒的禮物。

。

目錄

表格目錄

圖目錄

第一章 導論

1.1 研究動機

現在人類對電腦不在只是技術上，甚至視覺化需求也越來越多了。所以任何UI(User Interface)的發展也越來越蓬勃發展。現今的智慧型手機之所以造成一股搶購風潮，就是人機介面的設計越來越貼近人的直覺，直覺想做的事可以越來越不再受限機器或是電腦的接收方式。從這一點中可以看到人最原始的渴望，就是只要動念就可以做到任何事，像擁有魔法一樣，擁有無所不能的能力可以簡單用直覺就能做到了。現今科技也逐漸往這個方向邁進，讓人類對能擁有無所不能的超能力的嚮往與想像不再是停留在腦海中或電影裡，而是有可能實現在生活中的。這樣的趨勢下，追求 UI 越來越直覺化與電腦體積越來越小，功能越來越來強大必定將是未來科技的兩大發展方向。

依照上述的趨勢下追求 UI 改良變成很重要的一項技術。從以前到現在 UI 的演變可以大概分為兩個階段。一個在電腦發展的初期，希望在不需要有電腦背景知識的前提下，可以直接理解與操作的 UI，例如作業系統的發展，早期的 DOS 系統、Linux，需要下系統指令才能操作，現在都演變成有圖形介面的 Windows、Linux 等等，利用滑鼠或鍵盤即可操作。另一個是近五年以來，當電腦科技越來越蓬勃發展也越來越普遍時，電腦的運作邏輯功能越來越有強大，越來越能符合人類更高層次的需求。UI 也需要變得越來越符合人的思考模式了，使人直接用平常生活經驗即可操作或體驗，例如現在的智慧型手機不需要滑鼠用手指觸碰即可，放大或縮小也只用手指滑動：還有越來越熱門的 3D 互動介面...等等。

以 GUI 設計的歷史而言，第一個階段重點是放在 GUI 如何在電腦螢幕上實現出來，如 GUI 的視窗選單等設計，以及如何利用電腦接收輸出或輸入資料的能力來實現人機的互動。第二個階段是將電腦中的呈現與輸出資訊的方式變成最接近人類經驗的方式，例如將 UI 變成 3D (three-dimension)化，讓人可以在空間中直接感覺到

UI 與資訊的真實感直接操作，這方向的研究重點就在於如何呈現 3D 的顯示、以及操作者的輸入行為的偵測(例如，人用手指頭點選虛擬 3D 的物件，該要怎麼偵測是哪一個物件…等)。

在 UI 要不斷進步的趨勢之前，最重要的還是執行的速率要快，不然再好的 UI 的功能，也因為太慢而被淘汰。除以之外，電腦、設備體積也要越來越輕便才行，所以提升電腦單位體積的計算能力是科技的一大方向。讓電腦設備的體積要越來越小、運算能力要越來越強大，變成是所有現今科技發展的基礎，有了這樣的基礎才能真正得以實現所有技術的進步，為人類帶來種種的好處與方便。這也是目前嵌入式環境的開發越來越受到重視的原因。

隨著嵌入式平台越來越被重視的發展，跨平台的技術是非常重要的。未來嵌入式系統的應用程式開發環境會越來越普遍，也會公開化，提供給第三方廠商或個人使用，開發不再只是保留在製造端或是少數人才能進行的。不管是在開發的應用程式上或是選擇的程式語言上，跨平台的特性都是必然的優勢，所以選擇 Java 的執行環境來做開發環境。為了加強 Java 的執行效能，各種不同的平台有不同的加速方式，常用的技術有 Just-in-Time (JIT)的純軟體加速或是利用硬體設計 Java 加速器。針對低功耗的即時運算，用硬體來實現 Java 的加速計算，還是比較理想的選擇。然而，一般 Java 的 UI 程式庫，如 AWT 及 Swing，往往有許多針對特定平台的實作設計，常常會犧牲了移植性或效能

1.2 研究目的

本論文的目標是打算設計一套小型的 AWT 相容的 Java 程式庫，可以很方便的移植到具有硬體加速功能的 Java 平台之上。為了達到最低的軟硬體相依性，我們選擇以 Java 的嵌入式平台 Java ME CLDC 系統 classes 的一個子集合，做為我們設計的 AWT 程式庫的基礎。而我們採用的方法，則是從較雜的 Java ME CDC 的 Java AWT 開始，再根據 Java ME CLDC 的環境進行修改簡化，以設計出新的 Java AWT 程式庫。由於 GUI 最終的輸出是跟特定平台相關，因此在 Java 程式庫的最底層還

是要依賴一些 native functions。有別於一般的實作上是採用了比較大型的 graphics libraries (如 X 或 MicroWindows)，我們是自行設計了較小型的 native graphics library 來達到這個目的。

1.3 論文架構

本論文共分成五章，第二章相關背景探討會介紹現今 Java 嵌入式環境中的使用者圖形介面的研究，還有 Java 平台的介紹。第三章是分析由 Sun Microsystems 所開發的 Java ME/CDC, Personal Basis Profile (PBP) 中的 Java AWT 的設計。第四章是根據第三章的分析結果來設計 MMES AWT，並以 Java ME/CLDC 為底層系統實作出一個易移植的程式庫，第五章是驗證我們實作的 MMES AWT 的正確性。最後，第六章是結論與未來展望。

第二章 相關背景

Java 從一開始為了方便移植到不同平台所發想的跨平台語言，到後來廣泛被採用為不同系統的應用程式開發語言，不管是在大型的企業上、還是個人的用途上、還是簡單自動化的機器上，都存在有 Java 的身影。可以從中看到 Java 研發了各種技術都是為了能夠在第一時間符合最新技術的需求。Java 一直都是想要設計讓人更便利、更簡單可以實作的程式語言。Java 讓程式設計師可以避掉記憶體的計算與指標的麻煩，必且將人類思考的方式融入在程式語言中，大想法中包含小想法，不斷層層包裝、深入的概念，像物件導向一樣，先由大觀念的印象，不斷深入往下直到小細節，這樣子程式語言的概念，加強了程式設計師設計的概念。在思考軟體架構時也能從一開始比較高角度去看，了解了彼此互動的關係之後，確定之後，再去設計與衍生更小的環節。這樣將每一層的工作都能彼此分開與更專業化，這也是物件導向的程式概念重要的地方。Java 也用了層層包覆的概念，將系統能力與功能需求由大到小分成三種 Java EE、Java SE、Java ME。Java EE 是由 Java SE 所延伸出來的，功能範圍最大，Java ME 是由 Java SE 精簡之後又增加少許功能而來的，功能範圍最小，Java SE 居中。三個平台能力範圍的關係如圖 1。Java EE 是為了提供比較複雜的商業需求所存在的平台，需要消耗較多資源、運算能力較高的系統運作，執行的程式有可能須由一個以上的電腦來共同完成。Java SE 則是提供一般使用桌上型的電腦需求的平台，通常都是個人電腦、單機作業。Java ME 則是提供一般資源比較有限的嵌入式環境，例如手機、衛星導航系統等等。

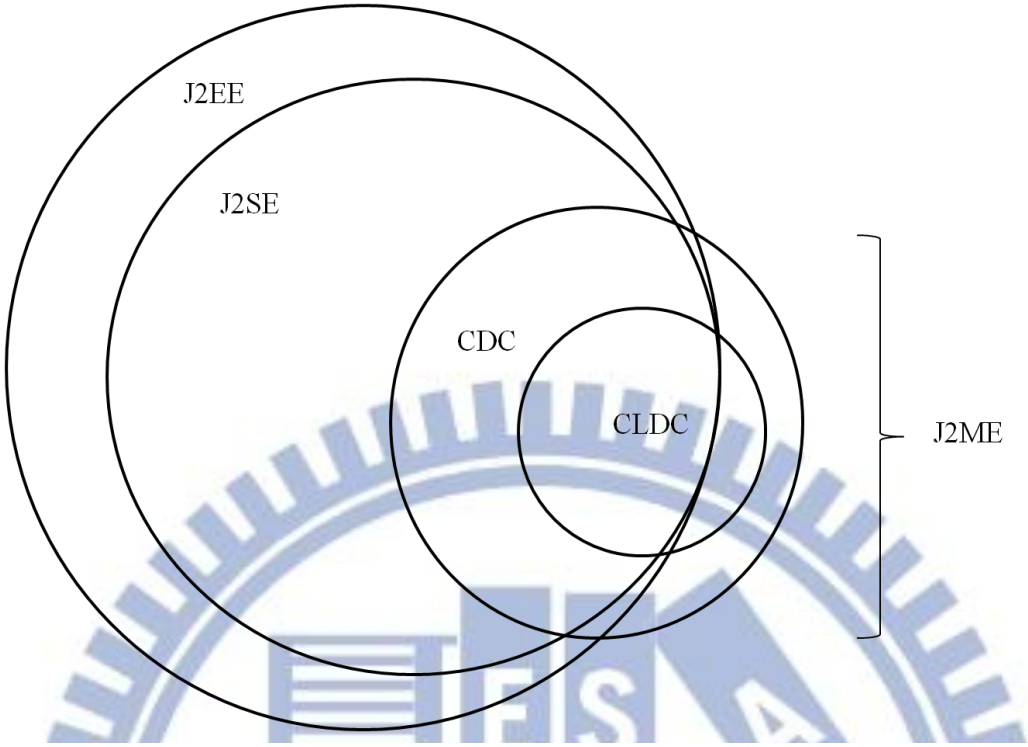


圖 1 Java 的平台功能範圍關係

功能性範圍最大 Java EE 主要是在商業上的用途，需要的功能有保密的機制，還有資料庫的管理以及網頁伺服器的需求..等，需要比較高效率與運算能力強的硬體，也需要比較多元化功能的技術，例如網頁、資料庫...等，Java EE 平台中就有許多技術來處理這些，Java SE 則是一般桌上型電腦的能力可執行的範圍，大部分符合個人基本需求與網路功能為主。Java ME 是為了符合嵌入式系統所存在的，因為有許多不同用途的裝置與不一樣運算能力的設備，將系統的架構分成了 Configuration 和 Profile 這兩層的觀念。兩種 Configuration 根據嵌入式環境能力的不同如表格 1，

	CDC	CLDC
電源	無限制	大多為電池
網路功能	支援網路連線	有限網路功能
記憶體	最小512KB ROM 256KB RAM	開發Java程式記憶體160KB～512KB

表格 1　CDC 與 CLDC 需要最小環境能力

分為 CDC(Connected Device Configuration)和 CLDC(Connected Limited Device Configuration)。

Profile 的概念是嵌入式環境根據特別功能，提供不同的系統 class，例如 MMAPI (multimedia API)支援串流影像與聲音[15]、MIDP(Mobile Information Device Profile)支援可以網路連線行動裝置[16]、WMA (wireless messaging API)支援無效網路能力[17]等等。除此之外，也可以 Profile 為基礎，開發其他需要的功能，[21]中以 Java ME Basis Personal Profile 為基礎，開發一套簡單的 MHP(Multimedia Home Platform)和圖形的系統。MHP 是啟動 DVB 應用程式的一個標準。Xilinx ML310 為執行此系統的硬體環境。執行環境包含一個嵌入式 Linux、一個以 frame-buffer 為基礎的 X server，還有 Java ME Personal Basis Profile (Java ME PBP)的環境。為了要執行 MHP 的影片功能，整合了 FFmpeg 程式庫和 JavaTV JMF Lite 提供分工與產生一個 MPEG-2 影像。此外，還將影像與圖形做整合。以 Java ME PBP 為參考的軟體大量的修改與優良化，在 MHP 編譯的過程中，將影像物件與 AWT 元件做結合。這個系統已經成功地提供 AWT 圖形與 JMF 影像顯示服務，包括顯示，任意切換、放大縮小、定位 MPEG-2 影像。

在嵌入式的環境中使用 GUI 一直都是熱烈討論的話題。因為 GUI 需要大量的畫圖動作與互動速度的要求，所以環境需要大量運算的能力與有效率的執行，所以對於嵌入式環境或運算能力比較有限的裝置來說，是一個影響效能很重要的因素，如果有網路傳輸的設備，彼此傳送 GUI 資訊更會造成延遲，所以如何能避免複雜的運算或是簡化 GUI 的架構，提升效能是非常重要的。G.Canfora, G.D Santo,E. Zimeo 針對可移動性裝置環境的限制，提出一種減少運算量、提高效率的做法，以 thin-client 模組為基礎開發一個軟體框架 Thin-Client aPplications for limiTed dEvices (TCPTE) [7]，這個軟體框架 TCPTE 支援在資源有限的移動裝置上開發 thin-client 應用程式[9]，而且這個軟體架構還考慮到易移植性。可將桌上型的應用程式移植到新的個人無線網路裝置的環境上，不需要做修改，自動將桌上型應用程式的 Java AWT 架構改成適合網路傳輸的 TCPTE AWT 架構。使用 TCPTE，可以在伺服器執行 Java 應用程

式和在遠程客戶端顯示 AWT 的介面。TCPTE 結合了 thin-client 的運算和 client-server 豐富的圖形介面，並且允許程式設計師開發 PWD 應用程式時，使用通常在開發桌面型應用程式時相同的程序與工具即可。

在嵌入式環境中的 GUI 除了運算的速率還要考慮很重要的一點，能夠跨平台、易移植性。在[12]中，試圖改善在嵌入式系統中 Java AWT 的可移植性。一個新的圖形庫比現在的 AWT 更容易移植並且能縮短移植時間的。提出的新圖形庫概念由兩層組成。底層定義一組最小圖學原型的集合，是需要被移植的，其他 AWT 需要的原型被分組到上層。實作此圖形庫來支援 AWT 在兩種平台上。一種是純軟體實作[18]，另一種是在 Java 處理器環境中[19]。前者雖然執行效能並不太實際，但後者相對起來有良好的改善，兼顧新的圖形庫的跨平台性與執行速度。

因為 GUI 需要的運算能力及架構複雜度較高，甚至在改良 Java 的嵌入式環境時，不把 Java AWT 整合到系統 class 中。在 [11]中，為了減少 Java 系統(包括系統 class、Java 虛擬機、OS)所佔的記憶統空間，所以在系統 class 中不考慮支援 Java AWT，並且設計一個非常輕、非常簡單、消耗支援最少的 OS。這此處就可以看到 Java AWT 或是 GUI 功能的軟體，的確在資源有限的環境裡，不管是資源的消耗上、執行效能上都是一個非常急需重視的問題。

除了在效率上的提升，GUI 的設計也一直是每個程式設計師努力的重點，因為好的 GUI，讓使用者使用器來方便、習慣、喜歡操作應用程式，才能真正享受到技術的好處，不然空有好的技術，卻不好操作，就無法發揮最大效益了。GUI 的元件、架構、提供的任何機制都深深影響著容不容易操作。除此之外，程式設計師也要方便撰寫，方便設計版面。所以 GUI 的架構一直以來都是很複雜、很難分析、很難明顯易懂的，要改良或設計 GUI 的架構就需要透過設計模式來協助。在 GOF [8]的書中，將程式行為歸納出 23 種設計模式。透過了解 GUI 的設計模式，將程式了解層次提升到設計層次。當然可以去閱讀相關資訊來了解並歸納出它的設計模式，但是最實際、最有效益的做法就是能偵測並回復一個程式中的設計模式，就能省下許多歸納與整理的時間可以直接進行分析與設計。所以對於設計好的軟體與 GUI 系統，

偵測設計模式的方法與工具很重要，要能提供完整與準確的資訊。在[10]論文中顯示了一個新的、完全自動化的設計模式偵測方法。這個方法基礎是對於 GOF 內容的重新分類。根據 GOF 的特徵分類使設計模式偏向在正向工程(設計程式者的立場)，這方法中重新的分類對於回復工程的(分析程式者的立場)是更好，更適合的。方法是使用簡單的、靜態的程式分析技巧來擷取程式內容。這篇論文中也建立一個新的工具，PINOT，用來實踐此方法。PINOT 可以偵測所有 GOF 的模式根據具體的定義產生的程式結構或系統行為。除此之外，還比原本存在的偵測工具，更快，更具體，偵測更多模式。PINOT 已經成功的偵測 Java AWT, Jhotdraw, Swing, Apache ant 和許多其他程式和程式庫頁的設計模式。

回復設計模式除了可以了解程式的架構或行為也可以了克服現有可擴展性問題。因為有許多軟體在維護上或擴充性的問題都來設計模式與實作上的考慮進行變種所造成的。所以要能進行理想的設計模式偵測，有時可能因為設計模式變型了，在實際的原始碼中無法完整偵測到，在[14]提出一個做法，藉由半自動化地辨識設計模式的物件來協助設計復原和理解程式。以往的作法是在沒有人工的干預之下一次試著去分析一個可能很大的軟體系統。這樣顯得比較沒有效率而且可能無法考慮到設計師的設計經驗下的做法。提出的一個新的辨識演算法在這地方進行改善。新的演算法利用區域和上下文的資訊，由一個反向工程師(程式分析者)和一個特別的底層資料結構，即是一種特殊形式有註解的抽象語法圖來完成辨識。透過人工的輔助更能偵測設計模式實際被運用的情形。在 Java AWT 和 JGL 的程式庫中採用此方法得到相當數量的評價。

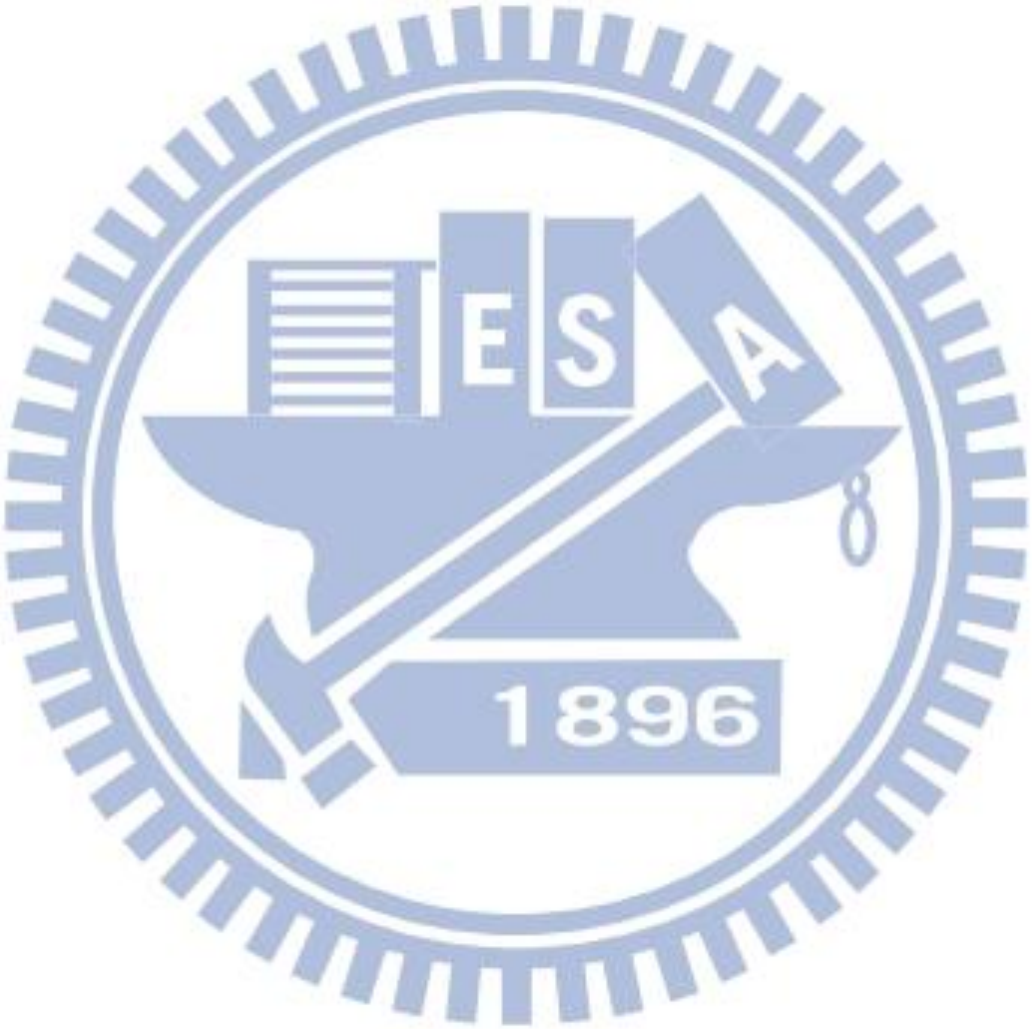
設計GUI的觀感也是非常重要，在[13]中討論一些關於GUI設計工具的發展工作。這些工作是著重在元件的佈局設計，在特定的介面指引確認之下，可以給了使用者介面一個明確、一貫的外觀。論文中提出的介面生成器提供一個環境，在 Java 的 applet 和應用程式中可以指定介面元件和布局的規範。這工具使任何了解 Java AWT 語法的人可以建立高質感的介面。整合了一般設計原則並且提供一個簡單使用的設計規範環境來達到此目的。這樣做可以提醒程式設計師在設計 GUI 時必須注意的細

節，不只是考慮到程式設計師個人的經驗而已，能更周全考慮到一個使用者的立場。

透過設計模式的確可以了解 GUI 架構，但對於 GUI 程式在執行過程中掌握度還是很低，若出現錯誤，往往很難從結果去推論，若能觀察整個程式執行過程，對於 GUI 的偵錯有很大的幫助。[20]描述的是設計和實作一個新的視覺互動的 Java 執行環境。這個系統展示了執行中物件的結構和物件內部細節和 method 執行情形。執行狀態的表示是建立在一個新的但簡單的表示技術，這技術會根據物件實際情況來分類，支援所有 Java 的特性，包括繼承、內部 class、靜態 method 等等。所有 GUI 的元件來自 Java swing 或是 Java AWT 可以被視覺化仰賴定義的執行狀態。這邊也有做回復之前執行狀態的機制，可以讓使用者回去看之前執行的狀態。這個視覺化系統比較著名的特性是它可以被應用在Java 虛擬機上，不需要特別的Java編譯器。一個新的前置處理器(原始碼對原始碼轉換)被運用在與Java撰寫的動態媒體器結合，用來實現視覺化動作。

在嵌入式平台中最受矚目的不外乎是手機平台。Google 在歷經幾年之後、耗費大量資金與心力，研發了手機作業系統 Android。Android 一推出，在業界受到前所未有的熱潮。它是採用 open source 的 linux 作業系統、然後底層用 c 語言、應用層使用 Java 語言。Android 的架構是有四層[22]，Application、Application framework、Library 與 Android runtime、Linux kernel。其中 Application framework 層是將應用程式所需的服務簡化成元件(Component)，所有的應用程式都由這些服務所組成。在 Android 裡的提供 GUI 服務是 Views 這個 class，與 Java AWT 不同的是 View class 為基本元件，並且將使用者會用到的 GUI 功能標準化。功能選單(Menu)就提供三種類型，Option Menu、Context Menu、Sub Menu，每個 Menu 選項排版都已固定好，只要撰寫其功能就可以。還有對話方塊(Dialog)也提供四種，AlertDialog、ProcessDialog、DatePickerDialog、TimePickerDialog。最後是提示訊息(Toast)。相同的是都有提供事件處理模式。另一個不同的是版面配置管理(Layout manager)，是在 XML 檔中宣告，讓版面設計部份與程式實作部分分開了，改由寫在 XML 裡，程式

執行時再藉由讀取呈現出來。這樣的好處是分工更專業，版面設計者可以不用很了解程式就可以設計。Android 處理圖形的能力很強大，2D Graphics library 並未使用原本 Java 的 Graphics，而是自行定義自己的。底層 Library 則提供了 2D 與 3D 圖形庫(由 C/C++撰寫)。



第三章 分析 PBP Java AWT

3.1 Java AWT in Java ME PBP 軟體架構

Java AWT 在 Java ME PBP 軟體架構分為四層，Java AWT class(抽象視窗層)、Java Native Interface(JNI 層)、Microwinodow 層、Xlib 層，架構如圖 2。

圖 2 Java AWT 在 J2ME PBP 軟體架構圖

Java AWT

Java AWT(Abstract Window Toolkit)是由 Java language 所提供的一個 class library。它是獨立於平台的，並且提供一組 Graphics user interface 設計的工具。這些工具是由平台的 native GUI 所實現的。所以 AWT 一個重要的特性是 GUI 在每個平台上保留有自己的風格跟觀感。所有 GUI 的 class 都繼承來自 Component。Component 加入到 Container 中就建立一個 GUI。Container 也可以視為一個 Component 加入到其它的 Container 中，形成層層包覆的巢狀結構。AWT 的 Component 功能的實現以及真正將它們 display 到螢幕或是輸出裝置，其實是由 native code 實作的。

Java Native Interface (JNI)

Java Native interface 是 Java 平台中非常強大的功能。使用 JNI 的應用程式可以合併 Java 與不同程式語言所撰寫的程式碼。Native code 是在特定硬體環境執行的

程式，通常都是不同於 Java 的程式語言，例如 C 或 C++。JNI 將兩者合併使程式設計者可以享用 Java 平台的優勢但又不需要放棄原本已存在的程式碼，既方便又可以擴充 Java 平台的功能。因為 JNI 是屬於 Java 平台的一部分，所以對於程式設計者來說，可以一次解決不同程式語言互通性的問題，也可以期待 native code 可以與 Java 平台所有的功能或機制彼此互動合作。 [1]

Microwindows

PBP Release Implement 中所使用的原生圖形函式庫，是由昇陽公司依照 Microwindows 的架構修改，成為一個適合的原生圖形函式庫。其為一個由 C 撰寫基於 Xwindow 的圖形函式庫，沒有類別與繼承的觀念。雖然是由 C 撰寫的，然而其大量使用 Function point 來達到類別與繼承的目的。因此對 Java 來說，底層繪圖環境就只是一個類別 GraphicsEnvironment，GraphicsEnvironment 一開始被需要時，會建立好所有視窗環境，然後啟動 GraphicsEngine，和建立一個 C 的 Thread 負責將 XServer 的 Event 傳遞到 Java 層。所有關於畫線、畫圓、畫方形或是處理 Image 以及字型的需求，直接利用 GraphicsEngine 的資料結構，就可以很容易的完成。至於底層的事件則經由 Thread 利用 polling 的方式，定期呼叫 GetNextEvent method 來檢查是否從 Xserver 收到事件，收到的滑鼠以及鍵盤事件包裝成 MWEvent 傳到 Microwindow 層，再由 Microwindow 層包裝成 Java Event class，再利用 JNI 呼叫 Java 的 method 丟進 Java 的 Event Queue，如圖 3。在實作過程中發現到 Microwindows 並沒有 Thread safe 的機制，因此必須要額外的製作 Thread safe 的機制，以防止執行時的錯誤。

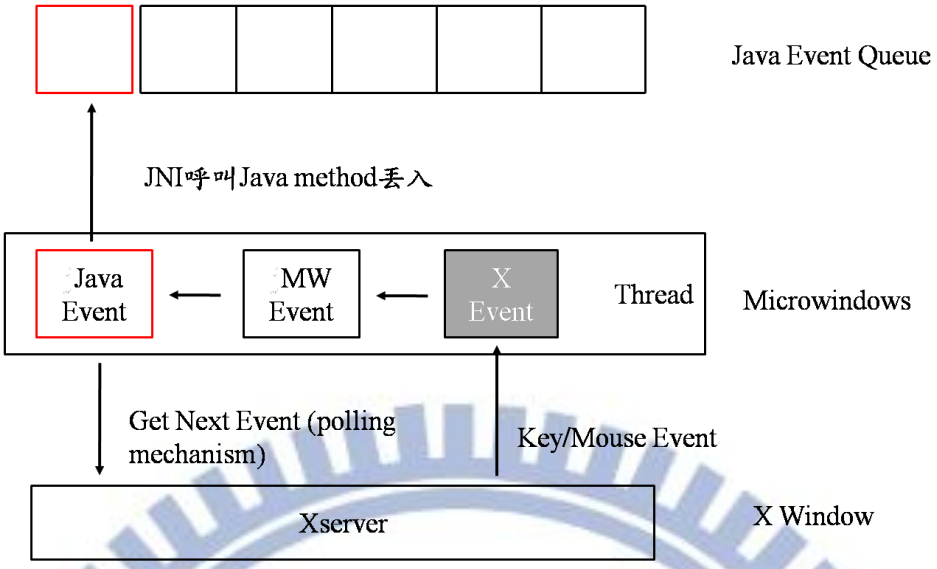


圖 3 Microwindws 建立 Thread 用 polling 機制處理 Event 的流程

Xlib

X Window 系統是 UNIX 世界中標準的圖形操作介面。X Window 系統的運作是 Server/Client 的模型。 Server 指的是 X Server, 它通常掌管一個完整的 Display。根據定義，一個傳統的 Display 包含一個顯示器、一個鍵盤、及一個滑鼠，或者還有其他選擇性的輸出入裝置，換句話說，它就是一個完整的圖形桌面裝置。而 Client 指的是在 Display 中執行的所有 X Window 應用程式，需要在螢幕上繪圖、需要接收滑鼠、鍵盤等輸入 .... 等，都必須向 X Server 發出請求，由 X Server 代為完成。X Server 與 Client 之間的溝通協定就稱為 X 協定。 X 協定不只可用於本機的 Display (亦即 X Server 與 Client 都在同一部機器上執行)，它還具備了網路的通透性，也就是 X Server 與 Client 可以在不同機器上執行，例如將遠端的 Client 視窗顯示到本地的 X Server 上，而本地的使用者在使用時就和操作本機執行的 Client 一樣，不會有分別。此網路的通透性正是 X Window 系統的最強大特點之一。

3.2 Java AWT in PBP

Java AWT 有主要的幾個部分，抽象視窗元件的實作、事件驅動機制、layout 管理、input device 的資訊接收。由於 Java ME PBP 比標準的 Java SE 的 Java 執行環境能力較低，在考慮能夠實現以上功能又具有跨平台性，將 Java AWT 做稍微的改變，採用的是 lightweight AWT 以及由 LightweightDispatcher 來分派來自 native 層的事件。

Lightweight AWT

不同的作業系統支援不同的繪圖工具，所以實作一個可移植的視窗工具組的工作任務變得困難。從作業系統中分離出視窗抽象(abstraction)，也就是使用者觀點的部分，會牽涉到工具組從一個平台移植到另一個平台時，需要大量程式設計的問題。若使用者相關的部分和實際工具組的實作部分是互相分離的的話，整體移植工具組的工作會簡單很多。這就是 Java AWT 實際上的做法，分離 AWT class 的抽象和他們在特定視窗平台的實作。AWT 不直接和作業系統中的視窗工具(native GUI)組溝通。它們會創造一個對等物件做為 AWT class 和原生(native)視窗工具組溝通時的管道。每一個 class 的實體物件都會關連到相對應的對等 class(peer class)。若要將 Java API 移植到另一個工具組上，只有 peer interface 必須重新實作。較高的抽象類別：如 Component、Button 都不用改變。這樣的 Java AWT 在移植時需要實作大量與平台相關的 peer interface，稱為 heavyweight，不具有跨平台性又耗費系統資源。無法跨平台的特性在嵌入式系統中就失去優勢。因此為了 Java AWT 更有跨台性與彈性，在 PBP 中，大部分的元件不需要與平台相關的元件(peer class) 實作出來。除了部分最底層的功能(ex. Window)才需要平台支援，稱作 lightweight。這樣的好處是與平台的相依性降低了，處理最少與平台相關的程式問題，增加移植的簡單度，又可以任意的改變元件外觀，不受限於每個平台。

Delegation event model(委託事件模型)

在 Java 中,將產生事件的設計與處理事件的動作獨立分開,稱為 Delegation event model。Java AWT 是一個事件驅動機制(Event-Driven Mechanism)的系統，在執行過程中,設計根據不同的情形下會產生事件,然後藉著事件會去驅動相關動作的產生，如果按照一般的程式設計邏輯，當發生事件時，需要去追蹤此事件屬於那一個 Component，之後還要尋找應該執行的 function，然後呼叫此 function 執行動作。所以只要產生一個事件,就必須要去追蹤的話就太耗費資源了,為了節省追蹤的工作，才研發出 Delegation event model。將產生事件的動作邏輯設計，與執行事件動作的邏輯設計分開，中間委託 EventQueue 和 EventDispatchThread 去負責完成這連結的動作，不僅產生的事件能觸發正確的動作還能節省資源，這是一個很好的設計，所以 Delegation event model 會保留在簡化的設計中。

Event-Driven Mechanism (事件驅動機制)

Java AWT 的事件驅動機制就是一個程式的執行流程決定於事件，這個機制有三大部分：一為事件(由 Event class 實作)，根據來源不同大致可分為三類，有鍵盤(key)事件、滑鼠(mouse)事件、或是繪圖元件(Component)事件；二為事件管理與分派，由 EventQueue 負責排程規劃，EeventDispatchThread 負責分派；三為事件的接收者(由 Listener class 實作)，Listener 是在繪圖元件上負責過濾 event 的機制，唯有註冊過的 event，才能被 Listener 接受並得到執行。事件驅動機制概念如圖 4：

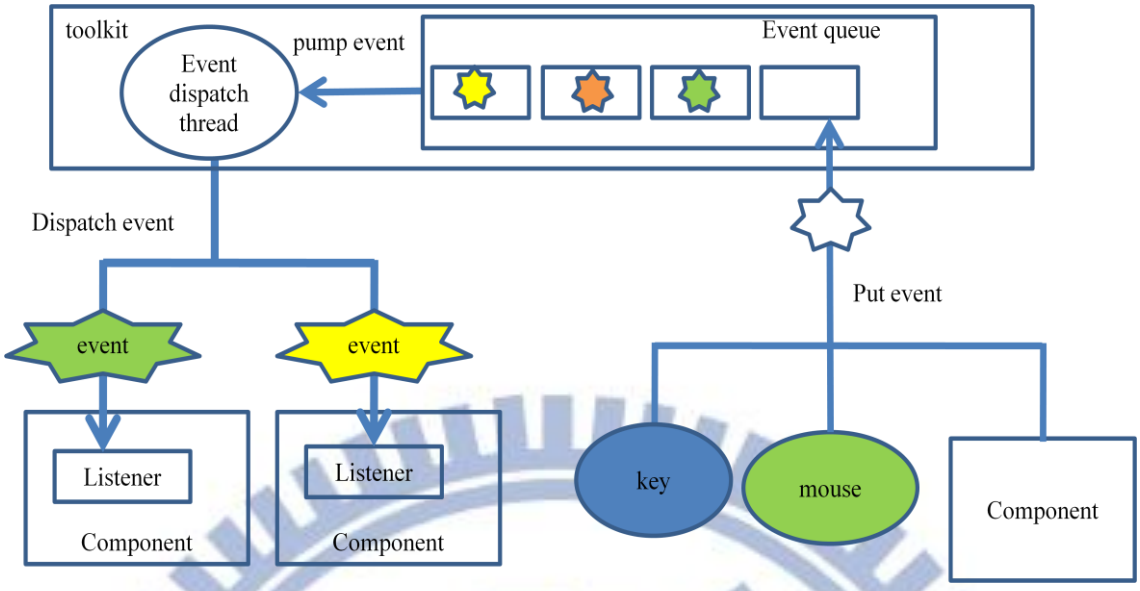


圖 4　Java 事件驅動機制概念

當使用者輸入文字時，這時就會丟出鍵盤(key)事件的時候；當使用者滑鼠點擊到繪圖元件時，這時就會丟出滑鼠(mouse)事件；當畫面需要重新更新時，Component 就會丟出 PaintEvent，或是視窗要縮小時 Component 會丟出 WindowEvent，不管是哪一種方式產生的事件，都會交由系統統一管理與分派。系統的工具列(由 Toolkit Class 實作)中有一個負責接收所有事件並進行排程的 EventQueue，接著在固定的一段時間內就會由 EeventDispatchThread 負責 pump 出事件，然後再根據事件的產生來源分派給相對應的 Component，而 Component 上的事件接收者 Listener 決定是否這事件有被註冊過，如果有就接受下來，接著 Component 就執行該事件的內容。這樣就完成了一個事件被觸發後直到完成動作的流程。至於判斷是否有註冊過的方法是 Java.awt.event 這個 package 裡已有定義好每個事件都有相對應的 Listener。如果 Component 中有宣告該事件相對應的 Listener，則表示此事件已被註冊。

分派來自 native 層的事件

在事件驅動機制中提到的EeventDispatchThread分派的事件是Java event class，不包括來自 native 層的事件。分派來自底 native 層的事件則會因為不同的 Java 平台有不同的分派方式。在 Java SE 中的 AWT，來自 native 層的事件，例如鍵盤事件、滑鼠事件，分析並把事件分派給屬於的 Component 這個工作，其實是交由 native GUI 負責。每一個 Component 在 native 層都有一個相對應的 peer 元件，負責記錄 Component 資訊，所以 native GUI 擁有 Component 的絕對位置座標，根據絕對位置座標判斷完事件是屬於哪一個 Component，就產生一個 native 事件給 peer 元件，透過 peer 元件在 native 層事件就直接分派給 Component 了。但在 Java ME PBP 中除了 root container 以外，其他 Component 都是 lightweight，在 native 層並沒有相對應的 peer 元件。native 層一點都不知道任何 Component 的資訊，必須將 native 層產生的事件必需包裝成 Java event object，然後傳遞給 Java 層，分派事件的工作就交由「輕量級分派者」負責(由 LightweightDispatcher Class 實作，為 Windows Class 持有)。從 native 層傳遞上來的 Java event object 都統一設定來源屬於 Window，然後丟到EventQueue中進行排程與分派。Window 接受被分派來的Java Event object，先交由 LightweightDispatcher 。 LightweightDispatcher 分析事件座標位置跟 Component 定義的矩形範圍是否有交集，如果有的話，即判斷屬於該 Component，則將 event 分派到該 Component。LightweightDispatcher 分派完如果事件沒被分派出去，表示為 Window 的事件，則由 Window 進行事件的處理。

3.3 class 種類

Java AWT class 在 PBP 中依照 class 特性分為十五類。Java AWT class 在 PBP 中的十五大類，每個大類的 class 列表在【附錄一】。根據系統觀點再分為四大功能，前三種是基本功能，能運作 Java AWT 基本功能。第四種其他功能依照需求或情況增加即可。四大功能與所屬的 class 種類，如表格 2。

事件的處理模式	Event管理與分配、Event、EventListener
抽象視窗元件	抽象視窗元件
與native層相關	平台工具組
其他功能	顏色、字型、特性、滑鼠、基本定義、權限管理、Error/Exception、Debug機制、避免錯誤、Layout管理、影像處理

表格 2　Java AWT 四大功能中包含的 class 種類

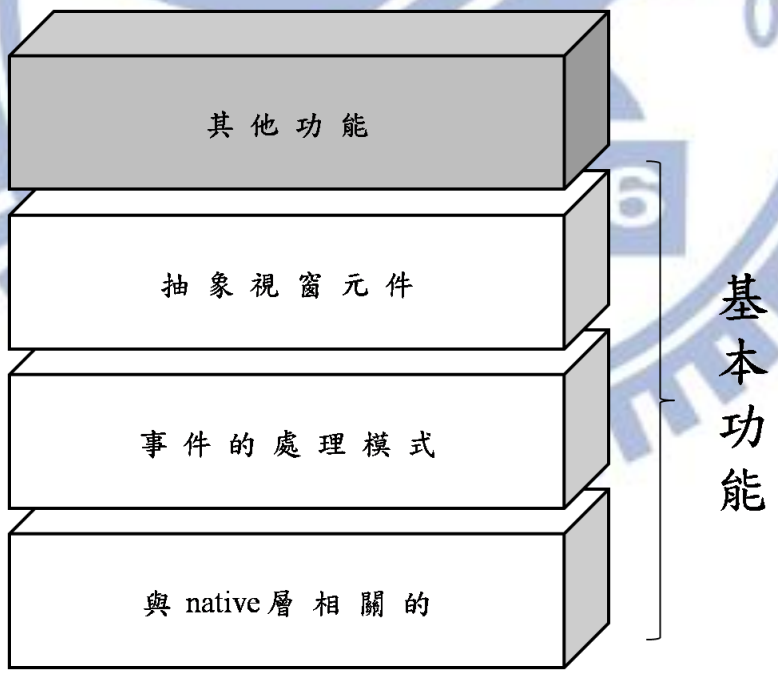


圖 5　Java AWT class 分層

這四大功能可以將 Java AWT 可以分為四層，如圖 5。底下三層是 Java AWT 的運作層、基本功能層。最底層的與 native 層相關，主要的目的是啟動並且溝通需要 native 層實作的動作。例如畫圖動作、起動畫圖環境、顯示視窗等等。第二層事件的處理模式，目的是在管理所有的訊息，Java AWT 將每個訊息視為一個事件，產生事件者會丟出訊息(產生一個事件)給執行事件者，產生事件者丟出時都會由事件的處理模式接受並且統一進行管理，等到需要執行此事件的時候，事件的處理模式才將事件傳給執行事件者。第三層是視窗抽象元件，是 Java AWT 中的基本單位，所有的情形都利用基本單位的特性來運作。在事件的處理模式中，元件就是產生事件者與執行事件者，在設計版面時，元件可以改變位置、大小、樣式來符合設計觀感等等。最上層其他功能，是以抽象視窗元件為基礎，根據特定情形才需要。例如 Debug 機制中只有在出現錯誤時才需要藉由這些 class 來獲得偵錯的資訊。其中有一個重要的功能，就是設計版面功能，屬於設計層次，有許多可以延伸變化的空間，獨立於 Java AWT 的運作。Android 之後就是將此設計層次與運作層次分開，然後充分開發彼此價值的好例子。

3.4 class 的設計模式

模式就是不斷重複出現的現象。設計模式就是找出 class 之間的互動不斷出現的現象，並且表示出這個模式背後的設計理念。了解設計模式可以幫助我們了解系統，提供了系統的脈絡，也可以藉此來描述系統的準則與策略。Java AWT 基本功能為移植和修改的基礎。分析 PBP 中的 Java AWT 基本功能 class，畫出 class 的 interaction diagram 如圖 6。

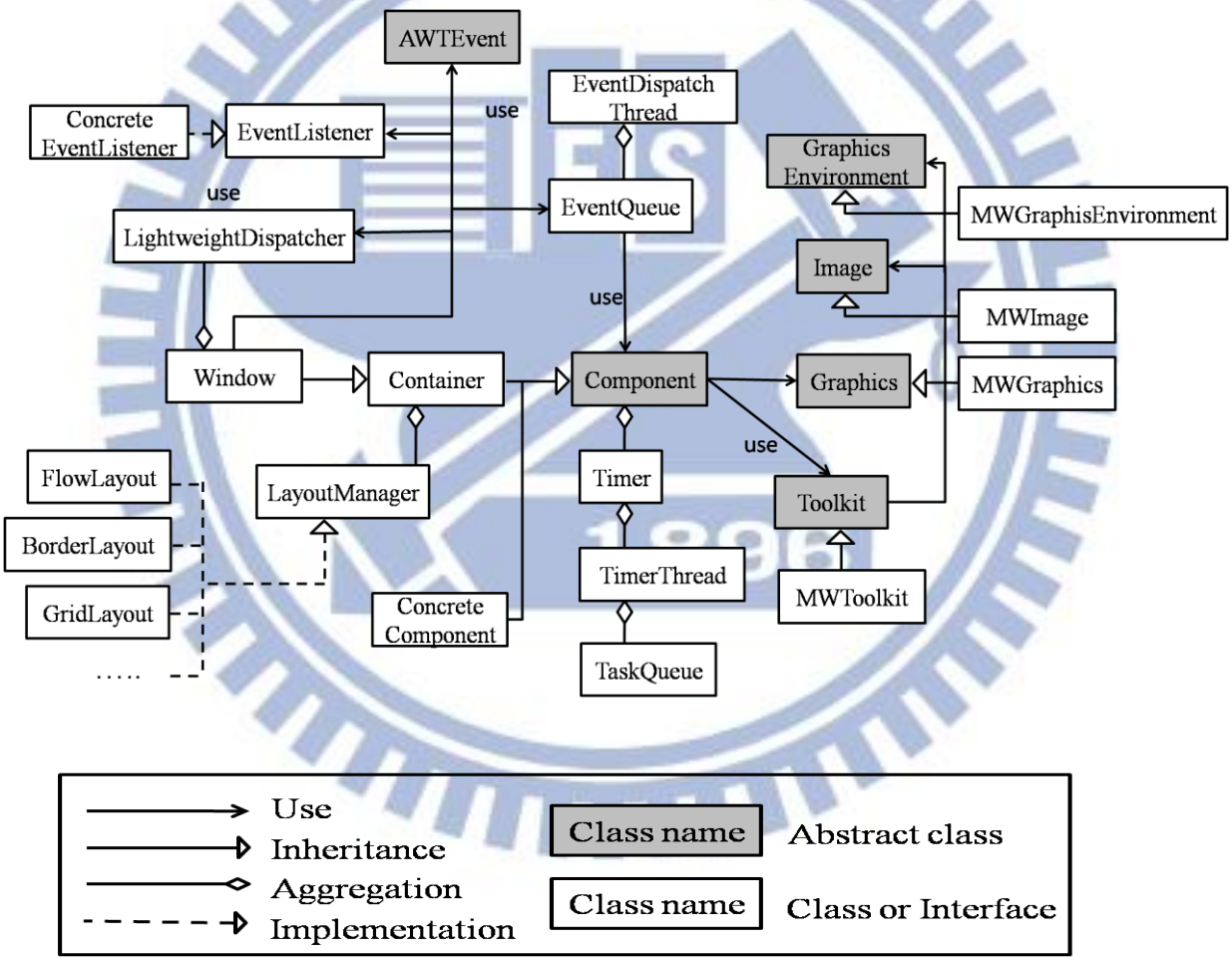


圖 6 Java AWT 基本功能 class 的 interaction diagram

根據[2] [3]裡的定義，分析出圖 6 Java AWT 基本功能 class 的 interaction diagram 有 Observer, Composite, Adapter, Strategy, Abstract Factory, Singleton 六種設計模式，接下來將會一一說明。

Observer,Composite,Adapter 模式

Observer 模式定義:在物件間去定義一個一對多的依存關係,使得當一個物件狀態改變時，所有與它相依的物件也都會被通知，並且自動更新。

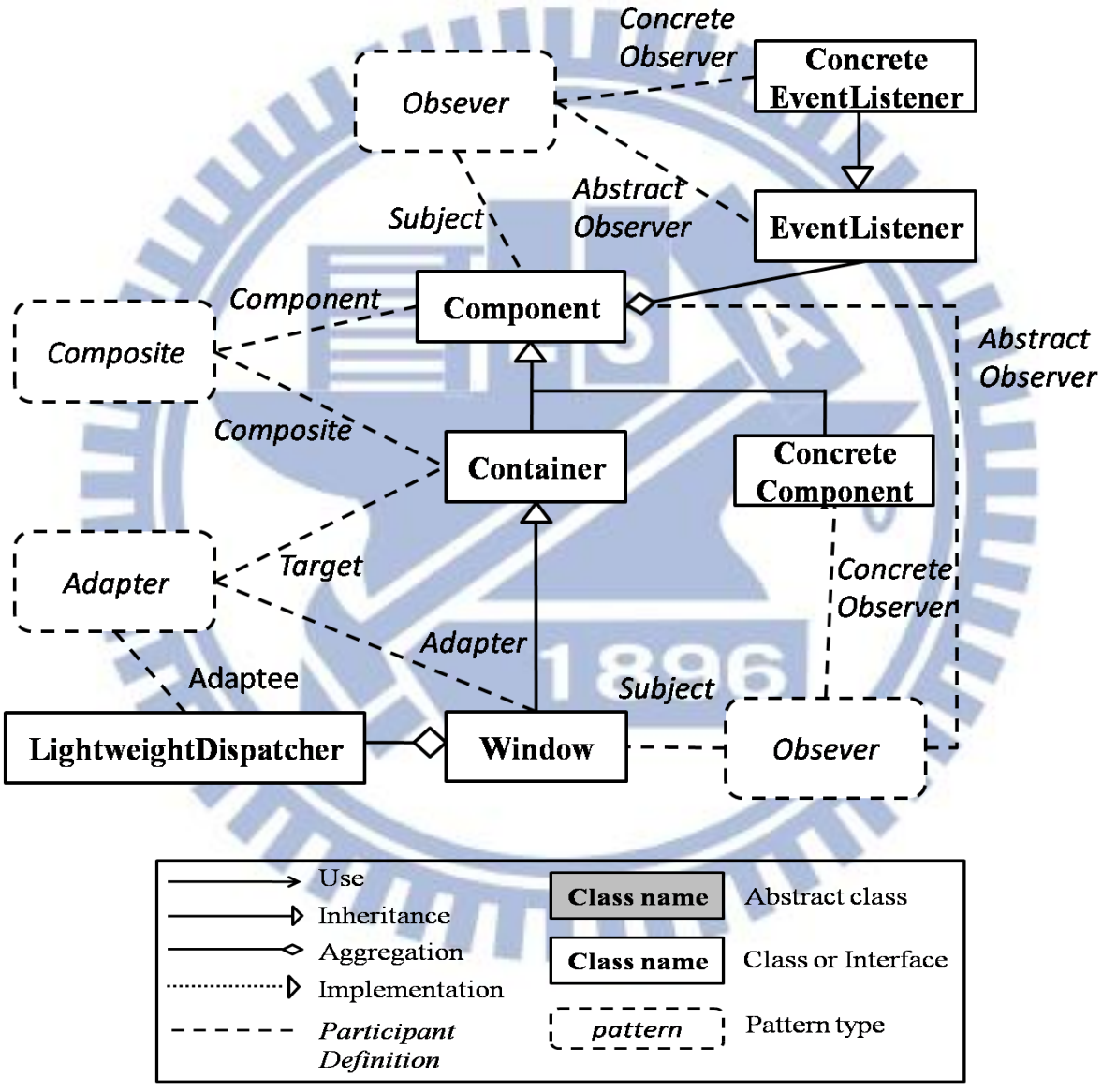


圖 7　Java AWT class 與 Observer,Composite,Adapter 模式的關係

Component 與 EventListener 存在著 Observer 模式的關係。與模式相關的 class 關係如圖 7。Component 為被觀察者(Subject)，EventListener 為觀察者(Oberver)。被觀察者發生狀態改變時，需要通知觀察者，觀察者再依照改變的狀態執行相對應的動作。EventListener 會先向註冊 Component，註冊需要觀察的事件(Event)，例如 MouseEvent、KeyEvent。當 Component 接到了 Event，就會根據 Event type 去通知 EventListener 並把 event 傳遞它，EventListener 接到屬於自己觀察的事件時，就會負責執行動作。

Window 與 Component 存在著 Oberver 模式的關係。與模式相關的 class 關係如圖 7。當 Window 接到底層傳上來的 event，就會委託 LightweightDispatcher 去分配 event，LightweightDispatcher 就會按照 Event 的座標，自行計算出 cursor 或 mouse 的鎖定的 target Component，並將 Event 傳遞給它。如果是 key event 或 focus event 直接傳給 programer 設定好的 focus Component。接著 Component 再根據 event type 傳遞給 eventlistener，eventlistener 再做相對應的動作。

Composite 模式定義:將物件組合成樹狀結構用來表示全部的階層關係。使用的目的是讓使用者用相同觀點看待單一物件或是組合物件。

Component 與 Container 存在著 Composite 的模式關係如圖 7。站在系統的角度，Container 在抽象概念上屬於 Component，系統將它視為一繪圖元件，有著繪圖元件的所有行為，所以繼承自 Component。實際上 container 是一集合概念，而 Component 是集合中的每一個 element。它所有的繪圖行為都是為了同時實現多個 Component 繪圖行為，除此之外 container 還有負責管理的工作，例如根據畫面佈置，決定這些 Component 的位置，或是給定一個座標，找出此座標屬於哪一個 Component 範圍內。系統會根據不同的情形會想將所有的 Component 進行分組，不同組的 Component 會被分派到執行不同的工作，或是相同的工作，但每一組裡 Component 會執行不一樣的步驟。這時需要統一管理同一組裡的 Component。指定它們同時進行相同或不

同動作，或是想要在 Component 彼此之間進行的互動時，就可以使用 container 來負責執行這些動作。

Adapter 模式定義: 轉換物件的介面成另一個使用者所期望的介面。Adapter 使那些因為不相容的介面而不能一起工作的物件得以連結。

Window 與 LightweightDispatcher 存在著 Adapter 的模式關係。與模式相關的 class 關係如圖 7。因為所有的 Component 都具備有 dispatch event 的功能，但正常情況下此功能都是將 event 分配給已註冊在 Component 上的 EventListener。Window 有別於其他的 Component，Window 除了分配 event 給 EventListener 之外，也負責將 event 分配給其它 lightweight Component，而這工作就委託 LightweightDispatcher 去執行。

Strategy 的模式

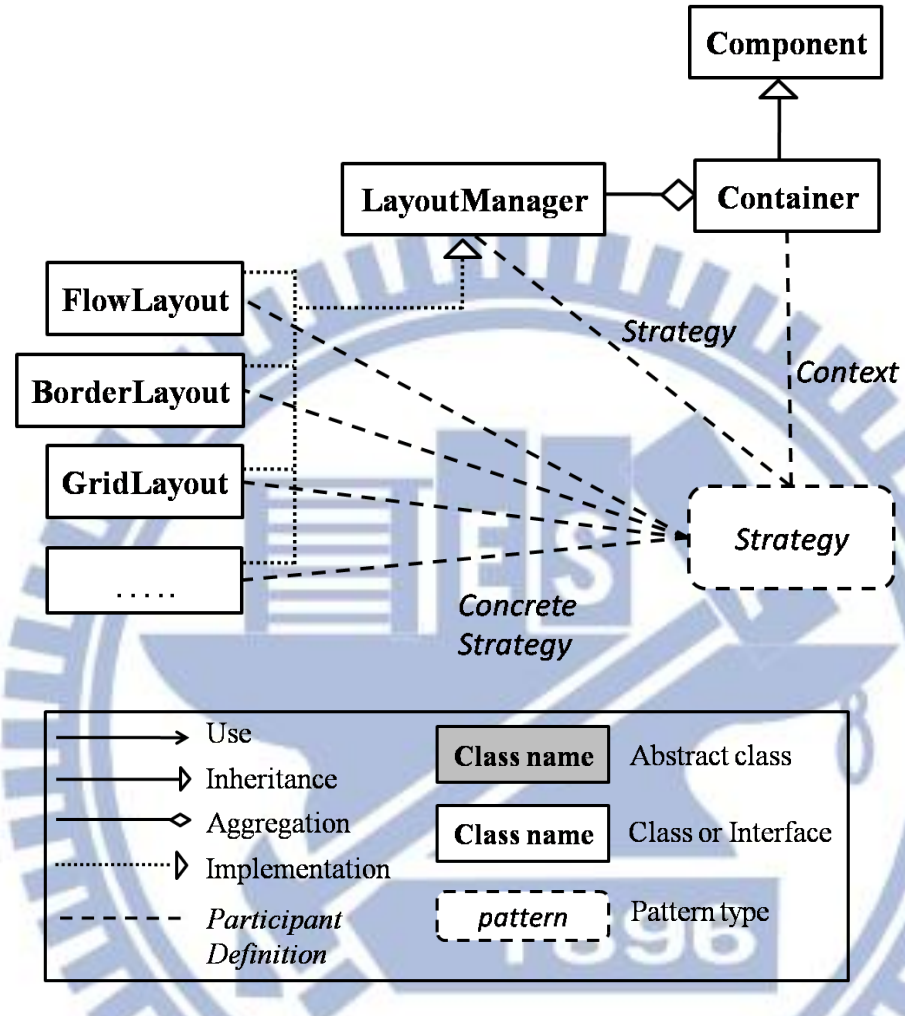


圖 8　Java AWT class 與 Strategy 模式的關係

Strategy 模式定義：定義一個演算法家族，把他們每一個都包裝起來，並使它們可以互相交換。Strategy 讓演算法可以與使用它的使用者獨立。

Container 與 LayoutManager 存在著 Strategy 的關係。與模式相關的 class 關係如圖 8。Container 其中重要的一個功能就是負責畫面的佈置，實作的內容就是排列 Component 的位置。根據想要觀看的畫面，有許多不同設計擺放的方式。一邊實作視窗元件 Component 的功能面的同時還要去顧慮畫面的佈置，為了減少程式設計的負擔，於是將畫面佈置特別獨立出來，與原本程式設計的邏輯區分出來。畫面佈置

是不依賴任何視窗元件功能，只要根據不同需求就可以進行實現，所以視為是一種策略。不同策略就只是演算法上的不同。Strategy 就表示相同的目標(context)要實現的功能可以由不同的演算法(strategy)來實現。Container 中負責畫面佈置的功能以 LayoutManager 來表示，那實作 LayoutManager 的 class 就是實作許多不同安排位置的演算法，例如像是 FlowLayout，水流式版面配置屬於一種簡易的版面配置方式，實際動作就是依序在同一列放置元件，沒有任何特殊編排，如果元件超過邊界，就置於下一列。

Abstract Factory, Singleton 的模式

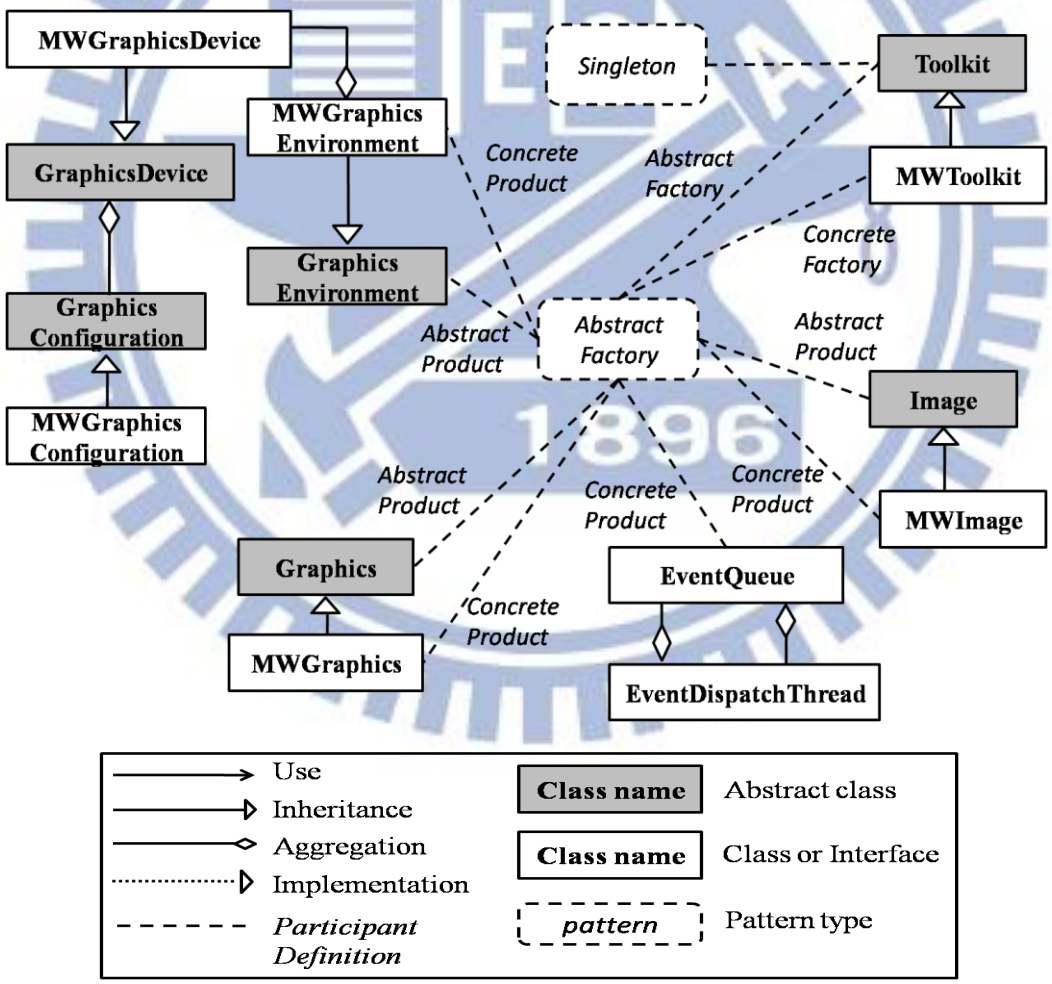


圖 9 Java AWT 與 AbstractFactory、Singleton 模式的關係

***Abstract Factory 模式定義*:** 提供一個創造相關或相依的物件家族而不需要指定他們具體的物件。有些物件在初始時會根據不同情形做設定，在這樣的情況下，Abstract Factory 確保在不需要考慮如何產生的細節之下，能產生正確的物件。

在不同平台下，需要不同組的 Image、GraphicsEnvironment、Graphics、EventQueue，稱為 Product (abstract class 為 abstract product，一般 class 為 concrete product)，初始這些物件的控制是相關的。負責產生這些物件的稱為 Factory(abstract class 為 abstract factory，一般 class 為 concrete factory)，Toolkit 就是 Factory。它們跟 Toolkit 就是 Abstract Factory 模式的關係。與模式相關的 class 關係如圖 9。在這樣關係中，Toolkit 就是提供了整個 Java AWT 系統或與平台相關的功能。

Factory 與各個 Product 的在 Java 中的重要性以下會說明:

Toolkit 是實作系統與平台相關的功能，用來產生 peer Component，但在 Java ME PBP 中大都是 lightweight Component，所以沒有此功能。除此之外，還提供三種服務：桌面量度(desktop metrics)、獲得字型資訊、影像下載與準備。Toolkit 透過產生 GraphicsEnvironment 物件來取得底層環境相關細節，提供桌面量度、字型資訊的服務。還有系統中共同會使用到的 Graphics 和 EventQueue 物件。以及執行影像下載與準備的 Iamge 物件。

EventQueue 用來管理 event。Component 可以不用管理自己的 event，可將所有管理 event 責任統一交由 EventQueue 負責。透過 EventQueue 的 method 可以來操作 event。Component 產生完 event 時，就會呼叫 EventQueue 中 postEvent()，將新產生的 event 丟到此 EventQueue 中，EventQueue 會根據 event 被放入的時間與優先權決定 event 的順序。EventDispatcherThread 則固定一段時間就會呼叫 EventQueue 中 getNextEvent()將 event 取出，並把 event 分配。

Graphics 用來執行各種幾何圖形畫圖動作的 class。當執行時需要顯示 Component 時，就會執行 Component 中 paint()裡的演算法。此部分的演算法是 programmer 是根據想要的外觀去呼叫 Graphics 裡的各種幾何畫圖動作的 method 所組成的，例如畫線、畫圓等。在 lightweight Component 中，外觀是可以自行設計，有別於 Java SE

的 heavyweight Component，其外觀與樣式由不同平台預先設定好的，所以使用 Component 時沒有設計的權利，比較方便但減少了設計的空間。Graphics 扮演著很重要的角色，而 Graphics 裡所有畫圖動作的快慢在程式的執行速度上佔有決定性的影響力。

Image 是個記憶體上的一個物件且可以被顯示。藉由呼叫 Component 或 Toolkit 的 getImage()可以來載入 Image，然後藉由 Graphics 的 drawImage 顯示到螢幕上。Image 是個 abstract class，由平台設定的 class 來實作，例如 MWImage。

GraphicsEnvironment 代表處理圖像的環境，也就是特定機器處理圖像的整體能力，包括字形、螢幕裝置....等等。目前透過這個 class 的可以取得環境的字型資訊，還有確定裝置的類型。一開始在初始 window 物件就會啟動此 class 的物件。在 Microwindow 的實作上，主要的功能除了建立裝置(GraphicsDevice)物件外，也會在建立一個 Java thread。負責在底層環境中執行等待與接受 event 的動作，然後將 event 包裝成 Java event 傳遞上來。

***Singleton 模式定義*:** 會使用 singleton 的目的是，確保此 class 只有一個 instance，並且以一個 global variable 的觀點來使用它。

Toolkit 有著 singleton 模式的關係，與模式相關的 class 關係如圖 9。在 AWT 中，Toolkit 提供許多工具給所有的物件使用，所以會不斷被用到，但不想要一直重覆 initialize 其物件，造成 garbage collection 清除工作的負擔。所以產生一個物件後，所有的 class 或 object 使用時輪流去得唯一物件。。

3.5 Native 層技術的支援

在 PBP 中需要底層支援的功能有底層環境資訊的存取、影像下載、呈現與處理、色彩模型的轉換與顏色的實作、字型的實作、滑鼠的顯示圖形。

Native 層環境資訊的存取

在 Java AWT 中最終的目的就是在輸出裝置上呈現圖形、文字和影像，而輸出裝置通常是電腦螢幕和印表機。即使輸出裝置不同，程式設計部分也希望不會因此而變動。但是在輸出時，如果取得裝置的相關資訊，例如螢幕的解析度等，這樣裝置就可以直接讀取而不用再做影像資料上的轉換。所以不希望主要程式受影響又需要取得這方面資訊情況下。這時 Java 就提供了以下幾個 class 將這些會跟著底層環境變動的地方包裝起來，提供統一的介面來取得這些裝置資訊。真正的實作與提供的資訊部分交由底層負責。[4]

GraphicsEnvironment

圖形環境物件可以代表處理圖像的環境，也就是特定機器處理圖像的整體能力，包括字形、螢幕裝置....等等。[4]目前透過這個 class 的可以取得環境的字型資訊，還有確定裝置的類型。一開始在初始 Window 物件就會啟動此 class 的物件。在 Microwindow 的實作上，主要的功能除了建立裝置(GraphicsDevice)物件外，也會在建立一個 thread。負責在底層環境中執行等待與接受 event 的動作，然後將 event 包裝成 Java event 傳遞上來。

GraphicsDevice

圖形裝置指的是電腦螢幕或印表機。[4]目前 Java 設定的類別有三種裝置，螢幕、印表機、影像。每一個 GraphicsDeviced 都擁有一到多個狀態。所以在輸出影像時取得目前的狀態(GraphicsConfiguration)最重要。當 GraphicsDevice 的物件被建立時，initialize 的動作主要也有兩件，一件是向底層註冊一個記憶體區塊為此裝置輸出專用。在 MicroWindow 中，是呼叫底層 openScreen()，然後回傳一個 id，做為裝置的識別。另一件事是建立 GraphicsConfiguration。

GraphicsConfiguration

裝置可以設定許多不同的狀態。[4]有許多顯示裝置都可以支援同時不同的解析度或色彩解析度。GraphicsConfiguration 主要都是用來回傳設定本身的訊息，例如

色彩模型。或是 graphics 活動空間與裝置空間之間的轉換，例如像圖像的 x，y 的座標方向與裝置空間不一樣，就需要轉換功能。還有建立與裝置設定相容的影像功能，也就是輸出的影像，色彩模型與資料格式都與裝置設定相同，這樣方便影像可以快速地從裝置讀取輸出。

在 GraphicsDevice 物件建立好，等於是向底層註冊了一個 device，並回傳 id。這時 GraphicsConfiguration 裡的真正資訊也已確定了，而 GraphicsConfiguration 物件建立時只是透過 device id 去取得這些資訊並儲存起來。最後 GraphicsConfiguration 物件會被傳回 window 物件，當作是取得底層圖像資訊的介面。當需要用到這方面相關資訊時便直接從 GraphicsConfiguration 中取得，而不需要再傳遞訊息往下透過 device id 去取得。

影像下載、呈現與處理[5]

在 Java 中儲存影像的基本單位是 Image class。Image 的物件其實只是一個指向影像資料在記憶體上的位置。產生 Image 物件的有三種方法，有兩種藉由呼叫 Component 或 Toolkit 的 getImage()可以來載入影像，載入影像有兩種來源，一種是用 URL 指向影像檔案的網路位置，另一種是用檔案路徑和名稱從自身檔案系統中在入檔案。第三種是有 byte array 的影像資料用 Toolkit 中的 createImage()來建立全新的影像。這三種方法最後都會回傳 Image 物件，用來指向影像位置。直到真的需要 Image 資訊時才會真正去下載影像資料。

影像的呈現是由 Graphics 的 drawImage()實作並支援影相觀察者的概念。影像觀察者由 ImageObserver 來實作。因為下載影像時間都很長，所以需要 ImageObserver 來監控整個影像下載的過程。呼叫 drawImage()時會畫出已取得的影像資料，若尚未取得影像，會先返回等待影像。影像已經抵達時，ImageObserver 會呼叫 repaint()來重畫，repaint()會在呼叫 drawIamge()，因為以完整下載完影像，因此馬上畫出。

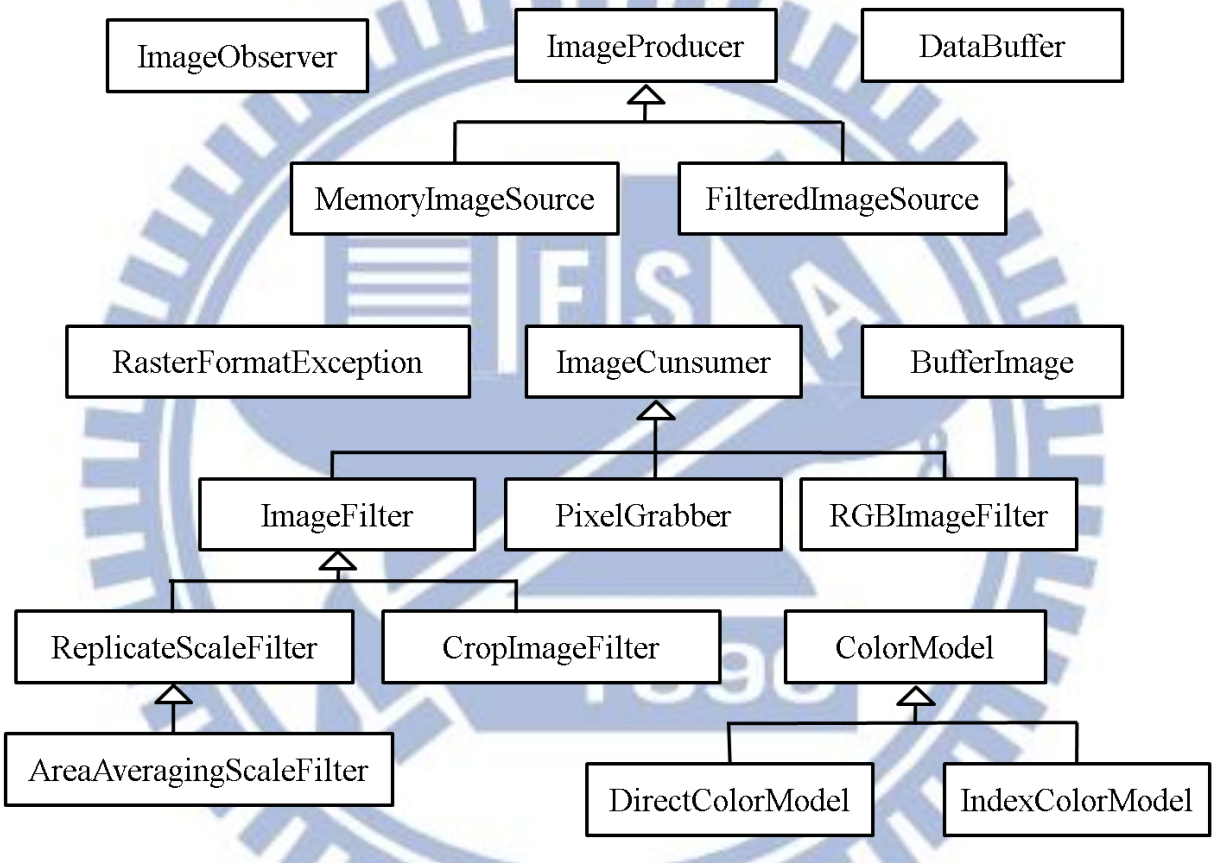


圖 10　Java.awt.image 這個 package 的 class 與繼承關係如

Java.awt.image 是 Java 影像處理的核心。Java.awt.image 這個 package 的 class 與繼承關係如圖 10。這個 package 主要的功能就是處理影像資料。影像處理模型是建立在 ImageProducer、ImageConsumer 介面。ImageProducer 是提供資料來創造影像物件，負責將 pixel data 傳給 ImageConsumer。Image 物件若需要做為影像處理的來源就可以實作 ImageProduser，或直接宣告 BufferedImage。MemoryImageSource 是一個實作 ImageProducer 物件，它從記憶體一 pixel value 的陣列中取出影像資料，傳遞出去。

總之，要取得影像資料就必須透過 ImageProducer。ImageConsumer 則是接收影像資料並且處理 pixel value。一個 ImageProducer 可以傳遞給多個 ImageConsumer，相同的影像資料可以做不同影像處理。模型如圖 11。

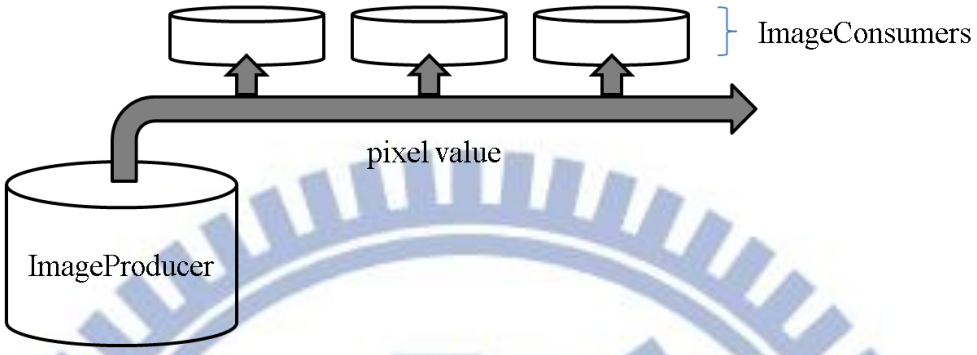


圖 11　ImageProducer 與 ImageConsumer 模型概念圖

影像過濾器是由 ImageFilter class 來實作，它是經過適當處理影像資料了再傳遞給一個 ImageConsumer。過濾器的模型如圖 12。過濾一張影像需要由 FilteredImageSource 物件來來完成。過濾器中最簡單就是修改像素的 RGBα 值，做

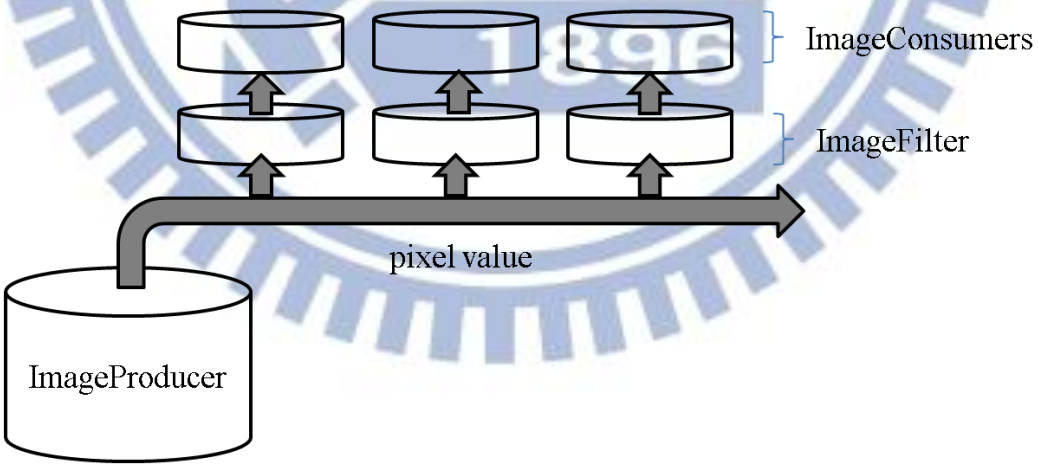


圖 12　ImageFilter 模型的概念圖

色彩的調整或是轉換。AWT 提供了這類的過濾器可以實作 RGBImageFilter。也有更複雜的影像處理，不對影像中每個像素作一致的過濾，例如模糊影像或是增強影像顏色的對比。這像的過濾器就需要繼承 ImageFilter。

Java 處理影像都需要 ImageProducer、ImageConsumer 、ImageFilter 合作才行，如果需要大量使用某個影像處理，這樣層層的宣告顯的太復雜了，於是希望有個更簡單的影像處模型，BufferImage class 的產生就是來解決這個問題。在 PBP 中，產生 BufferImage 物件時，會傳入 BufferImagePeer 物件當參數，所有的影像的取得透過 BufferImagePeer 的 funtion 交由 native 層來實作。

色彩模型的轉換與顏色的實作

ColorModel 這個 class 是在用來決定影像物件顏色如何轉換成預設的 RGBα模式，有紅色值、藍色值、綠色值跟 Alpha 值四種資訊。目前預設的 RGBα模式是一個整數值由 32 位元表示。0~7bit 為藍色值，8~15bit 為綠色值，16~23bit 為紅色值，24~31bit 位元為 alpha。如圖 13：

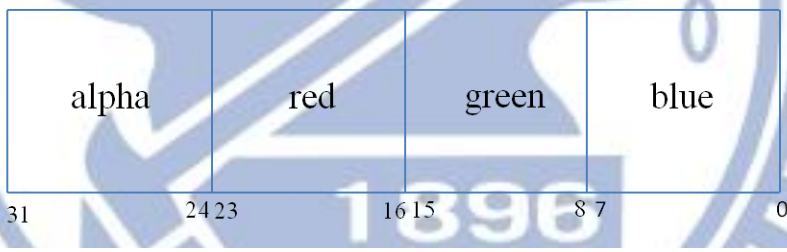


圖 13 RGBα模式

ColorModel 是個抽象 class，有 IndexColorModel 和 DirectColorModel 這兩個色彩模型分別繼承 ColorModel 實作色彩轉換的 function。除此之外，也可以自行定義其他色彩模型與 RGBα之間的轉換。Microwindows 目前在實作特定顏色時，會透過 ColorModel 的 getRGB()或 Color 中 getRGB()取得顏色的 binary value(32 bit)或是直接給此顏色的 binary value，傳遞到 MWGraphics 設定顏色的 native method，將 binary value 轉換成 RGBα模式，轉換到 Xlib 中的色彩映射矩陣(Color Map)，轉換成真正實作時的顏色值。。

字型的實作

Font的物件是用來表示一個在系統中特定的字型。在系統中有著一系列的字型，而這些字型都由平台所支援，可以藉著呼叫 Toolkit.getDefaultToolkit().getFontList() 取得平台所提供可用的字型組合，例如 TimesRoman,Helvetica,Dialog…等等。FontMetries 是提供一個工具，用來計算字顯示在螢幕上真正的長度跟寬度。可以藉由它所提供的資訊，來評估字在螢幕上的位置以及整個字型看起來真正效果。因為它會記錄每個字型的 ascent,descent,height,baseline。[6] 在 PBP 中是由 Microwindow 層提供一個 FreeTypeRender(由 C 撰寫的)，紀錄記憶體中載入的所有字型，可以提供 FontMetries 的資訊。實作字型的方式是根據字型檔提供的 bitmap 在相對應位置的 pixel 值填入顏色。

滑鼠的顯示圖形

Cursor 是用來表示滑鼠的物件。這裡面定義了許多滑鼠顯示的類型，例如移動中的樣子、縮小的樣子、指的文字時的樣子…等等。提供在不同情況下需要不同的圖形顯示的選擇。每換一次 Cursor 的類型將透過 Frame 將訊息往 parent 傳遞，最後由 Window 中的 native method　ChangeCursor()來實作。

第四章 設計 MMES AWT 與實作過程

根據之前的分析，我們希望能設計出更簡單的 Java AWT。因此我們希望提出一個 Java AWT 構想，根據 Java ME CDC/PBP 中的 Java AWT 的架構，將其架構精簡化，然後 native 層由 KNI 實作 native graphics library 的嵌入式平台版本，命名來自本實驗室，MultiMedia Embedded System，稱為 MMES AWT。在應用程式的界面方面，我們儘量保持原有的 Java ME CDC/PBP AWT API 設計，以支援大部份採用 lightweight AWT 的 Java 應用程式（例如符合 MHP 數位電視規格的 Xlets 應用程式）。而跟原本的 AWT 相較，MMES AWT 有以下的優點：

- 簡化了系統 classes 的需求。原本的 AWT 是架構在 CDC 的 Foundation classes 之上，較為複雜，MMES AWT 只依賴 CLDC 的 system classes，因此整個系統的程式庫大幅縮小了。
- 針對實際執行繪圖工作的 native functions，原本的 AWT 是透過 Microwindows 和 Xlib 來進行繪圖動作，雖然這兩個程式庫在 Linux-based 的嵌入式平台都有支援，但是針對不支援 Linux 的 deeply embedded 的平台，移植就會花比較大的工夫。而 MMES AWT 則是採用了我們自行開發的最簡化的 native 程式庫，簡化 native method 的存取，減少與底層的相依性，可以比較容易針對特定的嵌入式平台進行移植以及最佳化的工作。
- MMES AWT 設計適合由 KNI 實作 native graphics library，比 JNI(Java Native Interface)實作的版本減少耗費系統資源。

本章會先在 4.1 介紹 MMES AWT 的設計概念，然後在 4.2 節介紹設計及實作的細節。

4.1 設計的 MMES AWT 概念

MultiMedia Embedded System AWT ，簡稱 MMES AWT，的設計理念就是設計最簡單的、最基本的抽象視窗工具。實作在資源有限的嵌入式平台以實現基礎功能為主要，其他非必要功能則暫不考慮達到減少資源的耗費。所以實作的 class 也以三大功能為主：實作事件的處理模式、抽象視窗元件、與 native 層相關。MMES AWT 的事件處理模式修改成適合 KNI 實作的版本並且增加 EventPostThread，可以處理基本的輸入、輸出事件(PaintEvent、MouseEvent、KeyEvnet)。抽象的視窗元件簡化所有 class 的部分內容，只保留實 Frame 相關的 class，沒有 LayoutManager 系列的 class。native 層相關進行 class 內容簡化，沒有實作 GraphicsEnvironment、GraphicsDevice、GraphicsConfiguration、Image，並且增加設計的 MMESGraphics、MMESToolkit 兩個 class 來實作 Graphics、Toolkit 的功能。MMES AWT class 的 interaction diagram 如圖 14。

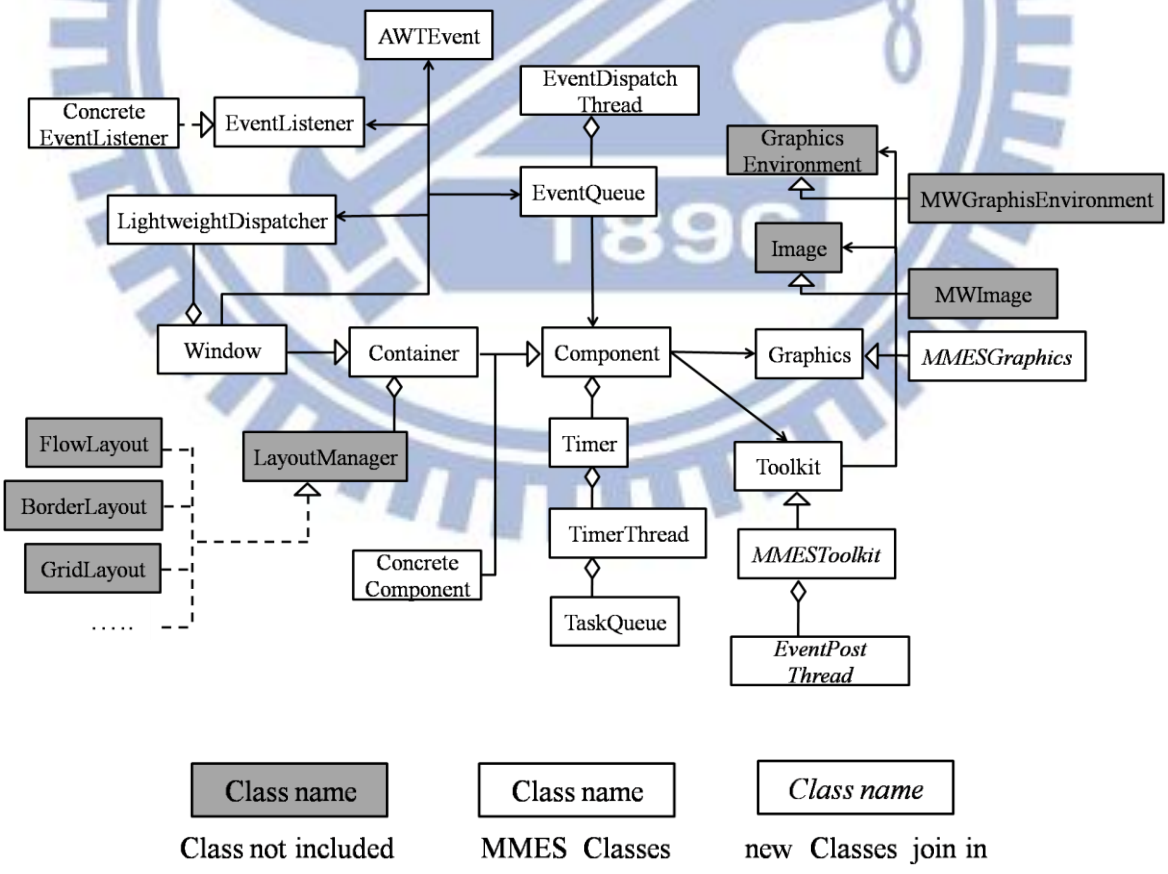


圖 14 MMES AWT class diagram compared with PBP AWT

MMES AWT 中目前不支援的 class 是根據先前的分析屬於進階功能的，例如影像的下載、呈現、處理，若需要擴充再增加相關的 class 與技術即可。

MMES AWT 目前支援事件處理模式，可以接收事件，並且定期分配事件給元件，目前已實作的事件有 PaintEvent、KeyEvent、MouseEventm。PaintEvent 是負責傳遞畫圖訊息的，系統中能完成輸出的部分，KeyEvent 是負責傳遞鍵盤的訊息、MouseEvent 是負責傳遞滑鼠的訊息，這兩種 event 在系統中完成輸入的部分。

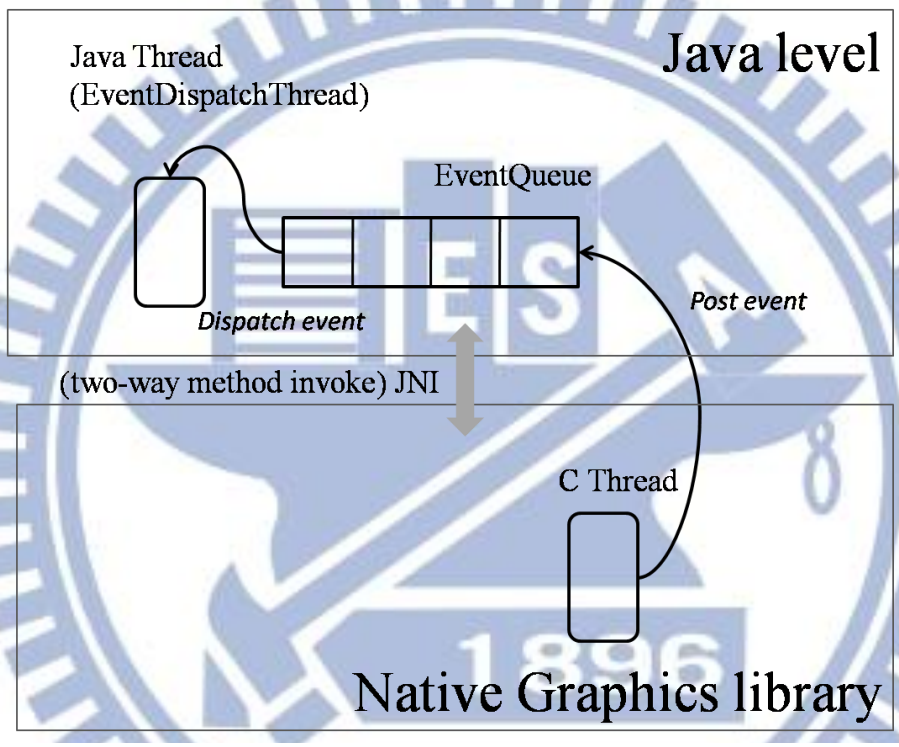


圖 15 Java ME CDC/PBP 中 Microwindows 處理 native 層事件方式

Microwindows 處理事件的方式(如圖 15)是建立一個 thread，接到事件包裝成 Java event object(細節如圖 3 Microwindws 建立 Thread 用 polling 機制處理 Event 的流程)之後，直接呼叫 Window 的 Post Event method，負責將 event 放到 EventQueue。JNI 支援 Java 層和 native 層相互呼叫 method 的功能，KNI 則沒有，只支援 Java 層呼叫 native 層 method 的機制。但是呼叫 native method 之後允許回傳值給 Java 層。所以調整的架構(圖 16)在 Java 層建立一個 Thread 名為 EventPostThread，負責用 polling 方式去檢查是否 native 層接到 event，如果接到之後，會將 event 的資訊進行

編碼，event 編碼的細節(如表格 3)，放入一個 32bit 的 event code value(如圖 17)，再將這value回傳給Java層。在Java層接到value之後進行解碼，產生Java event object，將解碼後的資訊傳給 event，再呼叫 Window 的 Post Event method。在這邊設計需要注意的是 Producer/Consumer 的問題，避免有 race condition 的情形發生。

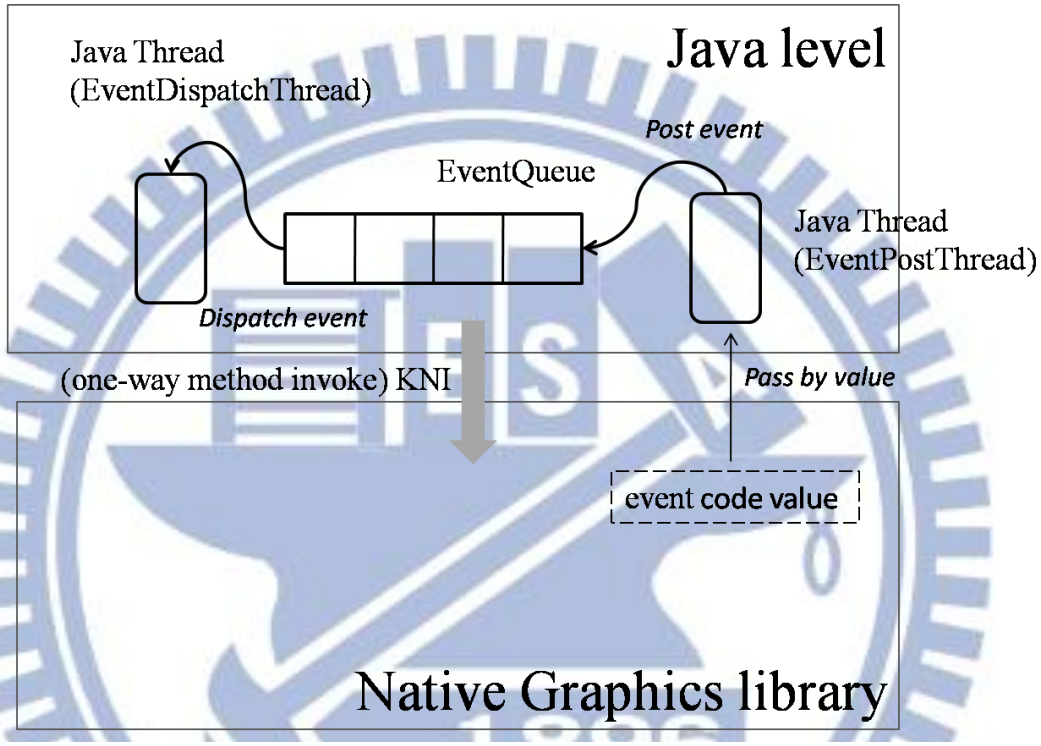


圖 16 MMES AWT 的 native graphics library 處理 native 事件方式

圖 17 MMES AWT 的 event code value format

表格 3 MMES AWT 的 native event 種類與 event code field 的細節

<table>
<tr><th>Event</th><th>type</th><th>mask</th><th>info1</th><th>info2</th></tr>
<tr><td>KeyTyped</td><td>0</td><td rowspan="2">Shift: 1</td><td rowspan="3">Key code</td><td rowspan="3">Key char (ascii code)</td></tr>
<tr><td>KeyPressed</td><td>1</td></tr>
<tr><td>KeyReleased</td><td>2</td><td rowspan="2">Control: 2</td></tr>
<tr><td>ButtonPressed</td><td>4</td><td>Left: 16</td><td rowspan="4">X座標</td><td rowspan="4">Y座標</td></tr>
<tr><td>ButtonReleased</td><td>8</td><td rowspan="2">Middle: 8</td></tr>
<tr><td>MouseDragged</td><td>12</td></tr>
<tr><td>MouseMoved</td><td>16</td><td>Right: 4</td></tr>
</table>

抽象視窗元件主要支援 Component 和 Frame 的功能。因為大部分的元件繼承來自這兩個元件，這兩個元件都支援了，其他的元件也確定可以正常運作。目前衍伸的元件只有實作一個 Frame，其他的元件之後可以依照情況增加。Native 層相關的 class，因為 KNI 的機制，由 native 層傳上來的資訊不能太複雜，integer 或是 true/false 值，所以省略了代表了 native 層環境的資訊的 GraphicsEnvironment、GraphicsDevice、GraphicsConfiguration，只留下可以啟動 native 層環境的 Toolkit、還有可以畫圖的 Graphics，並且新增 MMESToolkit 來繼承 Toolkit、MMESGraphics 繼承 Graphics，來實作兩個 class 中的 nativefunction 並且與 native 層互動。其他功能的 class 因為不是必要功能且與 native 層沒有直接關係，不影響移植，可以依照情形增加即可。目前 MMES AWT 時會用到的一些其他功能 class。MMES AWT 在 Java AWT 底層技術的支援中只支援顏色的實作和字型的實作，顏色的實作是將顏色的 integer 值傳到 native 層後，相映到 Xlib 中的 color map 再由 Xlib 去實作。字形的實作是直接由 Xlib 層載入字型庫實作。MMES AWT 在 Java 其他 class 的支援部分，為了簡化，設計個比較單純、陽春的版本，都暫時不支援。不支援也不會影響 MMES AWT 的移植。

之後可以依情況而在增加功能。MMES AWT class 的列表如【附錄二】【附錄三】，支援的情形 MMES AWT class 與技術支援情形如表格 4。

表格 4MMES AWT class 與技術支援情形

class功能分類	支援與否	支援的情形
事件的處理模式	有	支援PaintEvent、KeyEvent、MouseEvent的管理與流程
抽象視窗元件	有	簡化，支援Component、Frame相關class為主
與native層相關	有	簡化，支援Toolkit、Graphics
其他功能	部分	支援以上三種功能簡化後的class還需要的其他class
底層技術的支援	部分	顏色的實作
Java其他class的支援	否	

4.2 設計及實作細節

將 MMES AWT 真正的實作在 CLDC 的平台上實作過程可以分成三個步驟：第一是 Java 層的支援，原本 CLDC 的平台並沒有支援 Java AWT，增加原本平台中缺乏但是 MMES AWT 必需的 Java Class；第二是使用 Native Interface，要完整實作 MMES AWT 也需要平台上的支援，利用 native interface 建立起 Java 層與 native 層彼此溝通的橋梁；第三是 native 層的支援，真正畫圖的動作與圖學的演算法是需要 native 層的支援，才能讓畫圖動作真的可以在平台實現。所以必需在 native 層實作一個可以在作業系統中執行的 native graphics library，透過 Java 層傳遞來的參數，讓特定 native graphics function 得以啟動後，在作業系統中真正的完成完整的畫圖動作並呈現在螢幕上。

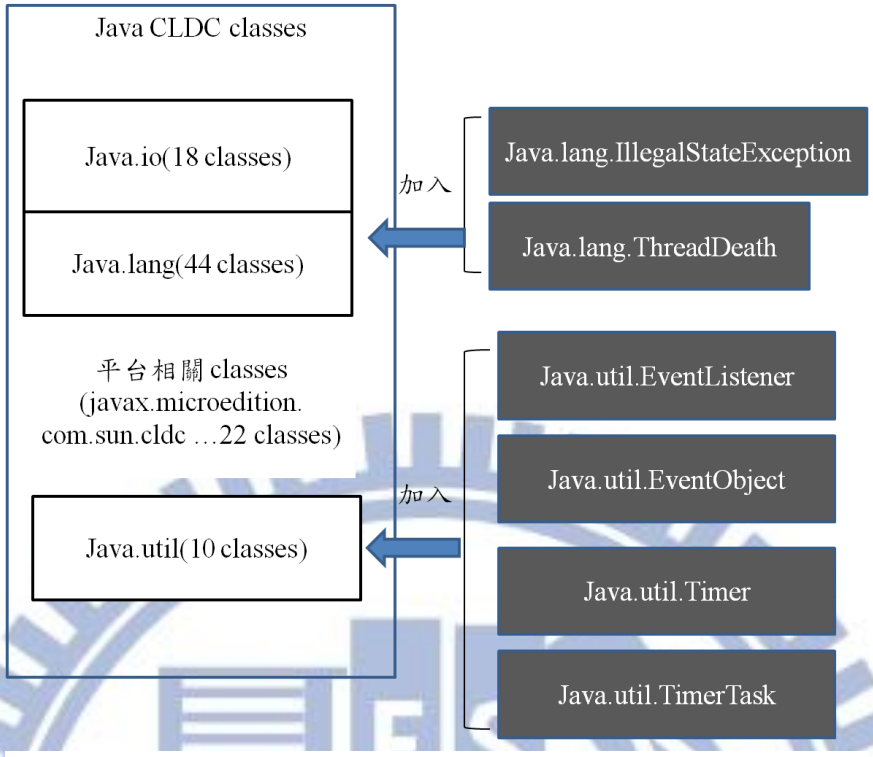


圖 18 MMES AWT 加入 CLDC 平台需增加的 class

4.2.1 增加 Java 其他套件的 class

Sun 的 CLDC 平台主要有 Java.lang、Java.io、Java.util、Javax.microedtion.io 四個 packages，Java.lang 有 44 個 classes，Java.io 有 10 個 classes，Java.io 有 18 個 classes，與平台相關的有 22 個 classes，但是 MMES AWT 會用到 system classes 還不夠，還需要增加 classes，有 Java.lang.IllegalSateException、Java.lang.ThreadDeath、Java.util.EventObject、Java.util.Timer、Java.util.TimerTask、Java.util.EventListener，如圖 18。除此之外，原本 Sun 的 Java ME/CLDC 的標準實作參考（reference implementation）並沒有 log 以及 exp 兩 functions 的支援，因此我們也針對 Java.lang.Math 做了部分修改，增加了計算 log 跟指數 exp 的兩個運算。

MMES AWT 所需要的 Java 套件，Java.lang、Java.lang.ref、Java.util、Java.io，做了部分修改與增加之後個套件 class 列表如表格 5、表格 6、表格 7，紅色為新增的

class，藍色為修改的 class。

表格 5 MMES AWT 所需的 java.lang/ref 套件 class 列表

java.lang (44 classes)	
ArithmeticException.java	ArrayIndexOutOfBoundsException.java
ArrayStoreException.java	Boolean.java
Byte.java	Character.java
Class.java	ClassCastException.java
ClassNotFoundException.java	Double.java
Error.java	Exception.java
Float.java	FloatingDecimal.java
IllegalAccessException.java	IllegalArgumentException.java
IllegalMonitorStateException.java	IllegalStateException.java
IllegalThreadStateException.java	IndexOutOfBoundsException.java
InstantiationException.java	Integer.java
InterruptedException.java	Long.java
Math.java	NegativeArraySizeException.java
NoClassDefFoundError.java	NullPointerException.java
NumberFormatException.java	Object.java
OutOfMemoryError.java	Runnable.java
Runtime.java	RuntimeException.java
SecurityException.java	Short.java
String.java	StringBuffer.java
StringIndexOutOfBoundsException.java	System.java
Thread.java	ThreadDeath.java
Throwable.java	VirtualMachineError.java
java.lang.ref(2 files)	
Reference.java	WeakReference.java

表格 6 MMES AWT 所需的 Java.io 套件 class 列表

Java.io (18 files)	
ByteArrayInputStream.java	ByteArrayOutputStream.java
DataInput.java	DataInputStream.java
DataOutput.java	DataOutputStream.java
EOFException.java	InputStream.java
InputStreamReader.java	InterruptedIOException.java
IOException.java	OutputStream.java
OutputStreamWriter.java	PrintStream.java
Reader.java	UnsupportedEncodingException.java
UTFDataFormatException.java	Writer.java

表格 7 MMES AWT 所需的 Java.util 套件 class 列表

java.util (14 classes)	
Calendar.java	Date.java
EmptyStackException.java	Enumeration.java
EventObject.java	Hashtable.java
NoSuchElementException.java	Random.java
Stack.java	Timer.java
TimerTask.java	TimeZone.java
Vector.java	EventListener.java

。

4.2.2 使用 Native Interface

在傳統的 Java 平台，是使用 Java Native Interface 來達到兩個目的，第一就是希望平台支援的 native function 不要因為不同虛擬機而做修改，所以透過使用統一介面來實作，方便在移植時做最少的修改。還有也提供在 Java level 使用的方便，Java programmer 可以寫程式時可以動態的連結 native library 或是 native function。JNI 的功能雖然好，但對於嵌入式環境還是有一些缺點，一是這機制昂貴而且需要使用到特定的記憶體，並且在呼叫的過程負擔比較大，時間上還有需要 OS 支援。再來是動態使用 native function，是用 function pointer 的方式，會有安全性的考量，嵌入式環境中如果沒有辦法提供完整的安全模式，將會很危險。就上述的種種情況，會希望能有 native interface 適合應用在更低電源，記憶體更有限的嵌入式環境中。比 JNI 更省資源、更安全的做法。KNI 就是在這樣情況下產生的。

KNI 的優點是在嵌入式環境可以更方便開發新軟體或功能。軟體開發的方便性與自由度很大時，就可以使裝置的可能性變多，對於整體提升價值很大幫助。在撰寫 native code 時也變得更簡單了，KNI 將邏輯設計與虛擬機細節處理獨立分開，不同於 JNI 與虛擬機相依性這麼高，使用 JNI 撰寫 native function 需要了解 garbage collection、物件的資訊、class 的資料結構、檔案系統或是其它虛擬機操作細節。在撰寫 native code 時使用 KNI 時不需要考慮了，簡單來說，KNI 支援的是 JNI 處理邏輯的部分，轉換到不同嵌入式環境時不用做大幅度的修改。

KNI 和 JNI 有以下幾點的不同:KNI 只支援 Java AWT 實作層的 API，也就是給開發嵌入式環境時在改寫系統 class 可以 access 的，不是給 programmer 使用的。native 層支援 Java 層的 function 是特定的，透過 Java 層 API 傳遞資訊給 native 層實作，programmer 不會直接使用到 native function。KNI 也不允許 native 層呼叫 Java 層的 function，或是直接存取 Java 層的資訊。因為 KNI 是靜態呼叫的方式，所以是 native function 是在虛擬機內部，只限定特定虛擬機使用，不能與其他虛擬機共享 native function。JNI 和 KNI 的整理比較如表格 8。

表格 8 JNI 與 KNI 的比較

Native interface / Native method	JNI	KNI
系統呼叫的方式	動態鏈結，利用 function pointer	靜態呼叫(static)
耗費系統資源	多、昂貴 (時間、OS) (特定的記憶體)	少、便宜
存取範圍	Programmer access	System class access
共享性	可以多台虛擬機共享	特定虛擬機
安全性	差，可被駭客破壞	佳，JVM執行檔內
呼叫Java層method、建立物件	可以	不可以
呼叫native層method	可以	可以

Java class 中所有需要被 native 層實作的 function 會宣告成 native，這種 Java class 被編譯同時，透過 JCC(JavaCodeCompact)工具來產生 JVM 來源碼之一的，nativnativeFunctionTableUnix.c，接著 JVM 進行編譯時就會將 native function 的實作連結到 JVM 內部。JCC 是一個由 Java 寫成的工具，它可以編譯所有 Java class 並且產生 nativnativeFunctionTableUnix.c 。在檔案內容中列出所有在 Java class 中被宣告 native 的 function 在 native 層的 prototype ，再利用此 native function prototype 與 KNI 來實作 function 內容。

在 MMES AWT 中需要實作 native function 有我們加入的 MMESToolkit、MMESGraphics、EventPostThread，以及原本存在的 Java.lang.Math。Java.lang.Math 增加的 native function，log 和 exp，由蔡純仁教授產生 s_exp.c、s_log.c、直接在其中撰寫這兩個功能。此外，在這邊我們也產生 nativeAWT.c、xdraw_awt.h、xevent_awt.h、xdraw_awt.c 四個檔案，來實作 MMESToolkit、MMESGraphics、Window、

EventPostThread 的 native function，以下就列出 MMES AWT 需要實作的 native function 如表格 9。

表格 9 Native Function 列表

Class	native function name	description
MMESGraphics	error	出現error時顯示訊息
	pDrawLine	畫一條線
	pCreate	建立一個Graphics
	pSetColor	設定顏色
	pSetBackground	設定背景顏色
	pCopyArea	複製記憶體區域
	pFillRect	畫一個填滿的矩形
	pClearRect	清除一個矩形區域
	pDrawArc	畫一個弧
	pFillArc	畫一個填滿的扇形
	pDrawPolyine	畫一個曲線
	pDrawPolygon	畫一個多邊形
	pFillPolygon	畫一個填滿的多邊形
	pDrawString	寫文字
	pDispose	消失一個Graphics
Math	log	計算數學log的運算
	exp	計算數學exp的運算
MMESToolkit	sync	結束後同步化其他function
	pInit	初始化一開始的畫圖環境
Window	pChangeCursor	改變滑鼠的顯圖
EventPostThread	pNextEvent	擷取native層的Event

4.2.3 Native 層的支援

native graphics library 實作都透過 nativeAWT.c、xdraw_awt.c，幫忙傳遞來自 Java 層的參數，然後呼叫 Xlib 去實作。不同於 PBP 的架構，減少了 microwindow 層來幫助提高執行效率，簡化架構、並提供 thread safe 機制。軟體層的架構如圖 19。

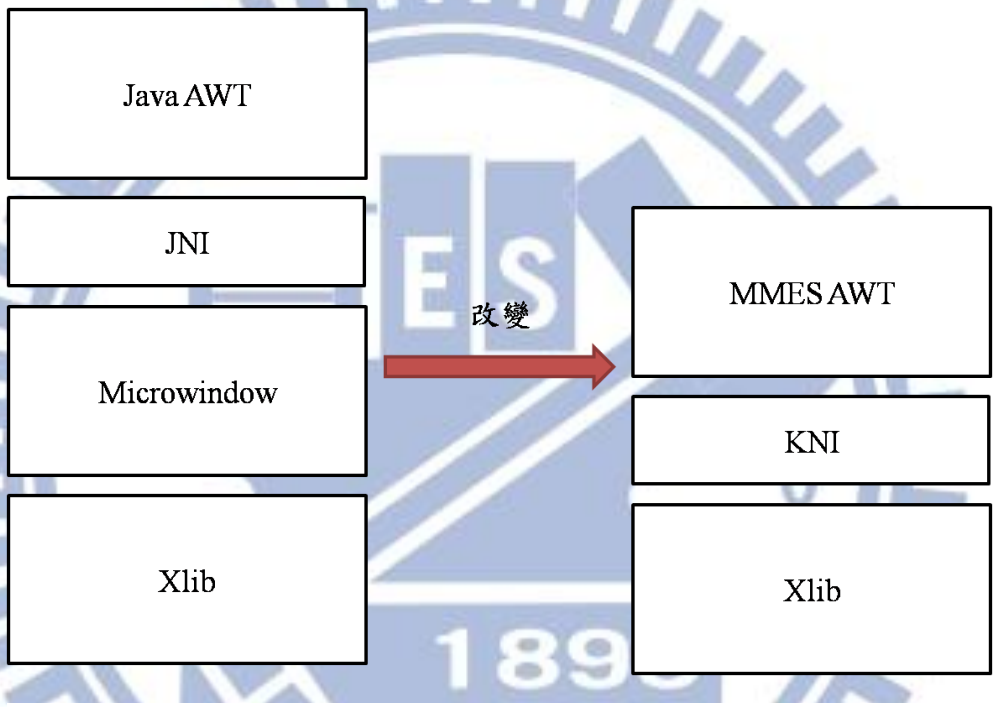


圖 19　MMES AWT 架構修改來自 PBP 的架構

第五章 系統實作的驗證

在本章中，我們提供一些實作的結果展現，以驗證我們在第四章中的設計是可行的。在 5.1 節，我們會描述一下我們選用的實作平台細節，以及我們實作的版本和原有的版本在 classes 數量、程式大小，native functions 複雜度等等的比較。在 5.2 節，我們會選擇一些常用的繪圖指令，以定性的方法來描述我們實作的版本和原本的 AWT 的差異，並以實際執行結果來確認我們實作的版本是可行的。

5.1 MMES AWT 的實驗環境細節與比較

實作平台：Java ME　CLDC

Java 虛擬機：KVM

編譯器：gcc -3-4-6

作業系統：Ubuntu 9.0.4

執行硬體：Intel(R)Core(TM)2 Duo CPU 、3.5 GB RAM

MMES AWT 與 PBP AWT 的整體比較如表格 10。

表格 10 MMES AWT 與 PBP AWT 的比較

	PBP AWT	MMES AWT
Class數量	113個	56個
Classes記憶體大小	1057KB	570KB
系統的Class數量	218 個	78個
系統Classes記憶體大小	3074KB	635KB
Native function複雜程度	高	低
字型	FreeType字型	Xlib內建字型
影像下載、呈現、處理	有	無
顏色模型	多種	一種(888RGB模型)

由表格中可以看出 MMES AWT 不管在記憶體大小、class 數量、系統 class 的支援都明顯比 PBP AWT 來的更小，更省資源，native function 的複雜度也更簡單。在字型、影像、顏色模型的部分因為在 Java 層的呼叫關係複雜、native source code 部分也是比較難懂的，還沒瞭解到可以簡化的程度。考慮到系統簡化的話，字型和顏色就使用預設的，然後不支援影像。未來要讓圖形功能更完整的話，可以從這幾個部分著手。

在 native function 複雜度上，列出 drawLine、開啟主視窗的兩個不同的 native function 完整實作過程中呼叫層級的比較，說明複雜度上的差異。如表格 11。

表格 11 Native function 複雜度比較

Native function複雜程度	舉例1: 連接到顯示器	舉例2: drawLine
PBP AWT	9層	11層
MMES AWT	7層	6層

畫出一條線這個功能是從 Graphics 呼叫 drawLine 到完全寫到記憶體來完成。整個過程在 PBP AWT 與 MMES AWT 中如圖 20，其中 Java 層固定呼叫三層不變，native 層差別比較明顯，PBP AWT 總共需要透過 11 層，MMES AWT 只需要透過 6 層呼叫，MMES AWT 架構明顯比較簡單，除此之外，PBP AWT 中 Microwindow 大量使用 function pointer 來達成繼承的關係，又加強了架構的複雜度。

PBP AWT　Draw A Line	MMES AWT　Draw A Line
1. drawLine	1. drawLine
2. drawLine	2. drawLine
3. pDrawLine	3. pDrawLine
4. Java_java_awt_MWGraphics_pDrawLine	4.Java_java_awt_MMESGraphics_pDrawLine
5. DriverBasedGraphicsEngine_DrawLine	5. DrawLine
6. DriverBasedGraphicsEngine_DrawVerLine	6. XDrawLine
7. drawVerLineSolid	
8. X11Driver_DrawVerLineSolid	
9. XDrawLine	
10. FB32Driver_drawverline	
11. assign pixel values to address	

圖 20 Draw a line 在 MMES AWT 與 PBP AWT 的比較

在建立 native 層畫圖環境時會先連接到顯示器，一旦連接後才能開始畫圖工作。建立 native 層畫圖功能是 Frame( )開始到呼叫到 xlib 的 XOpenDisplay()才算真正建立，這過程在 PBP AWT 與 MMES AWT 中如圖 21。PBP AWT 需要透過 9 層，MMES AWT 在 Java 層少了 GraphicsEnvironment，native 層又少了 X11Driver，所以只需透過 7 層。

PBP AWT 連接到顯示器	MMES AWT 連接到顯示器
1. Frame()	1. Frame()
2. Window()	2. Window()
3.GraphicsEnvironment.getLocalGraphicsEnvironment()	3. Toolkit.getDefaultToolkit()
4. Toolkit.getDefaultToolkit()	4. MMESToolkit()
5. MWGraphicsEnvironment()	5. getLocalGraphicsEnvironment()
6. Init()	6. pInit()
7. mwawt_hardwareInterface->OpenScreen()	7. XOpenDisplay()
8. X11Driver_New()	
9. XOpenDisplay()	

圖 21 連接到顯示器在 MMES AWT 與 PBP AWT 的比較

5.2 輸出功能實現和 Draw method 的比較

在這邊我們實作程式是 PBPDemoFrame 中的 DrawDemo，以下進行一些定性的比較。

圖 23 DrawDemo 在 PBP AWT

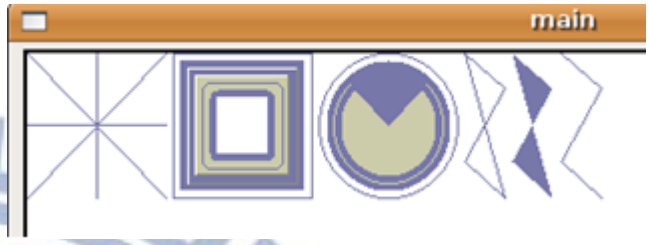


圖 22 DrawDemo 在 MMES AWT 中

執行結果的比較：根據圖 22、圖 23 來看，大致上相同。

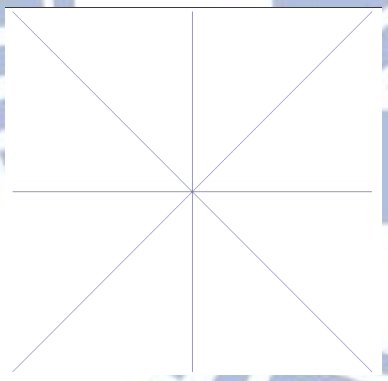


圖 25 DrawLine 在 PBP AWT 中

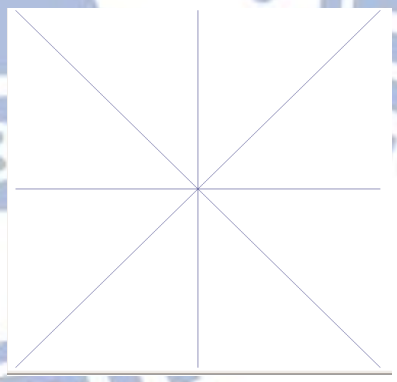


圖 24 DrawLine 在 MMES AWT 中

Drawline 的比較：根據圖 24、圖 25，幾乎一模一樣，沒有差別。

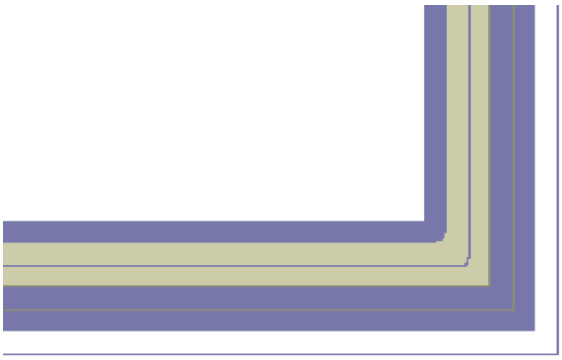


圖 26 DrawRectangle 在 PBP AWT 中

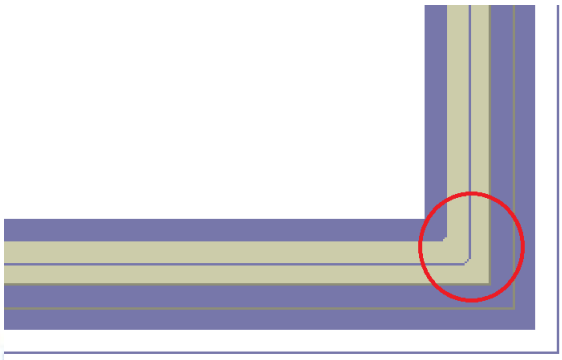


圖 27 DrawRectangle 在 MMES AWT 中

DrawRectangle 的比較: 根據圖 26、圖 27，大致相同只有 drawRoundRectangle 中在矩形的轉彎處 MMES AWT 比較平整。

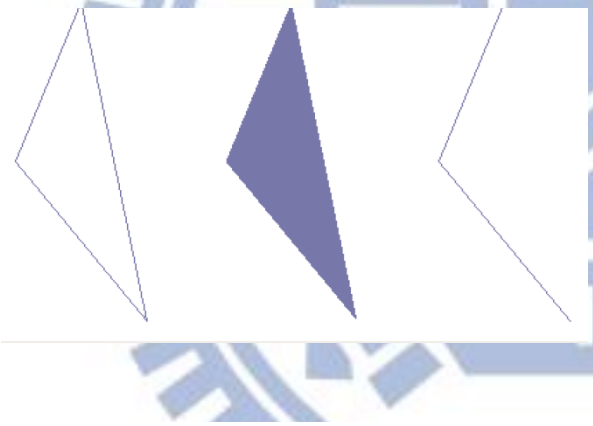


圖 28 畫多邊形在 PBP AWT 中

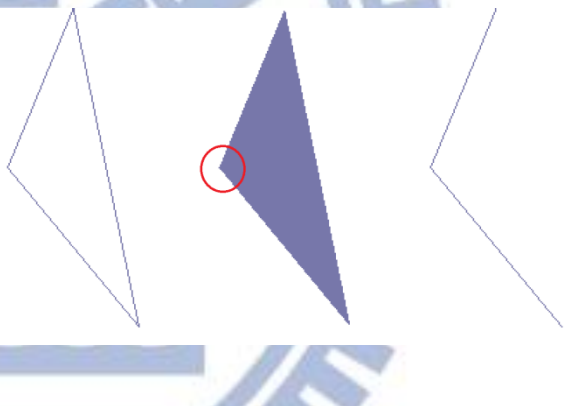


圖 29 畫多邊形在 MMES AWT 中

DrawPolygon, Polyline 的比較：根據圖 28、圖 29，MMES AWT 在多邊形的角度比較清楚。

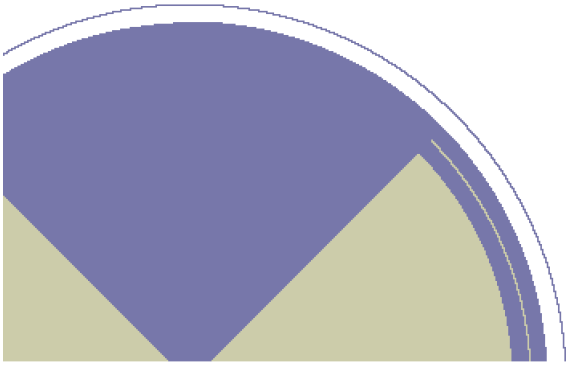


圖 31 畫圓在 PBP AWT 中

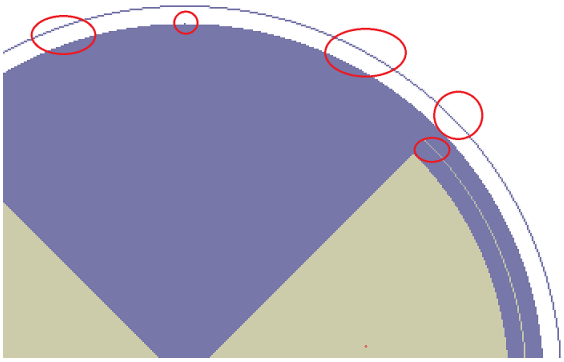


圖 30 畫圓在 MMES AWT 中

DrawArc 的比較:根據圖 30、圖 31，MMES AWT 的線條比較細，弧線段有間斷、比較不連續，圓的頂部會有多一點。

5.3 輸入功能實現與 Mosue,Keyboard 的實驗結果

為了實現輸入功能，利用 PBPDemoFrame 的 KeyboardDemo 和 MouseDemo 來實驗， 測試 Keyboard 的執行結果如下：

按下「k」鍵時，出現的結果

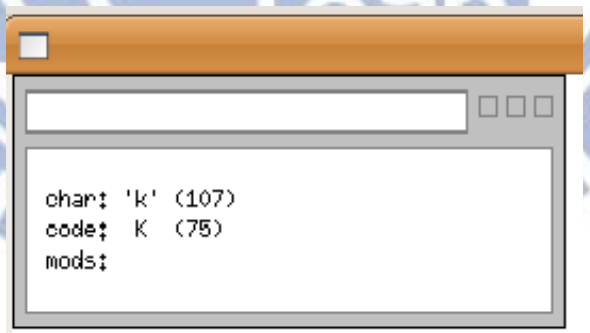


圖 32 結果一

按下「Shift」鍵時出現的結果：

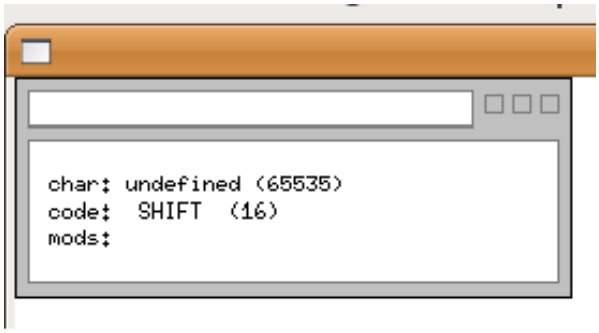


圖 33 結果二

按下「Control」鍵時出現的結果：

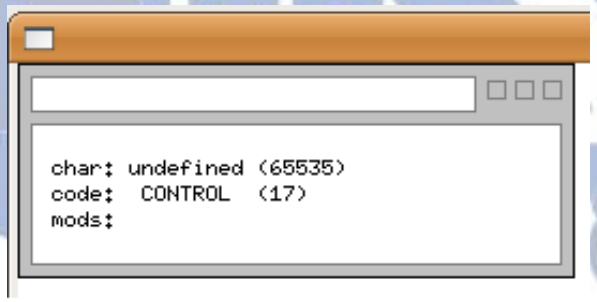


圖 34 結果三

測試 Mouse 的執行結果：

滑鼠移動到規定範圍內會有紅色框出現

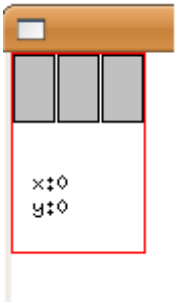


圖 35 結果四

滑鼠按了左鍵會顯示

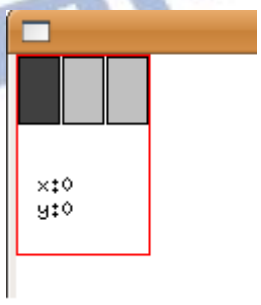


圖 36 結果五

第六章 結論與未來展望

在產生 MMES AWT 的過程中，學生花了一年的時間看懂 JavaME CDC/PBP AWT，還有 Microwindows 的 sorce code，並且分析架構與了解如何簡化。6 個月的時間產生 MMES AWT 並且移植到 JavaME CLDC 環境確保能正確執行。3 個月的時間測試 PaintEvent、KeyEvent、MouseEvent 的測試可以正確執行無誤。3 個月的時間統整設計概念、整理資料、歸納結論到整個專案明確完整描述。在描述整個專案的部分，除了用原本的 Java 相關資訊之外，試著想用更簡單的方式讓人一目了然，所以花了些心力找描述方法，直到找到了設計模式能夠明確表示系統才算告一段落。最後 2 個月時間完成論文。

MMES AWT 為一個 Java AWT(抽象視窗工具)最基本的 class 集合，class 數較少，只有 56 個 class，使用到的 system class 較少、並且由 KNI 來支援，需要平台資源較少少。這樣子一個最簡單、最基本的 Java AWT 的模型，並且確保可以正常運行，對於未來有許多發展的價值。在較少資源的嵌入式環境，不考慮動態鏈結的情況下，這將是最適合的模型。目前 GUI 系統由於功能越來越多元，架構也越來越複雜，這模型為 GUI 的基本架構，可以供 GUI 架構修改、改良的基礎；MMES AWT 系統包含最基本運作的 class 與佔最少環境資源的特性，之後若要移植、或是改由硬體環境實作，都最為簡單、方便實作；因為架構最簡單易懂，設計模式也少，對於 GUI 架構的初學者來說，是能初步了解的好範例、入門的基礎。否則都必須要了解很多觀念與資訊之後，從中找出相對應的程式行為，經過一番化繁為簡、歸納之後才能真正了解，這樣省去了解複雜架構的時間。未來 GUI 功能會越來越多元與千變萬化，例如變成 3D(3-degree)的 UI，或是設計上需要增加新的功能，都可以 MMES AWT 為基礎，深入的修改或擴充也提供了大方向。未來希望可以在實驗室的 Java 硬體執行環境上實作，提供要實現 GUI 功能的參考。若建立完整之後，預計也會在這基礎上建立 3D 圖形庫、實現 3D GUI 的功能。

【附錄一】

Java AWT in PBP 的classes (108 classes)	
Event管理與分配	EventDispatchThread EventQueue LightweightDispatcher java.util.Timer java.util.TimerThread java.util.TaskQueue
Event (java.awt / java.awt.event)	ActionEvent ActionListener AdjustmentEvent ComponentEvent ContainerEvent FocusEvent InputEvent InvocationEvent ItemEvent KeyEvent MouseEvent PaintEvent TextEvent WindowEvent ActiveEvent AWTEvent
EventListener (java.awt / java.awt.event)	AdjustmentListener AWTEventListener ComponentAdapter ComponentListener ContainerAdapter ContainerListener FocusAdapter FocusListener ItemListener KeyAdapter KeyListener MouseAdapter MouseListener MouseMotionAdapter MouseMotionListener TextListener WindowAdapter WindowListener EventQueueListener
抽象視窗元件	MenuComponent MenuContainer Component Container Frame Canvas Window Dialog
平台工具組	Graphics Graphics2D Toolkit GraphicsConfiguration GraphicsDevice GraphicsEnvironment
顏色(java.awt/ java.awt.color)	ColorSpace Color SystemColor
字型	Font FontMetrics
特性	Composite AlphaComposite Conditional ItemSelectable Transparency Adjustable
滑鼠	Cursor
基本定義	Dimension Insets Point Polygon Rectangle
權限管理	AWTPermission java.security.BasicPermission
Error/Exception	AWTError AWTException IllegalComponentStateException
Debug機制、避免錯誤	Java.sun.awt.nullGraphics AWTEventMulticaster Java.sun.awt.ConstrainableGraphics AWTEventListener
Layout管理	LayoutManager LayoutManager2 BorderLayout CardLayout FlowLayout GridBagLayout GridBagConstraints GridLayout
影像處理 (java.awt/java.awt.image)	AreaAveragingScaleFilter BufferedImage ColorModel CropImageFilter DataBuffer DirectColorModel FilteredImageSource ImageConsumer ImageFilter ImageObserver ImageProducer IndexColorModel MemoryImageSource PixelGrabber RasterFormatException ReplicateScaleFilter RGBImageFilter Image MediaTracker

【附錄二】

java.awt (25 classes)	
Adjustable.java	AWTEvent.java
AWTEventMulticaster.java	Color.java
Component.java	Conditional.java
Container.java	Cursor.java
Dimension.java	EventDispatchThread.java
EventPostThread.java	EventQueue.java
Frame.java	Graphics.java
IllegalComponentStateException.java	Insets.java
LightwightDispatcher.java	MMESGraphics.java
MMESToolkit.java	Point.java
Polygon.java	Rectangle.java
Shape.java	Toolkit.java
Window.java	

【附錄三】

java.awt.event (31 classes)	
ActionEvent.java	ActionListener.java
AdjustmentEvent.java	AdjustmentListener.java
Component.java	ComponentListener.java
ContainerAdapter.java	ContainerEvent.java
ContainerListener.java	FocusAdapter.java
FocusEvent.java	FocusListener.java
InputEvent.java	InvocationEvent.java
ItemEvent.java	ItemListener.java
ItemSelectable.java	KeyAdapter.java
KeyEvent.java	KeyListener.java
MouseAdapter.java	MouseEvent.java
MouseListener.java	MouseMotionAdapter.java
MouseMotionListener.java	PaintEvent.java
TextEvent.java	TextListener.java
WindowAdapter.java	WindowEvent.java
WindowListener.java	

參考文獻

[1] ”The Java™ Native Interface Programmer’s Guide and Specification”,Sun Microsystems, Inc,1999

[2]E. GammaHelm, R. Johnson, and J. VlissidesR.,” Design Patterns: Elements of Reusable ObjectOriented Software”, Addison-Wesley professional computing series. 2002

[3]Alan Shalloway,James R Trott.,鍾永哲,“設計模式入門-----物件導向設計的新觀點”, 台灣培生教育出版有限公司 2002

[4]Knudsen, J.,”JavaJava 2D 圖學技術”, O'REILLY. 2000

[5]Nagaratnam,鄭智仁,詹韋佐,”JavaJava AWT API BIBLE”,松格資訊有限公司 2000

[6]ZukowskiJohn,”Java AWT Reference”,O'Reilly Media 1997

[7]Canfora, G., Santo, G. D., & Zimeo, E.,”Toward Seamless Migration of Java AWT-Based Applications to Personal Wireless Devices”, *11th Working Conference on Reverse Engineering of IEEE*. 2004

[8]E. GammaHelm, R. Johnson, and J. VlissidesR.,”Design Patterns: Elements of Reusable ObjectOriented Software”, Addison-Wesley professional computing series. 2002

[9] Canfora, G., Santo, G. D, Zimeo, E ,“Developing Java-AWT Thin-Client Applications for Limited Devices”, *IEEE INTERNET COMPUTING*. IEEE Computer Society. 2005

[10]Shi, N., & Olsson, R. A.” Reverse Engineering of Design Patterns from Java Source Code”, *IEEE International Conference on Automated Software Engineering.* 2006

[11] Mcdowell, Brucer Montague,”JavaJavaCAM: Trimming Java down to size”, IEEE Internet Computing 1998.

[12]Wang, P.-C., Lin, C.-Y., Lin, C.-L., Li, Y.-F., Yang, C.-K., Hou, T.-W.,” A Portable Feather-Weight Java-based Graphic Library for Embedded Systems on Java Processor”, *International Conference on Intelligent Information Hiding and Multimedia Signal Processing of IEEE* 2009.

[13]T N Nigam, I.-K., T V Prabhakar, I.-K.,“Platform Independent Tool for Designing Quality Graphical User Interfaces”, *IEEE* 1997

[14] Jorg Niere, Wilhelm Schafer, Jorg P. Wadsack &Lothar Wendehals.,”Towards Pattern-Based Design Recovery”, ICSE 2002

[15]”Mobile Media API (MMAPI); JSR 135”,Oracle http://Java.sun.com/products/mmapi/.

[16]”Mobile Information Device Profile (MIDP); JSR 118”,Oracle http://www.oracle.com/technetwork/Java/index-jsp-138820.html.

[17]”Wireless Messaging API (WMA); JSR 120, JSR 205”,Oracle ,http://Java.sun.com/products/wma/.

[18]楊程凱,”Design and Implementation of a Portable Java-based Graphic library in JVM for Embedded System”,國立成功大學 工程科學系 碩論 2006

[19]李奕坊,”Using Java Processor to Accelerator the Feather-weight Graphic Library in Embedded System”, 國立成功大學 工程科學學系 碩論. 2008

[20] Paul Gestwicki, Bharat Jayaraman,”Interactive Visualization of Java Programs”, IEEE 2002

[21]Min-Hong Chen, Feng-Cheng Chang, Hsueh-Ming Hang,”Design and Implementation of an MHP Video and Graphics Subsystem“, IEEE 2006

[22] Abhyudai Shanker , Somya Lal,” Android Porting Concepts”, *IEEE* 2011