國立交通大學

資訊科學與工程研究所

碩士論文

應用介面產生器於介面開發流程之研究

A Study of Applying Interface Generator to Interface Development Process

研究生：林靖哲

指導教授：陳登吉　教授

中華民國　一百　年七月

應用介面產生器於介面開發流程之研究

A Study of Applying Interface Generator to Interface Development Process

研 究 生：林靖哲　　Student：Ching-Che Lin

指導教授：陳登吉　　Advisor：Deng-Jyi Chen

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2011

Hsinchu, Taiwan, Republic of China

中華民國一百年七月

應用介面產生器於介面開發流程之研究

學生：林靖哲　　指導教授：陳登吉博士

國立交通大學資訊科學與工程研究所碩士班

摘要

隨著科技的進步，人機介面越來越普遍被應用在日常生活中，像是Wii, iPad。每當有新辨識器推出的時候，軟體開發人員需要為這個辨識器撰寫一份介面程式，供銜接辨識器與應用軟體。我們所知的介面程式的開發方式有二種。一是開發人員直接對介面程式進行撰寫。二是利用程式產生器產生介面程式。第二種方法開發人員不需直接對介面程式進行撰寫。相對的，需要了解介面產生器的使用方法以及該介面產生器所需的specification code的規範。

本研究將針對上述兩種開發方式進行分析與比較，檢視兩種開發方式流程上的差異，同時採用這兩種開發方式實際開發介面程式。透過實際的開發，了解介面程式開發所需要的時間，並以COCOMO II模型搭配開發介面程式使用的程式碼行數估算開發所需時間成本，佐證實際開發所需時間的合理性。由比較結果，得出以下結論：在本研究的環境限制之下，透過介面產生器產生介面程式比直接撰寫介面程式，有更高的生產力。

A Study of Applying Interface Generator to Interface Development Process

Student: Ching-Che Lin Advisor:Dr. Deng-Jyi Chen

Department of Computer Science and Information Engineering
National Chiao Tung University

Abstract

Nowadays, Human Computer Interface is widely used in life, like Wii, iPad. We need to development for recognizer an interface code for interface new recognizer and application when a new recognizer is released. There are two methods to development interface code. First, the program developers write interface program directly (Hardcode). Second, use the program generator to generate the interface code. The developers do not need to write interface program directly in the second method. The relatively, the developers need to understand how to use the interface generator, and how to write the specification code which interface generator needed.

In this study, we will analysis and comparison the above methods. View the different of these two methods of development process. Furthermore, these two methods will be used to actual development interface program. Via the actual development, understanding of the interface program development time required. And use the COCOMO II model to estimate the time required for development costs, support the development time required is reasonable. Finally, we got the following conclusions. Under the constraint of this study, use the interface generator to generate interface code is more productive than hardcode.

誌謝

本論文得以完成，感謝陳登吉老師的大力協助及孜孜不倦的指導，花費了許多時間及精神，開發新的靈感、修正論文方向，並提供實驗室資源使用。感謝曾建超老師、蕭嘉宏老師給予論文及口試上的指導。也感謝學長姐們的指導與鼓勵、同學之間的互相鼓勵及學習、實驗室的學弟們的幫忙，讓我能夠順利的完成論文的寫作。

最後特別感謝我的家人，在這段時間頻繁來往台北、新竹，舟車勞頓。你們的支持與鼓勵，是支持我努力下去的最大動力。

目錄

圖目錄

表目錄

一、緒論

1.1 研究動機

隨著現今科技的進步，我們可以看到許多人機介面被應用在日常生活中，像是：運動控制器(Wii)[34]、觸控式螢幕產品(平板電腦、智慧型手機)[38][39]、語音控制系統[40][41]等等…，我們發現每當有新的辨識器出現時，開發人員就必須開發新的介面程式用來銜接新的辨識器及應用軟體。

我們知道的介面程式開發方法有兩種，一種是直接撰寫介面程式的開發方法，另一種就是透過程式產生器來產生介面程式的開發方法。

直接撰寫介面程式的開發方法，如同字面上的意思，就是由開發人員直接撰寫用於銜接辨識器與應用軟體的介面程式碼。

透過程式產生器(由於本研究產生的程式為介面程式，其後統一稱之為介面產生器)產生介面程式的開發方法[27]，則是需要撰寫介面產生器所需要，且具有一定格式規範的程式碼(specification code)，然後由介面產生器讀取撰寫好的 specification code，自動產生可銜接辨識器與應用軟體的介面程式，而非直接進行介面程式的撰寫。觀念上，就像是利用 lex 這種工具來產生 lexical analyzer 的方法，或像是利用 yacc 這個工具來產生 parser 的方法。

1.2 研究目的

雖然開發介面程式的方法不同，可是最終都還是必須達到能夠與應用軟體互動的目的。本研究希望尋找利用不同的介面程式開發方法，在開發介面程式時的差異。所以，本研究的主要目的，將以上一個章節所提及的兩種介面程式開發方法作為研究對象，試圖**尋找兩種介面程式開發方法(直接撰寫介面程式的開發方法、透過程式產生器產生介面程式的開發方法)在開發介面程式時的差異。**

1.3 研究方法與步驟

本研究的主要目的是尋找利用兩種不同的介面程式開發方法，在開發介面程式時的差異。所以我們一開始先選定本研究所需要使用的辨識器、介面產生器。並加以修改，成為符合本研究的工具。再進一步分析兩種開發流程的特色與差異，最後統計與比較利用兩種開發方法，實際開發介面程式所需要的時間與程式碼規模。

這裡，我們在選定好作為範例的應用軟體[8]後，去信希望能取得該辨識器與應用軟體之介面程式的開發時間(見附錄)，由於對方至今在尚無回信，本研究無可以比較的對象，故本研究選擇了一個替代方案：使用軟體工程中的軟體開發成本估算模型[33]，來進行軟體開發時間成本的估算，用以佐證實際開發介面程式所需時間的合理性。本研究的步驟如下

1. 選擇本研究所使用的辨識器與介面產生器，且加以修改成為本研究可用之辨識器與介面產生器。
2. 分析兩種介面程式開發方法的流程與差異比較。
3. 利用兩種介面程式開發方法，實際開發介面程式並尋找開發時間與程式碼規模的差異，最後以軟體成本估算模型佐證實際開發時間的合理性。

1.4 研究範圍與限制

- 辨識器

  辨識器種類繁多，應用層面廣泛。本研究選擇的是以手勢辨識為基礎的辨識器。其中手勢辨識亦包含有影像識別、加速度感應識別、觸碰式感應識別等…。所以更準確的來說，本研究選擇的是利用加速度感應來辨識手勢的手勢辨識器。
- 應用程式

  應用程式可選擇任一普通常見的軟體。而本研究選擇的是以醫療復健為基礎的應用軟體，藉由遊戲的方式，活動使用者的腕部，已達到復健的目的。本研究並不探討此應用軟體的醫療、成效等相關議題。
- 介面程式

  用於銜接上述的辨識器及應用程式的程式，由介面程式了解辨識器的辨識結果，並將此辨識結果轉換成應用程式能讀懂得資訊。
- 介面程式產生器

  產生上述介面程式的程式產生器，避免介面程式重複開發所浪費的時間成本，節省開發資源。

本研究僅探討有無利用介面產生器於 Windows 作業系統平台上開發介面程式，在流程、時間及程式碼規模的差異。作業環境為 Windows 7 並可相容於 Windows XP。

1.5 章節概要

本研究共分為六個章節，各章節之內容摘要依序如下：

第一章：敘述本研究的研究動機、研究目的及研究方法與步驟。

第二章：相關研究探討，對本研究所選擇與使用的辨識器、介面產生器，以及其應用與後期會使用的軟體成本模型作說明。

第三章：分析介面程式開發方法的流程與差異比較。

第四章：利用兩種介面程式開發方法實際開發介面程式，記錄開發時間與程式碼規模並進行差異比較。

第五章：展示。

第六章：說明本研究的結果與未來尚可探討的議題。

二、相關研究

2.1 辨識器

市面上的辨識器種類十分多種[34][35][36][37][38][39][40][41]，有語音辨識器、手勢辨識器、動作辨識器等等…，下面我們將針對上述提及的辨識器作簡單的介紹。

2.1.1 辨識器的選擇

- 語音辨識器[40][41]

語音辨識器是將人類語音中的詞彙內容作為計算機可讀的輸入，採用分離字辨識或連續語音辨識等方式，識別出使用者所說的詞彙[32]。其中分離字辨識法，就是在使用辨識器時，每說一個詞彙，就必須停頓一次，讓電腦進行辨識的動作。如今，大部分的語音辨識系統均採用連續語音辨識的方式，避免口音，說話速度、干擾的背景噪音所造成的影響，進而辨識出使用者說出的詞彙之間的界線。而且辨識器中所提供的 Context Free Grammar 也可用來解析辨識出的詞彙，進一步了解使用者所說的話的涵義。其產品例如：Microsoft Speech Recognizer[41], Simon[40].

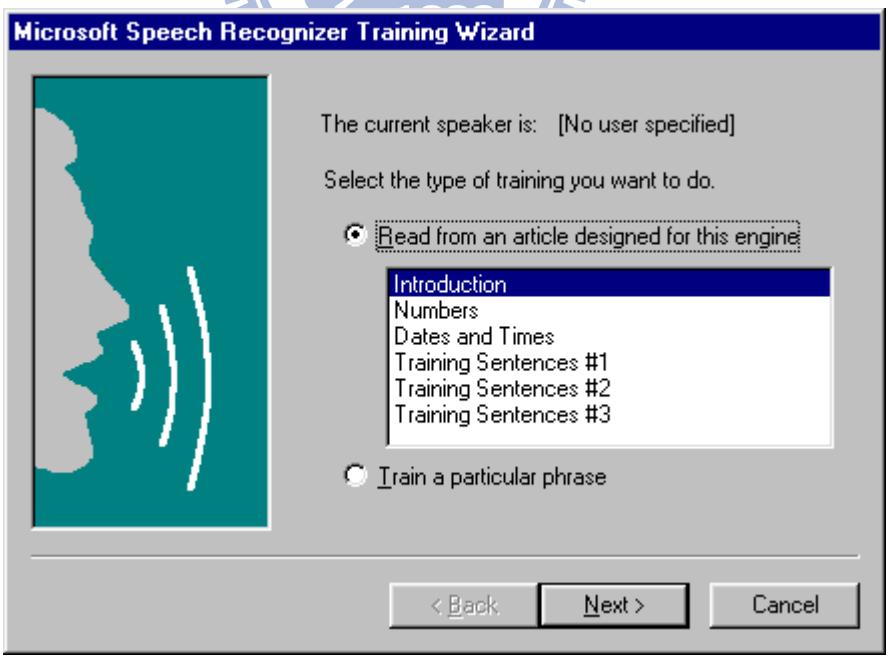


圖 1、Microsoft Speech Recognizer 使用畫面

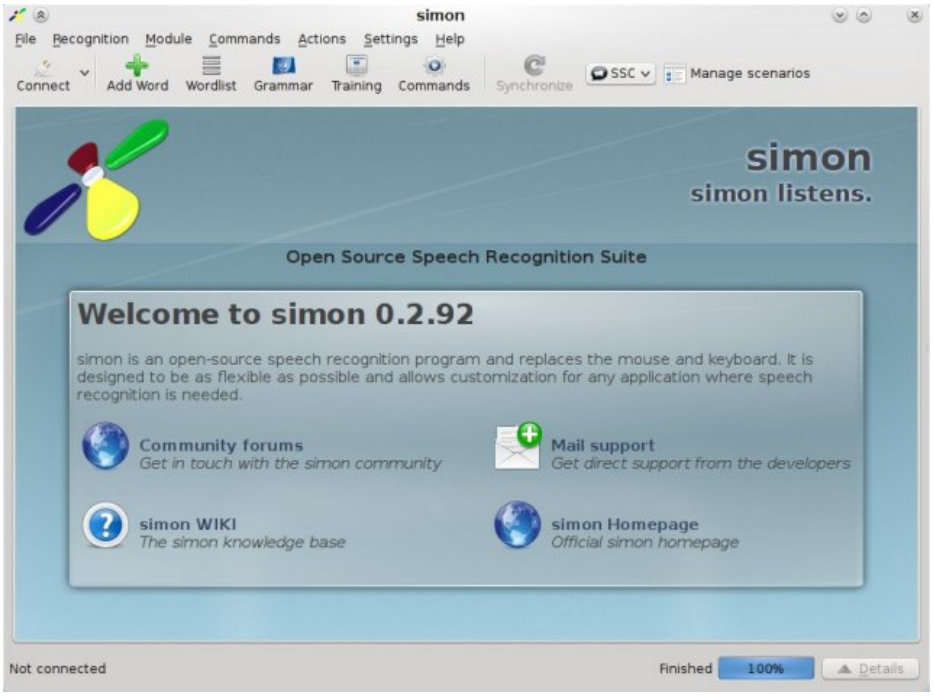


圖 2、Simon 使用畫面

- 手勢與動作辨識器[34][35][36][37][38][39]

  依據手勢與動作辨識器的使用方式，又可進一步分為觸碰式辨識與非觸碰式辨識。

  - 觸碰式[38][39]：須直接觸碰感應面板。藉由電容式或電阻式面板，感測使用者手的接觸面與移動，進一步分析觸碰點位置的變化，其產品例如：FingerWorks(ipad, iphone…), HTC touchFLO3d, HTC sense…均廣泛被使用在智慧型手機、平板式電腦上。

圖 3、FingerWorks

圖 4、HTC sense

  - 非觸碰式[34][35][36][37]：非觸碰式的手勢辨識，不需要觸碰感應面板。相對可能需要其他的感應設備，如加速度感應器(Wii)、光纖感應器(VPL DataGlove)、攝影裝置(Kinect, Mgestyk)及其他未公開感應器的產品(g-speak)。

圖 4、Wii

圖 5、DataGlove

圖 6、Kinect

圖 7、Mgestyk

在資料與產品的尋找過程中，我們發現動作感應及手勢辨識漸漸的成為辨識器的主流之一，其中 Wii 在 2005 年由日本任天堂公司所提出並產品化[34]，是屬於近年來較早提出且商品化的辨識器，在開發資源上有著比較豐富的優勢。除了本身應用於遊戲上廣為受歡迎之外，亦有許多論文及研究，探討將 Wii 應用在不同的領域，例如：教學上的電子白板[43]，或協助醫療復健的平衡訓練、手/腕部運動[3][4][5][6]等等…。

其中，將 Wii 應用於醫療復健是個新鮮的領域，同時也是本研究有興趣的方向。故本研究的辨識器選擇以 Wii 控制器(Wiimote)為基礎的辨識器。

2.1.2 Wiimote 的應用實例

Wiimote 的應用，廣為人知的遊戲部分在此就不多加說明，此章節將針對 Wiimote 應用於教學上的電子白板以及醫療復健的運動作介紹。2008 年，Johnny Lee 於個人網站[43]提出利用 Wiimote 中含有紅外線感測器與藍芽傳輸裝置的特性，與紅外線發射器及電腦結合，藉由追蹤及判斷紅外線發射點的位置，做成簡單的電子白板，並同時將此技術公開。由於該技術所需設備之價格低廉，時至今日已被使用於日常所見的教學中。

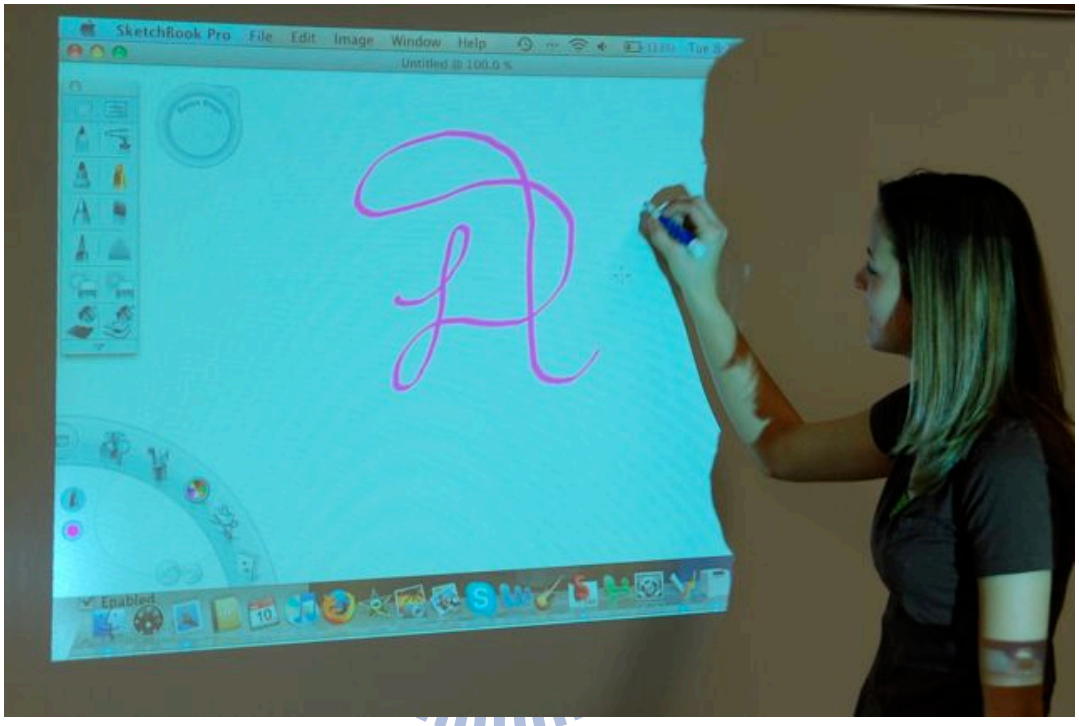


圖 8、Wiimote 應用於教學(電子白板)

另外，國內外的醫院、研究機構也嘗試利用 Wii 的遊戲或是 Wiimote 的機制，改善病患在復健上的問題，除了可增加病患的興趣、提高參與復健的意願[7]之外，搭配專業醫事人員的意見及協助，也可以達到復健的目的，作為傳統復健的另一種新選擇。

圖 9、Wii 應用於復健 I(平衡)

圖 10、Wii 應用於復健 II

最後，本研究所選擇的是藉由偵測使用者腕部伸展運動，控制古典遊戲 pong 中的平板，使平板左右移動，目的在不讓小球掉落至平板之下，同時達到腕部復健的目的[8]。同上章節之結論，本研究選擇以 Wii 控制器為基礎的辨識器，並與醫療復健應用軟體作結合，作為一個範例。

圖 11、Wiimote 應用於復健(腕部)

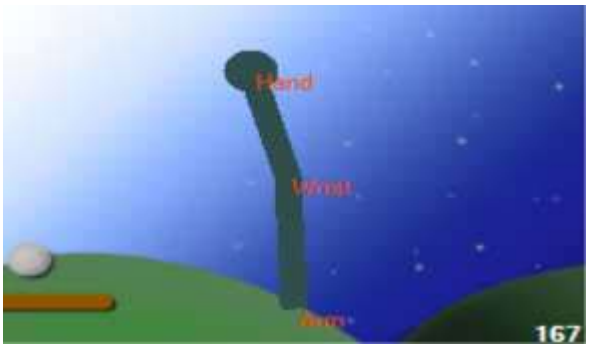


圖 12、Wiimote 應用於腕部復健的遊戲

2.1.3 Wiimote 的運作

Wiimote 本身包含了三向動作感應器、紅外線發射器及藍芽傳輸裝置，其中本研究中所利用的是三向加速度動作感應器以及用來傳輸感測到的數據的藍芽傳輸裝置，使用者在使用 Wiimote 作出動作時，三向感應器用來偵測使用者動作的 x, y, z 軸的加速度，根據感測到的數值，再藉由藍芽傳輸裝置，把感測到的數值，傳送到電腦的藍芽接收端，電腦中的辨識器將會接收由藍芽傳送來的訊號，並加以解析，分析記錄下來的數值，判斷使用者當下所作出的動作為何[1][2]。並將判斷結果送至介面程式，供介面程式參考。

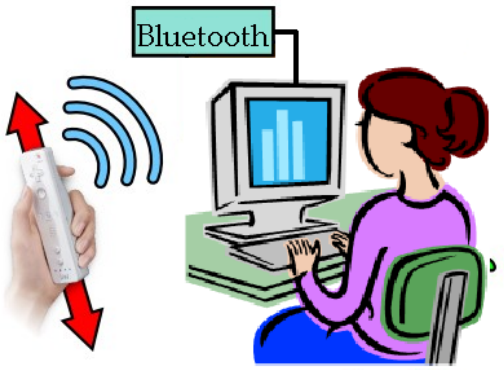


圖 13、Wiimote 藉藍芽傳遞偵測數據至電腦端概念圖

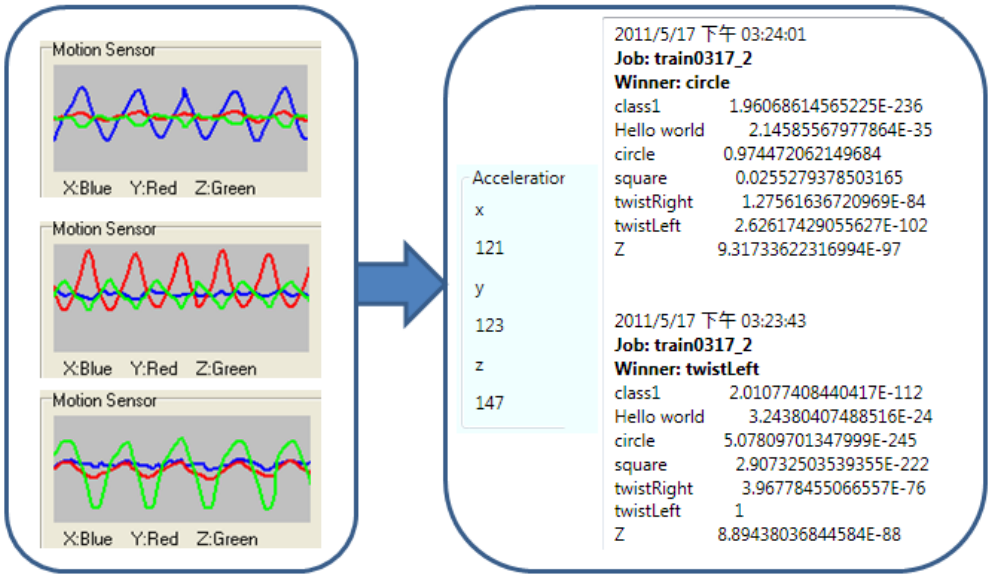


圖 14、Wiimote 與其辨識器間的運作流程

2.1.4 電腦端的辨識器–WiiGLE

如同前面章節所提，Wiimote辨識器的資源非常豐富，像是WiiGLE[42], WiiGee[44], Wiiremote, Wiiyourself…等等，此處我們選擇的電腦端的辨識器是WiiGLE。WiiGLE是由德國University of Augsburg的 Human Centered Multimedia團隊開發。

它同時提供了動作感測及動作辨識的模組，在利用藍芽連線後，便可藉由 x, y, z 三向加速度感測器的數據，及時的反應給電腦端，也可利用 Wii 的 B 鍵進行動作辨識。同時，其他種類的辨識器很多只有提供到即時的動作感測，並不提供連續動作的辨識，或是辨識器是運作在 Linux 系統上，還有一些是因為藍芽裝置的相容性問題導致無法順利使用[44]。另外 WiiGLE 內建提供了 DTW 分群法、Weka BayesNet 分群法、Weka Kstar 分群法…等數種不同的分群法，用於識別使用者的動作。WiiGLE 也提供了使用者自建動作的模組，讓使用者可以對辨識器自行訓練新的手勢動作。

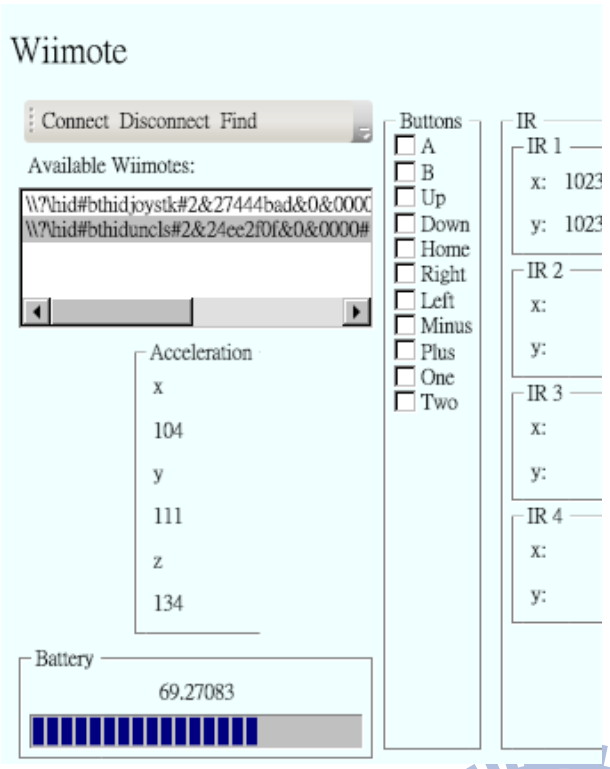


圖 15、WiiGLE 使用畫面

圖 16、WiiGLE 的辨識結果畫面

2.2 介面產生器

本研究所需要的介面產生器必須符合兩個條件

1. 由於本研究會針對辨識器與應用軟體作修正，所以必須要能取得介面產生器的程式原始碼
2. 介面產生器必須要產生能夠銜接辨識器與應用軟體的介面程式

2.2.1 介面產生器的選擇

首先我們收集了從 2000 到 2010 年發表於 IEEE 與 ACM，有關程式產生器的期刊、論文、產品並以其用途加以分類，大致上分成了

- 使用者介面程式產生器[45][46]
- UML 程式產生器。[10][12][16]
- 硬體程式產生器[9][11][13][14][29][30]
- 編譯器程式產生器[28][48]
- 介面程式產生器[26][27]
- 其他[15][17][18][31][47]

綜合上述所蒐集到各種程式產生器的資料，可得表 1。由表 1 得知，唯有介面程式產生器類別中，Generic Interface Bridge[27]這份文獻及其產生器是符合我們的條件。Generic Interface Bridge 在概念上，可以產生兩個部分，一個是銜接前端辨識器的 Frontend-Interface，另一個是用來銜接後端應用軟體的 Backend-Interface，Generic Interface Bridge 系統藉由這樣的方式，銜接了辨識器與應用軟體。

再回到一開始選擇程式產生器條件的部分，由於本研究是需要取得介面產生器並利用產生出來的介面程式銜接辨識器與應用軟體兩端，所以選擇以交大軟體工程實驗室所提出並發表於期刊上的通用橋接介面系統[27](Generic Interface Bridge，以下簡稱 GIB)，作為本研究的介面產生器。

類別	用途	程式原始碼	例
GUI code generator	使用現有的元件，以視覺方式產生使用者圖形介面	商業軟體 無法取得	Visual Studio, Borland C builder
UML code generator	將 UML 轉換成目標程式碼	商業軟體 無法取得	ALTOVA, Actifsource, [10], [12], [16]
Hardware code generator	由電路圖形的方式產生 cross-compiler 所需程式或組合語言程式	商業軟體 無法取得	[9], [11], [13], [14], [29]
Compiler code generator	產生 compiler 所需的 lexical analyzer, syntactic analyzer	部分可取得	Lex, Flex, Yacc, bison…, [28]
Interface code generator	產生用於銜接 recognizer 與 application 的介面程式	可取得	Generic interface bridge
Others	程式語言 wrapper 產生 SQL 程式碼 其他…	部分可取得	Swig, [15], [17], [18], [31], [47]

表 1、程式產生器分類列表(2000-2010)

2.2.2 介面產生系統簡介

本章節我們將針對 GIB 系統[26][27]架構及其流程作簡單介紹，GIB 系統分為三大部分，Frontend-Interface, GIB Kernel, Backend-Interface 三個部分，下面會分別對各模組的功能作說明。圖 17 為 GIB 系統的架構流程圖。

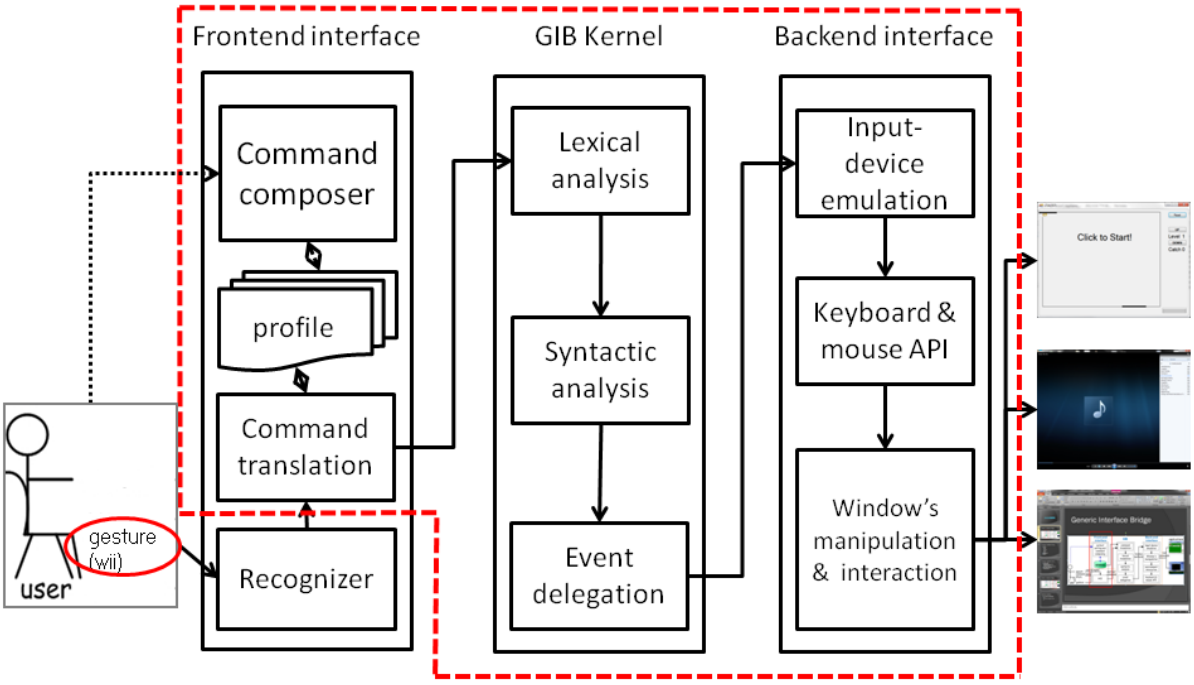


圖 17、Generic Interface Bridge 系統架構流程圖

- Frontend-Interface
  - Command composer
    使用者於使用 GIB 系統前，需先利用這個使用者介面模組，設定辨識結果所要對應到的指令，並將其存於 profile 中，最後產生 Frontend-Interface 的程式碼。
  - Command translation
    用於銜接辨識器的部分，接收由辨識器所辨識出來的結果。比對 profile 中是否有以存在的辨識結果及其對應指令。並將該對應指令傳送到 GIB Kernel 的部分。
- GIB kernel
  - Lexical / Syntactic analyzer & Event Delegation
    接受由 Command translation 所傳遞來的指令，進行字彙及句法的分析，如該指令通過分析後是屬於合法指令，則由接下來的 Event Delegation 發出事件，通知 Backend-Interface 應該作出哪些模擬動作

- Backend-interface
  - Input-device emulator
    接受由 GIB Kernel 中 Event Delegation 部分所發出的事件，解析及判斷該指令是要進行哪個硬體裝置的模擬(滑鼠或鍵盤)，找出負責相對應的硬體設備。
  - Keyboard & mouse API
    使用滑鼠、鍵盤的 API 程式，實際模擬滑鼠的移動、點擊或鍵盤敲擊。

使用流程上：

1. 使用者在使用 GIB 系統前，先利用 Frontend-Interface 中的 Command composer 部分進行判斷結果與對應指令的設定。並將其存於 profile 中。
2. 接下來，使用者就可以利用辨識器進行操作，其操作過程經由辨識器辨識出結果後，送到 Command translation 比對是否存在該辨識結果的對應指令
3. 若相對應指令存在，則將該指令傳送到 GIB Kernel 進行字彙與句法的分析，如果該指令合法，則由 Event Delegation 發出事件。
4. 由 Event Delegation 所發出的事件將告訴 Backend-Interface 應該利用滑鼠或鍵盤的 API 程式指揮滑鼠、鍵盤進行什麼樣的動作與應用軟體互動。

使用者就利用這樣子的流程，藉由 GIB 系統與應用軟體進行互動。

2.2.3 介面產生系統應用在本研究的限制

當我們實際取得 GIB 系統並應用於本研究時，我們發現了原有 GIB 系統上的一些限制，詳細情況如下：

- 座標圖層上座標位置對應錯誤
  由於過去採用的是語音控制系統，並沒有簡單的移動控制，所以包含滑鼠的移動都需要由口說指令來達成，所以定位的部分為求方便，是可以設定許多的 Tag 作為座標定位，方便移動到畫面中的某處，而這些 Tag 都被放在名為 Stage 的圖層上。
  可是這也造成了座標被固定的問題，在螢幕解析度或者視窗大小改變的時候，座標並不會跟著改變，所以造成了座標偏移的問題。同時也導致了無法正確執行原有指令執行時產生錯誤。
- 圖層造成的閃爍問題
  同樣因為座標圖層的概念，物件座標等都置放在圖層上，然後再將圖層套用在應用軟體上，又因.Net Framework 版本及程式本身問題，導致圖層無法順利透明化，產生在使用應用軟體時的閃爍問題，遮蔽了原有的應用軟體，妨礙使用者執行應用軟體。
- 鍵盤模擬
  原 GIB 系統並未實踐鍵盤模擬。

- LL(0) parser
  原 GIB 系統所採用的 parser 是 LL(0)的 parser，我們知道在 parser 中 LL(0)是需要 backtracking 的動作，效率低又花費時間，因此這邊改良過去所使用的文法，並將其重新設計成 LL(1)的 parser，希望能改善其執行效率。
- Command composer
  原 Command composer 的設計，針對對應指令的部分，是必須由開發人員直接鍵入指令的方式來建構，如此一來對於不了解 GIB 文法的人很難撰寫對應指令，且原 Command composer 並無提供錯誤檢查機制，也就是說即便了解了 GIB 文法，開發人員還是有可能會製作出含有錯誤的指令且未被察覺。以上問題都對事後 GIB 系統的操作及執行，造成了一定的不穩定性。

關於上述的限制，我們都必須加以改善，GIB 系統才能順利的在本研究環境下執行。

2.3 軟體成本估算模型

由 Ian Sommerville 的軟體工程[33]一書中提到，專案估算時，是需要評估軟體工程師生產力的，這將有助於決定專案成本或時程，或是用來評估流程或技術的改善是否有效。而評估軟體工程師生產力的度量標準，通常是以"與大小有關"(程式碼行數)、"與功能有關"(功能點數與物件點數)的兩種度量方式。

2.3.1 選擇軟體成本估算模型

軟體工程中，用於估算軟體成本的模型有許多種，依據 Magne Jørgensen and Martin Shepperd 在 2007 年所發表的文獻[21]中可以發現，截至 2004 年，軟體成本估算方法大致被分為 Regression, Expert judgment, Work breakdown, Function point, Theory…等 12 種不同的估算方法。

其中使用Regression方法的COCOMO II[20][25](COnstructive COst MOdel 2000)模型占了 49%，以及使用Function Point的FPA[24](Function Point Analysis)模型也占了 22%，此兩種模型占了全部估算方法的70%以上，被廣泛使用。本研究選擇以此兩種模型進行分析，選擇一種適合本研究的軟體成本估算模型。

首先，如同該文獻的歸類，COCOMO II 模型是一個以經驗為主的模型，這個模型是從許多軟體專案收集資料，經過資料的分析後再找出最適用的公式。另外，在 Lionel C. Briand and Isabella Wieczorek 所發表文獻中 COCOMO II 被歸類為參數型，這是因為

COCOMO II 模型提供了非常多的參數，這其中包含了專案特型、開發硬體條件、開發人員經驗與素質等等…，每一個參數都有相對應的數值範圍。藉由統計程式碼行數(Lines Of source Code, LOC)代入適用的公式與正確的參數，COCOMO II 被廣泛應用於估算軟體開發成本。使用此估算方式，估算者如何選擇參數數值，將會影響往後估算的結果。

再來，FPA 的估算方式，是以使用者的角度，計算專案程式的功能點數，來估算程式的規模大小，進而估算軟體開發的成本，此方法並不參酌開發人員經驗、開發硬體環境、開發工具的熟練度，專案的困難度等等…可能影響估算結果的控制變因。使用此估算方式，功能點數的計算也會根據估算者的判斷和系統的種類而有所不同[33]。

其餘像是 Expert judgment 的估算方式需要請到多位有經驗的專家來進行分析、估算，本研究無法請到這樣的專家，故此方法亦不是用於本研究。

考慮本研究的許多不確定變因。因此本研究選擇的是 COCOMO II 模型，利用參數的方式調控各種變因，進而希望能達到準確估算開發時間成本的目的。

2.3.2 介紹 COCOMO II 模型

COCOMO 模型於 1981 年提出[25]，歷史悠久，中間經過改良，直到 2000 年推出最新的 COCOMO II.2000[20]版本。我們所採用的是 COCOMO II 的初期設計模型，可用在使用者確認需求後，初步粗略估算軟體開發成本。COCOMO II 用於初期設計模型的估算公式如下。

$$Effort = A \times Size^{B} \times M \tag{1}$$

$$B = 0.91 + 0.01\sum_{j=1}^{5} SF_j \tag{2}$$

$$M = RCPX \times RUSE \times PDIF \times PERS \times PREX \times SCED \times FCIL \tag{3}$$

公式(1)中常數係數 A，將由 COCOMO II 的統計資料作回歸分析得到。如公式(2)，指數系數 B 則由五個因素(Scale Factor)影響，包含過去經驗、開發彈性、風險程序的使用、團隊向心力、軟體成熟度等。如公式(3)，M 則是依本研究特性而調整的乘數因子，可依照產品可靠與複雜度(RCPX)、再利用需求(RUSE)、平台困難度(PDIF)、開發人員能力(PERS)、

開發人員經驗(PERX)、時程(SCED)、支援設備(FCIL)等…，7 個專案與程序的特性調整參數。

由於 COCOMO II 模型是支援瀑布模型及螺旋狀模型進行軟體開發[20]，而瀑布模型與螺旋狀模型的開發方式，其實包含了有以下部分。

- Requirement specification
- Design
- Implementation
- Integrate & Testing
- Maintenance

從 COCOMO II 的文獻[20]告訴我們，測試及整合階段比重應該在軟體大小成長時增加，相對的，實作階段比重則應該在軟體增大時減少。同時該文獻及其他資料[19][22]也顯示，近代軟體開發於各階段所占的比重如表 2。

Phase	Percentage
Plans & Requirements (LCCR-PRR)	7%
Product design (PRR-PDR)	17%
Detail design (PDR-CDR)	27%-23%
Code and unit test (CDR-UTC)	37%-29%
Integration & Test (UTC-SWAR)	19%-31%
Transition (SWAR-SAR)	12%

表 2、Waterfall Phase Distribution Percentages[20]

稍後章節，此公式將依照本研究的特性調整各參數，估算軟體開發所需之成本。

三、介面程式開發方法介紹與比較

本章節將介紹本研究所使用的兩種介面開發方法的流程，並加以比較。

3.1 介面程式開發方法 1 – Hard code

3.1.1 開發流程

如圖 18，可以發現使用直接開發方式，開發人員首先必須取得辨識器的程式原始碼，並加以閱讀、理解。再來才能著手撰寫介面程式。當介面程式撰寫完畢後，將針對不同的辨識結果去設定不同的滑鼠鍵盤對應命令。設定完成後，才可交付給使用者使用。如果使用者希望能夠修改辨識結果所代表的意義時，須回到之前的階段，請開發人員修改程式原始碼。

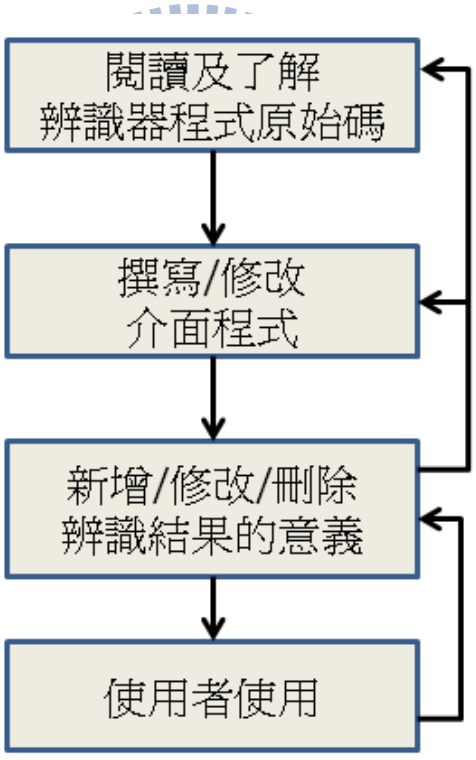


圖 18、直接撰寫開發方式開發流程

3.1.2 特色

由上述流程的介紹，可以發現以下特色。

- 開發人員採用直接撰寫介面程式的開發方法時，首先需要取得辨識器的程式原始碼後，進行閱讀以及了解辨識器原始程式碼。
- 開發人員在撰寫介面程式或辨識結果對應動作時，必須要了解如何使用滑鼠及鍵盤的 API 程式
- 開發人員在決定辨識結果對應動作時，需要用 hardcode 的方式撰寫，當使用者想修改時，也必須去更動原始程式碼

3.2 介面程式開發方法 2 – Via Interface Generator

3.2.1 開發流程

如圖 19，可以發現透過介面產生器產生介面程式的開發方式，開發人員不需要了解及閱讀辨識器的程式原始碼。可是相對的要了解介面產生器的運作方式及其指令的建構。再來需要的就是操作介面產生器並利用其內建功能建立辨識結果與指令的對應。最後產生出介面程式，我們在將介面程式放到正確的位置後。即可交付給使用者進行操作。當使用者希望修改辨識結果代表意義時，則請開發人員修改設定檔。

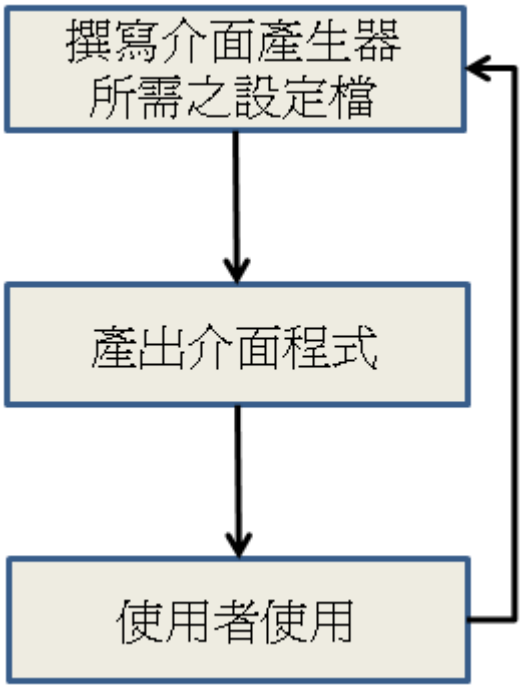


圖 19、使用介面產生器開發方式開發流程

3.2.2 特色

- 透過介面產生器的開發方式，必須要了解如何使用 GIB 系統及 Specification code 的撰寫方法。
- 開發人員因為不用直接撰寫介面程式的程式碼，所以在這裡不用了解如何使用滑鼠及鍵盤的 API 程式。
- 介面程式的產生方式，在透過介面產生器的方式中，是讀取開發人員所撰寫的 Specification code，藉由 Specification code 中的設定，來產生介面程式。

3.3 開發方法比較

整理上述兩種開發方法的特色，可得表 3。

	Hard code development	Via Interface generator
Understand the source code of recognizers	Yes	No
Understand how to use the API of mouse and keyboard	Yes	No
Development approach	Write interface program directly	Understand how to use GIB to generate interface code and write specification program for interface generator

表 3、兩種開發方式差異比較表

- 閱讀及理解辨識器原始程式碼
  相較於直接撰寫的開發方式，透過產生器的開發方式不需要直接閱讀辨識器的原始程式碼；不過相對的，透過介面產生器的開發方式，必須要了解如何使用介面產生器及 Specification code 的撰寫方法。

- 滑鼠鍵盤 API 的學習
  相較於直接撰寫的開發方式，透過產生器的開發方式不需要了解如何使用滑鼠及鍵盤的 API 程式，因為在 GIB 中，滑鼠及鍵盤的指令是由開發人員設定，經過指令合法性檢查後，動作的下達將交由 Backend-interface 來進行，所以開發人員只需要寫好 Specification code 即可，不用了解如何使用滑鼠及鍵盤的 API 程式。

- 開發方式

  最顯而易見的比較，就是直接撰寫的開發方式需要由開發人員直接撰寫介面程式的原始程式碼，而使用介面產生器的開發方式，則需要了解如何使用介面產生器、並撰寫 Specification code，而非直接的撰寫程式碼。

四、實際開發比較差異

前面章節所提，我們選擇的是將 Wiimote 應用於醫療復健的應用軟體作為本研究的範例[8]，去信詢問數次，請教應用軟體的使用情況以及介面程式的開發時間，該文獻之作者皆無回信(附錄)。所以本研究尋找了三位碩士級的研究生進行直接開發介面程式的動作，並同時記錄下開發時間及開發的程式碼行數，以供判斷。最後佐以 COCOMO II 模型，估算開發時間成本，驗證實際開發所花費時間的合理性。

4.1 實際開發時間記錄

如表 4 我們可以看到，利用直接撰寫介面程式碼的開發方式，第一位開發人員使用的大約為 6 小時、第二位開發人員使用的時間大約為 8 小時、第三位開發人員使用的開發時間大約為 6 小時。以上開發時間均包含學習時間。綜合以上紀錄，三位開發人員的平均開發時間為 7 小時。

另外觀察到，如使用介面產生器的方式開發介面程式，第一位開發人員所花費的時間為 16 分鐘、第二位開發人員所花費的開發時間為 24 分鐘、第三位開發人員所花費的開發時間為 22 分鐘。三位開發人員平均開發時間為 20 分鐘。

	Hard code development	Via Interface generator
Developer1	≈ 6 hrs	16 mins
Developer2	≈ 8 hrs	24 mins
Developer3	≈ 6 hrs	22 mins
Average	≈ 7 hrs	20 mins ≈ 0.33 hrs

表 4、開發人員實際用兩種方法開發介面程式所花費時間

4.2 COCOMO II 模型調整

依據 2.3.2 節中的分析，採用 COCOMO II 模型估算所需的開發成本，可套用公式(1)。其中常數係數 A，根據 COCOMO II 發明人 Boehm 從龐大的資料集所歸納出的結果，他推薦在初期設計模式(early design model)時，此係數應為 2.94[33]。指數係數 B 則根據開發彈性，使用風險解決程序，開發團隊向心力及組織流程成熟度而定，將會落在 0.91~1.23 之間[33]。另外，乘數因子 M 是分別由 7 個專案特性所組成，我們作出下面的設定，直接撰寫開發方式的乘數因子為 RCPX=Normal, RUSE=Low, PDIF=Low, PERS=Extra High,

PERX=Very High, SCED=Normal, FCIL=Extra High(代表數值詳見附錄)，並代入公式(3)。得 Effort 的公式如下：

$$Effort = 2.94 \times KLOC^{B} \times 0.19 \quad (4)$$

同時透介面產生器開方發的乘數因子為 RCPX=Normal, RUSE=Normal, PDIF=Low, PERS=Extra High, PERX=Low, SCED=Normal, FCIL=Extra High(代表數值詳見附錄)，並代入公式(3)。得 Effort 的公式如下：

$$Effort = 2.94 \times KLOC^{B} \times 0.31 \quad (5)$$

另外，由於本研究僅占瀑布型開發流程的 code 階段，根據 2.3.2 節的引用資料[20]，故只取公式(4)(5)的 37%-29%，又程式規模並不算大、難度亦不算高，所以決定 code 階段占全部成本的 37% [20]，故我們重新得到一個公式如下。

$$Effort_{coding} = Effort \times 37\% \quad (6)$$

4.3 實際開發行數記錄與 COCOMO II 時間估算

根據我們的紀錄，我們發現第一位開發人員開發介面所使用的程式碼行數大約是 328 行，第二位開發人員開發介面所使用的程式碼行數約為 281 行，第三位開發人員開發介面所使用的程式碼行數約為 262 行。三位開發人員平均開發介面所使用的程式碼行數約為 290 行。相對的，使用介面產生器產生介面程式的方法，是需要撰寫 specification code，又由於我們的應用軟體固定，所指定達成的動作也是固定的，所以三個開發人員開發所使用的程式碼行數也較固定，都是 16 行，得到三個開發人員平均開發的程式碼行數為 16 行。

	Hard code development	Via Interface generator
Developer1	328 lines	16 lines
Developer2	281 lines	16 lines
Developer3	262 lines	16 lines
Average	290 lines	16 lines

表 5、開發人員實際用兩種方法開發介面程式所花費程式碼行數

將程式碼行數(以千行為單位)套用公式(6)，以及 B 以 0.91 及 1.23 套用公式(6)後，求算出 Effort 的上下界。計算出來的 Effort 單位為人月(Man-Month)，為求比較方便，我們再將此結果依 COCOMO II 文獻[20]轉換成人時(1 Man-Month=152 Man-Hour)[20]。最後我們發現利用 COCOMO II 模型估算本研究的開發成本，直接撰寫的方式所估算出來的 Effort 大約在 6.8 小時至 10 小時。而透過介面產生器產生介面程式的開發方式，估算的 Effort 則需要 0.31 小時至 1.18 個小時。詳細計算過程如下。

直接撰寫開發方式

$$Effort_{Lowerbound} = 2.94 \times (0.29)^{1.23} \times 0.19 \approx 0.12[man-month]$$
$$Effort_{Upperbound} = 2.94 \times (0.29)^{0.91} \times 0.19 \approx 0.18[man-month]$$
$$Time_{Lowerbound} = Effort_{Lowerbound} \times 37\% \times 152 \approx 6.8hrs$$
$$Time_{Upperbound} = Effort_{Upperbound} \times 37\% \times 152 \approx 10hrs$$

透過介面產生器開發方式

$$Effort_{Lowerbound} = 2.94 \times (0.016)^{1.23} \times 0.31 \approx 0.005[man-month]$$
$$Effort_{Lowerbound} = 2.94 \times (0.016)^{0.91} \times 0.31 \approx 0.02[man-month]$$
$$Time_{Lowerbound} = Effort_{Lowerbound} \times 37\% \times 152 \approx 0.31hrs$$
$$Time_{Upperbound} = Effort_{Upperbound} \times 37\% \times 152 \approx 1.18hrs$$

	Average lines of code produce	Expected cost based on COCOMO II model	Average actual development time
Hard code	290 lines	6.8 ~ 10 hrs	7 hrs
Via Interface generator	16 lines	0.31 ~ 1.18 hrs	0.33 hrs

表 6、利用 COCOMO II 模型估算的開發時間與實際開發花費時間列表

由以上的實際開發記錄，佐以 COCOMO II 模型所估算的軟體開發成本，我們發現實際開發時間落在估算的範圍之內。因此我們傾向相信這份實際開發結果的數據。不過，本研究實驗數據樣本尚不足且各參數(包含前面所提的變數因子，以及人月人時的轉換、coding 所占開發比重)的選擇也需要更精確，才能得到更為準確的數據。所以我們得到以下結論，利用產生器開發的方式，比直接撰寫的方式有更高的生產力。

五、展示

展示我們將分為兩個部分。第一個部分，我們會展示利用直接撰寫介面程式開發方式所開發介面程式，實際用於銜接辨識器與應用軟體時的互動情況。之後，我們將使用介面產生器從撰寫 specification code 到產生介面程式，到把介面程式建立成 dll 檔，最後使用的時候同樣能達到操作應用軟體的目標。

5.1 利用 Hard code 方式開發介面程式

Step1.(打開 BlueSoleil)

首先我們先打開藍芽的接收器，這裡我們可以看到 Wiimote 的裝置，選擇連線後，電腦就可以接收到由 Wiimote 傳送出來的訊號。

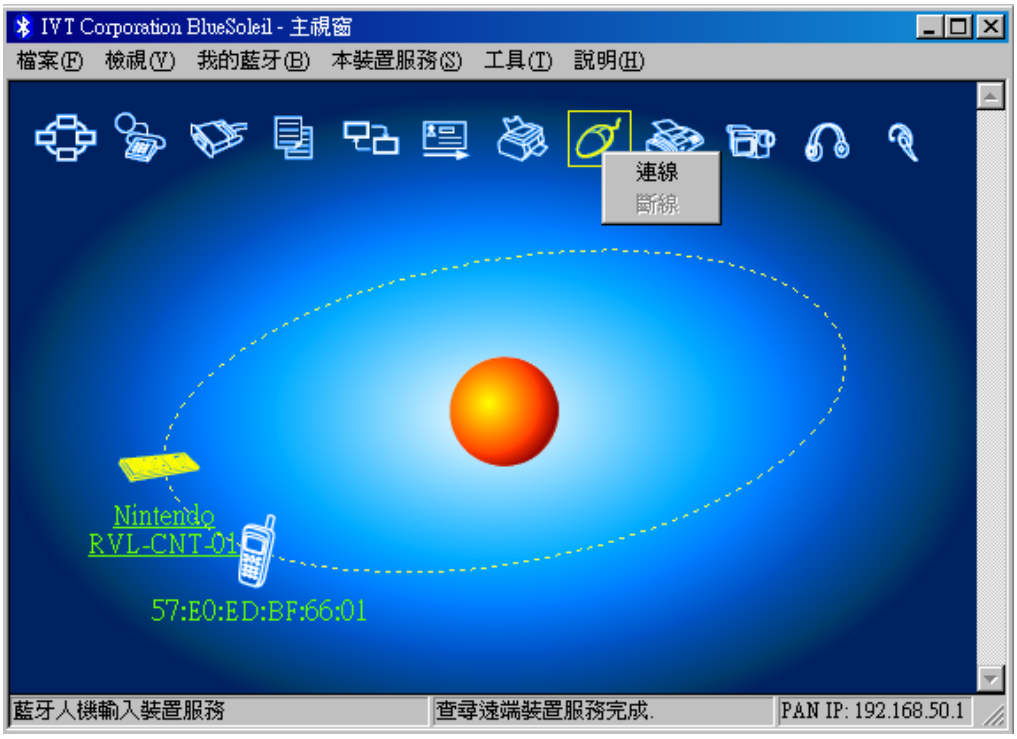


圖 20、BlueSoleil(藍芽接收器)啟動畫面

Step2.(打開 WiiGLE)

再下來開起辨識器的程式，接收由接收器得到的訊號。

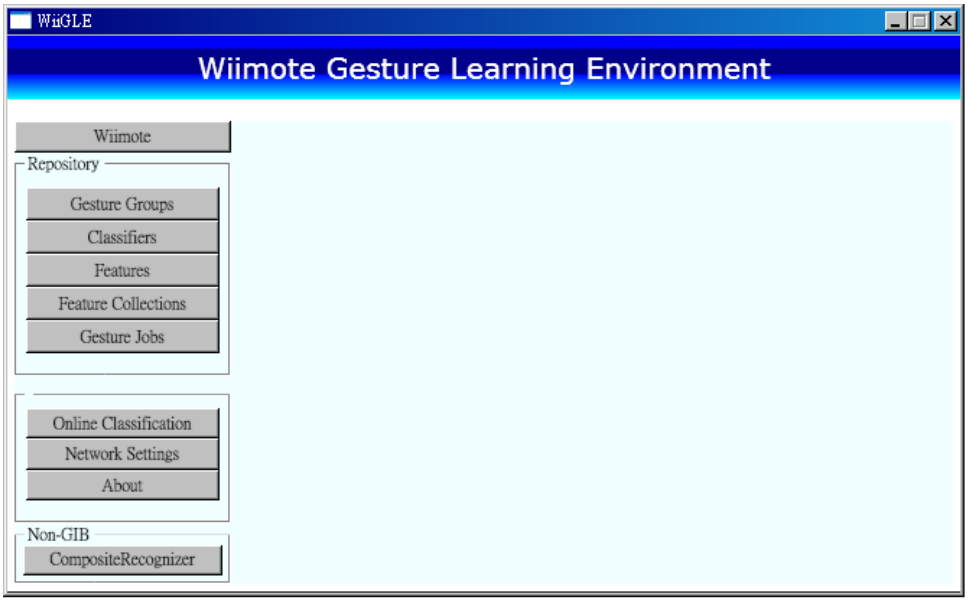


圖 21、WiiGLE 啟動

Step3.(點選 Composite Recognizer)

這裡我們選擇 Composite Recognizer 的部分，這樣可以同時進行被動辨識及主動辨識。主動辨識就是當偵測到數值時即時進行的反應。被動辨識就是當我們按下 Wiimote 的 B 鍵時，進行數值的紀錄並辨識，以辨識使用者的動作。

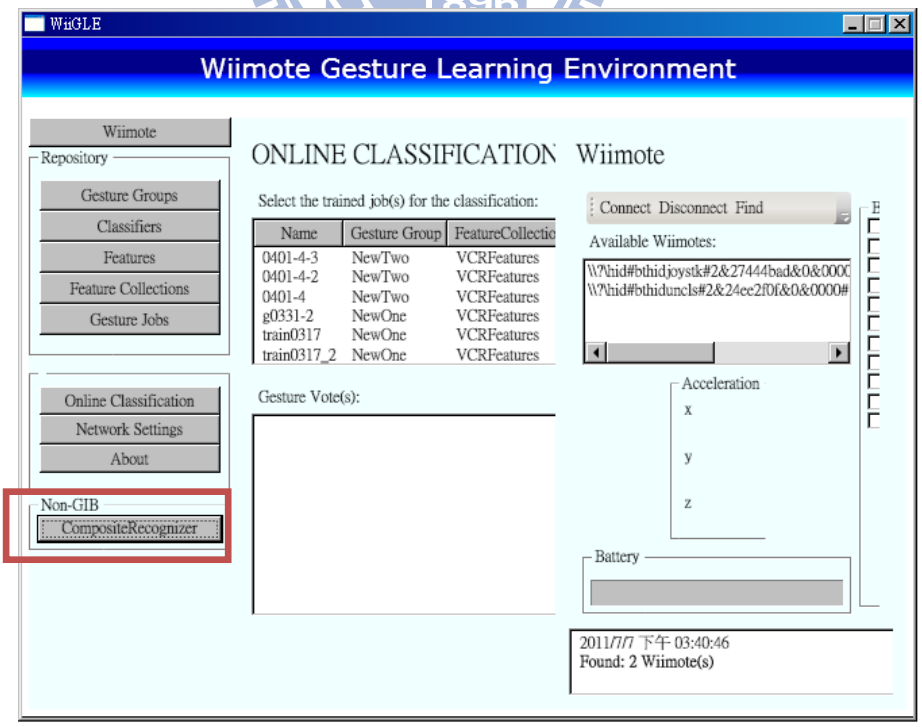


圖 22、WiiGLE Composite Recognizer

Step4.(選擇 Wiimote & 點 Connection)

選擇要辨識器要連接的 Wiimote，出現"Connect"字樣即代表與 Wiimote 的連線成功，之後可以看到由辨識器偵測出來的數據。藉由這個數據辨識器就可以做出相對應的動作。像是滑鼠的移動及點擊的動作。

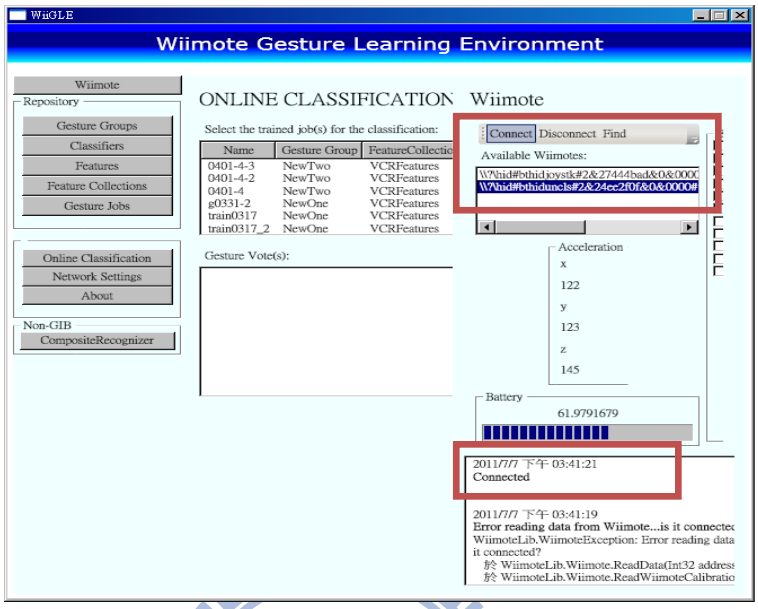


圖 23、Wiimote 連結成功

Step5.(選擇可辨識類別)

然後選擇可辨識的手勢類別，這樣就可以針對不同的可辨識手勢做出識別。

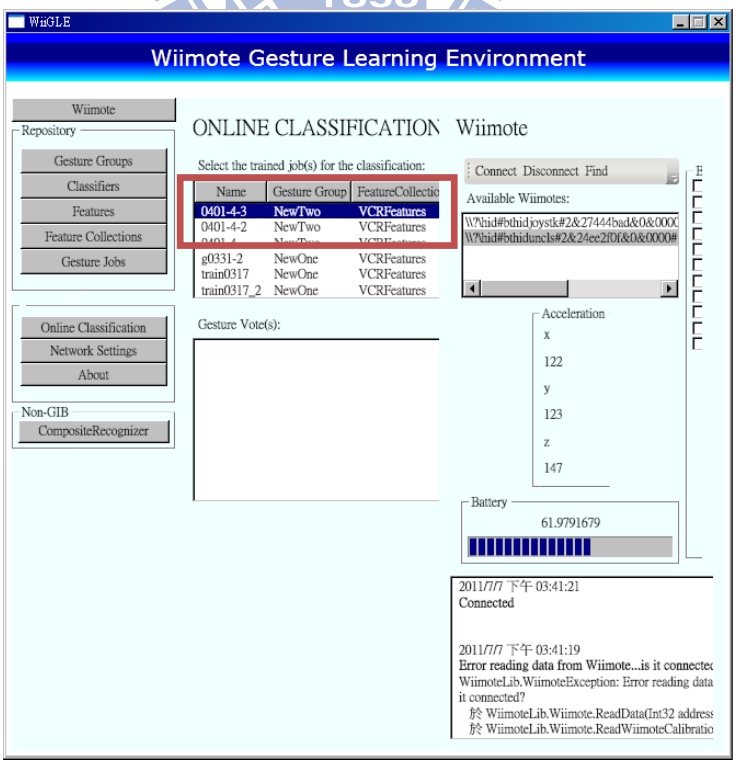


圖 24、選擇可辨識手勢群組

Step6.(點開 Application & 展示)

接下來就可以進行操作。

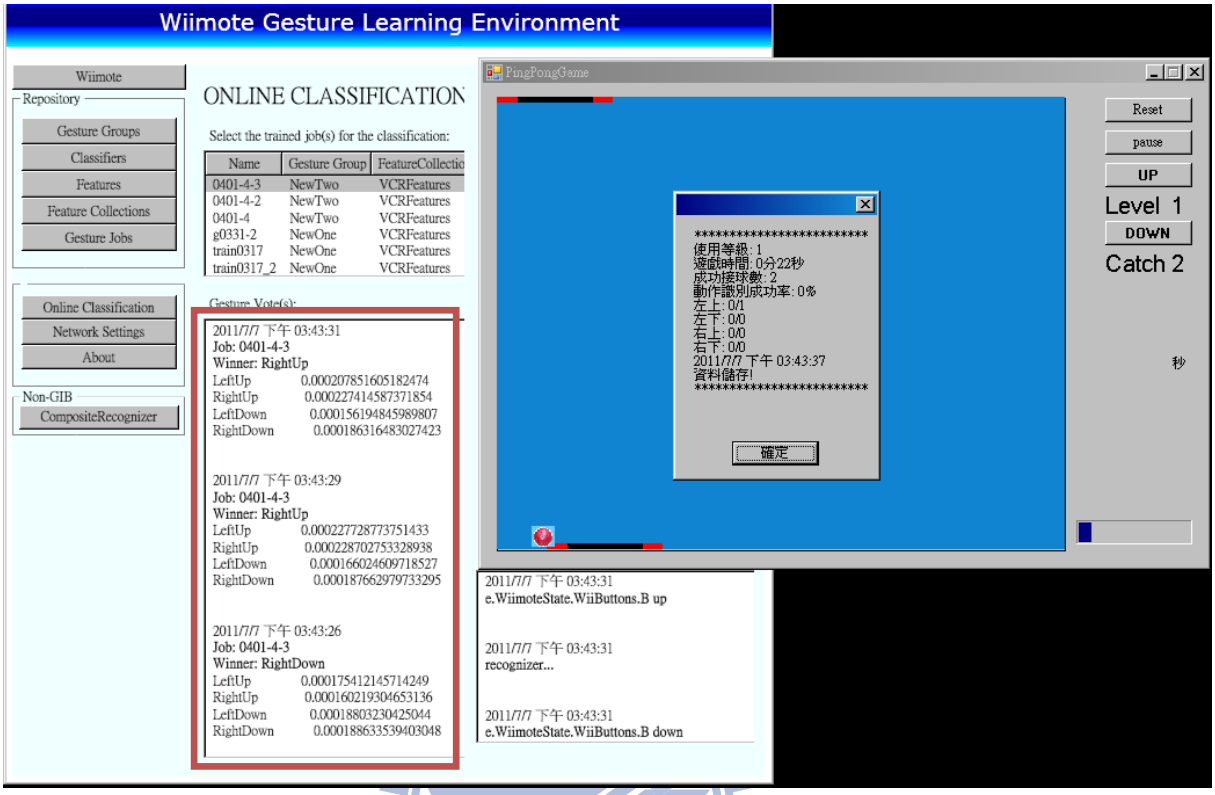


圖 25、辨識器辨識結果與應用軟體畫面展示

5.2 利用程式產生器的方式開發介面程式

Step1. (打開 Command composer)

首先我們藉由 GIB 的 Command composer 進行辨識結果與指令對應的設定。

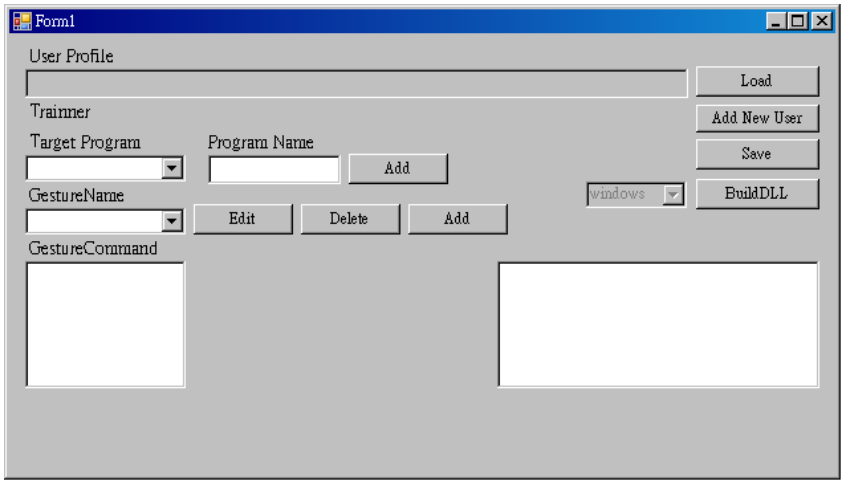


圖 26、Command composer 截圖

Step2. (建新 profile)
建立新的 profile 檔。

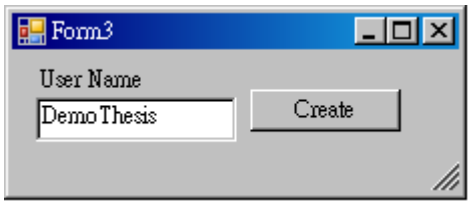


圖 27、Create new profile

當按下 Create 後，Profile 資料夾下就會產生產生器所需要的 specification code 檔案。如圖 28 可見，除了該 profile 的命名，尚無任何資料。

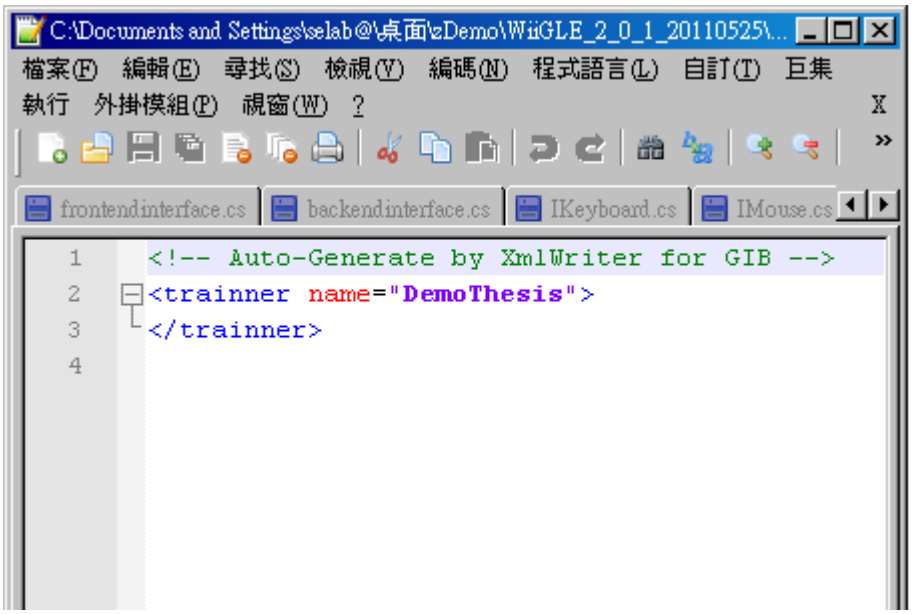


圖 28、設定前 profile 內容

Step3. 選擇目標的 profile 檔案後，點選 Load。便可把該 profile 讀入我們的 Command composer。由圖 29 框起來的部分也可以發現，當 profile 正確讀入時，該 profile 的正式名稱也會被 update 到使用者介面上。

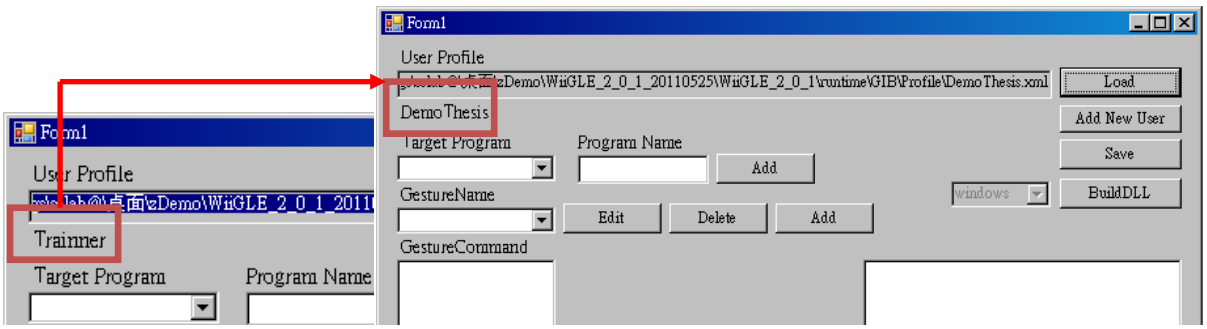


圖 29、Profile 讀入 Command composer

Step4. 首先可以看到，該 profile 完全不含任何可互動的目標程式，所以我們先進行新增的動作，將目標程式的名稱輸入 Program Name 的欄位，同時點 Add。

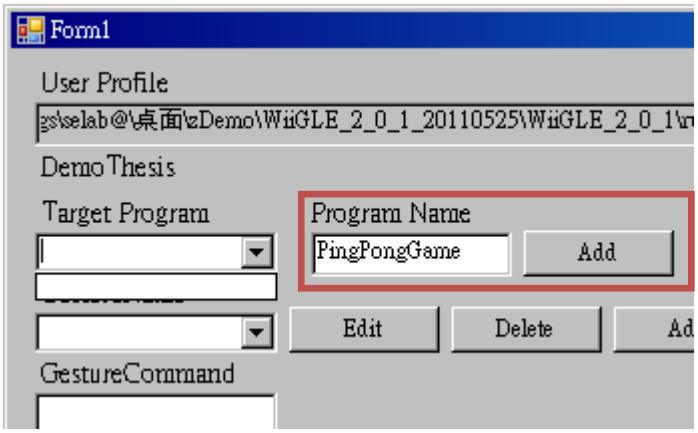


圖 30、新增目標應用軟體

如此一來，該辨識器的動作指令，就只會送到我們希望互動的目標程式。

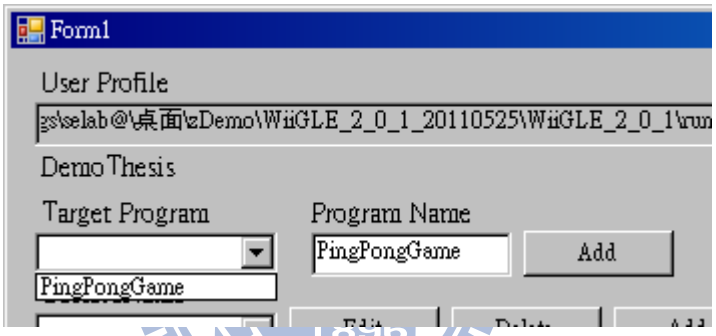


圖 31、目標應用軟體新增成功

Step5. 新增辨識結果對應指令，點選 Gesture Name 的 Add。

圖 32、新增目標應用軟體

之後會出現一個彈跳視窗，包含了一個辨識結果所應該對應到的所有動作設定。像是一個辨識結果有多少個動作組成、是滑鼠還是鍵盤的動作、及指令合成與檢查字彙句法的機制。如圖 33，標示 1 的地方為第一步，產生指令。標示 2 是第二步，進行指令合成與字彙句法檢查。標示 3 是第三步，新增或更新該指令資訊。

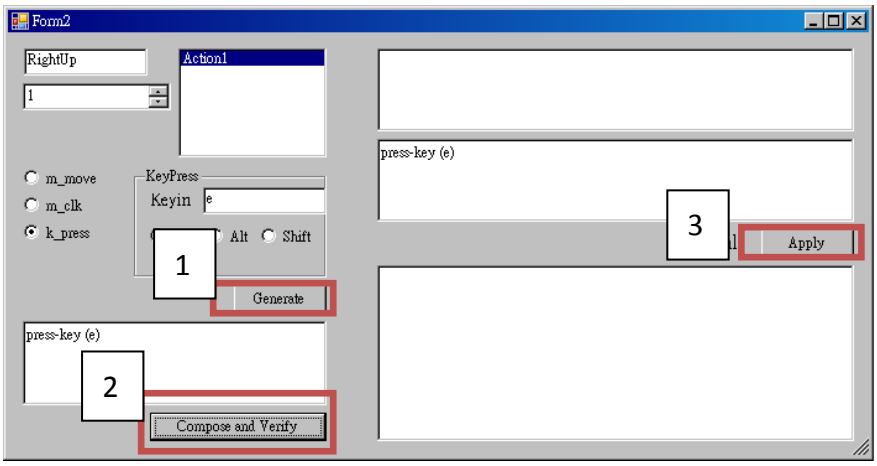


圖 33、新增辨識結果對應指令視窗

新增/更新完一道對應指令的畫面。

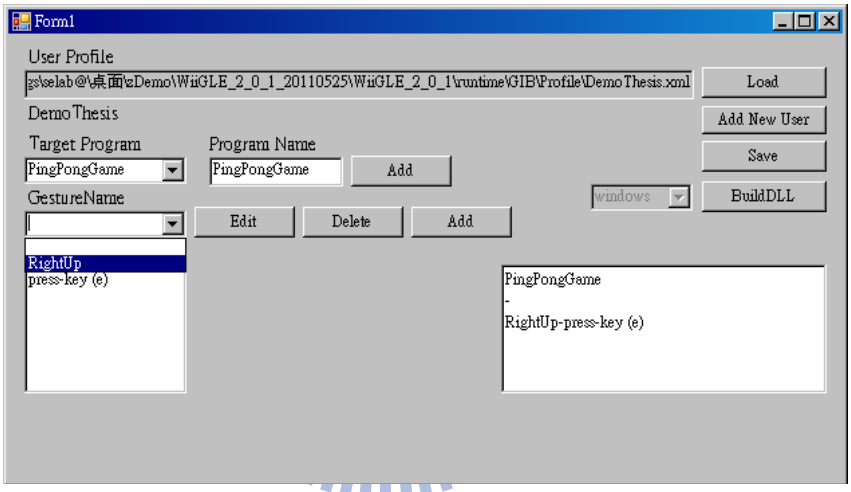


圖 34、新增對應指令成功並回到 Command composer 主畫面

Step6 依序完成所有對應指令的設定。

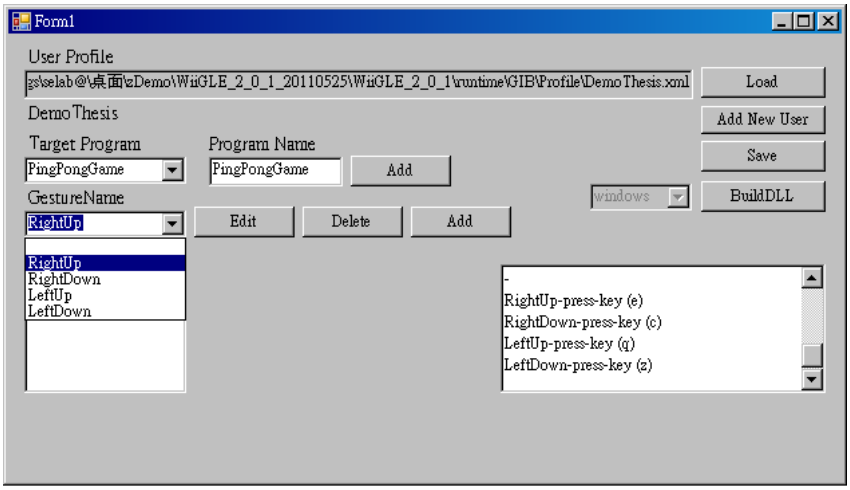


圖 35、所有對應指令設定完成

點選 Save 將所有設定正式存入 profile 中。此時可觀察所產生的 specification code。如圖 36 所示，包含了對應應用軟體名稱，辨識結果名稱，辨識結果應對應的動作指令。

```
1  <!-- Auto-Generate by XmlWriter for GIB -->
2  <trainner name="DemoThesis">
3      <target_program name="PingPongGame" type="
4          <gesture>RightUp</gesture>
5          <command>press-key (e)</command>
6          <gesture>RightDown</gesture>
7          <command>press-key (c)</command>
8          <gesture>LeftUp</gesture>
9          <command>press-key (q)</command>
10         <gesture>LeftDown</gesture>
11         <command>press-key (z)</command>
12     </target_program>
13 </trainner>
```

圖 36、specification code 產出

此時，我們可以進行最後的步驟點選 build，依據我們建立的 profile 產生實際的程式碼(詳見附錄)，同時把該程式碼編譯成 dll 檔。其後將所需要的 dll 檔，搬移到應該放的對應位置，即完成所有前置作業。

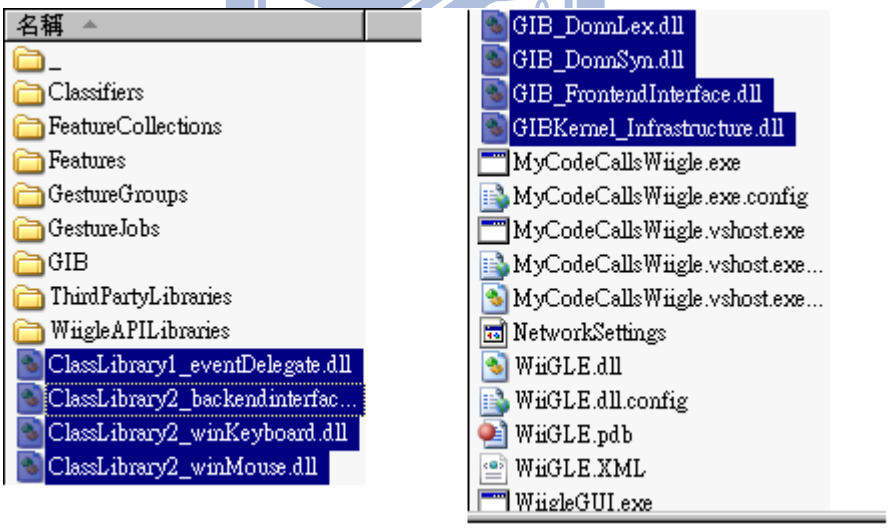


圖 37、Generic Interface Bridge dll 檔 產出

之後的操作步驟，如同 5.1 的步驟 1~6，此處便不再重複。最後我們可發現，利用介面產生器所產生的介面程式，同樣可以銜接辨識器與應用軟體，並達到與直接撰寫介面程式同樣的功能。而且所花費的時間，比起直接撰寫介面程式的方法要少上許多。

六、結論與展望

6.1 結論

首先，在進入結論之前，我們先回到本研究一開始的研究目的部分，本研究的目的是在於尋找兩種介面程式開發方法(直接撰寫介面程式的開發方式、透過程式產生器產生介面程式的開發方式)在開發介面程式時的差異

而由第三章的地方我們發現，兩種開發方式，在開發流程上的差異，像是直接撰寫介面程式需要了解辨識器的原始程式碼、了解如何使用滑鼠鍵盤 API 的部分，透過產生器的方法需要了解如何撰寫 specification code，介面產生器的使用方法，以及事後維護上的困難度也不同。

第四章的地方，藉由實際開發介面程式的時間記錄，我們發現了兩種開發方式，開發所需要的時間成本(這其中包含了實際開發時間以及學習時間)，同時也引用了軟體工程中常用的軟體成本估算模型，COCOMO II 模型，利用程式碼的數量及各種參數的調整，估算軟體開發所需要的時間成本，用於佐證實際開發所統計出來的時間是合理的。

第五章的部分，我們也用了兩種不同開發方式，分別開發的介面程式，實際運用在本研究所選擇的辨識器與應用軟體作為實例。最後，兩者皆可達到同樣的使用目的。

因此我們得到了最後的結論，**在本研究的環境限制下，利用介面產生器產生介面程式的開發方式，比起直接撰寫介面程式的開發方式，是有更高的生產力**。但確切數字，尚須有更大量的數據統計，使實際時間統計能更公平、公正，以及更多準確且可用資訊，如此 COCOMO II 的估算也才能更加準確。

6.2 展望

本研究在此僅討論利用不同的介面程式開發方式後，介面程式開發效率上的比較，並沒有對產生出來的程式碼品質作深入的探討，未來讓可針對此議題作發展。另外本研究的開發環境是基於 Windows 的環境作開發，未來可針對將介面產生器移植在不同的作業平台下，可能會面臨的問題進行討論。最後本研究所使用的辨識器是手勢辨識器，人機介面中尚有許多的辨識器可作討論，例如眼球追蹤辨識器等等…。

參考文獻或資料

1. Tian Seng Leong, Jeffrey Lai, Jeffrey Panza, Peter Pong, Jason Hong, Wii Want to Write: An Accelerometer Based Gesture Recognition System, ICREATE'09, pp. 4-7, Pan Pacific KL International Airport Hotel, Malaysia, November 2009.
2. Thomas Schlömer, Benjamin Poppinga, Niels Henze, Susanne Boll, Gesture recognition with a Wii controller, TEI'08, pp. 11-14, Bonn, Germany, Feb 2008
3. Guirao Aguilar, Julian, Breathing as user interface for pulmonary rehabilitation: respiration tracking using Wii remote controller, University of Tromsø, Master's Thesis, May 2010
4. Christian Schönauer, Thomas Pintaric, Hannes Kaufmann, Full body interaction for serious games in motor rehabilitation, AH'11, Tokyo, Japan, March 2011
5. Fraser Anderson, Michelle Annett, Walter F Bischof, Lean on Wii: physical rehabilitation with virtual reality Wii peripherals, Studies in Health Technology And Informatics, Vol. 154, pp. 229-234, Studies in Health Technology And Informatics 2010
6. Luc Geurts, Vero Vanden Abeele, Jelle Husson, Frederik Windey, Maarten Van Overveldt, Jan-Henk Annema, Stef Desmet, Digital games for physical therapy: fulfilling the need for calibration and adaption, TEI'11, Funchal, Portugal, 2011
7. Avinash Ramchandani, Kevin Carroll, Roel Buenaventura, Jason Douglas, Justin Liu, Wii-habilitation increases participation in therapy, Virtual Rehabilitation'08, pp. 69, Vancouver Convention & Exhibition Centre, Canada, 2008
8. Jilyan Decker, Harmony Li, Dan Losowyj, Vivek Prakash, Wiihabilitation: Rehabilitation of Wrist Flexion and Extension Using a Wiimote-based Game System, Rutgers University, GSET'09
9. Scharwaechter, Leupers, Ascheid, Meyr, Youn, Yunheung Paek, A code-generator generator for multi-output instruction, CODES+ISSS'07, pp. 131-136, Salzburg, Austria, 2007
10. Iris Groher, Stefan Schulze, Generating Aspect Code from UML Models, 4th AOSD Modeling with UML workshop, Boston, USA, 2003
11. Matsumoto T., Matsufuji S., Code generator Implementation on FPGA for an Optical ZCZ Code Using a Sylvester Type Hadamard Matrix, IWSDA'07, pp. 228-232, Chengdu, China, 2007
12. Usman M., Nadeem A., Tai-hoon Kim, UJECTOR: A Tool for Executable Code Generation from UML Models, ASEA'08, pp. 165-170, Hainan Island, China, 2008
13. Bharadwaj J., Chen W.Y., Chuang W., Hoflehner G., Menezes K., Muthukumar K., Pierce J. The Intel IA-64 compiler code generator, Mirco IEEE, Vol20, Issue5, pp. 44-53, 2000
14. Xinming Tan, Yingxu Wang, Cypian F. Ngolah, Design and Implementation of an Automatic

RTPA Code Generator, CCECE'06, pp. 434-437, Ottawa, Canada, 2006
15. Fertalj K., Kalpic D., Mornar V., Source code generator based on a proprietary specification language, HICSS'02, pp. 3696-3704, Hawaii, USA, 2002
16. Fertalj K., Mario Brcic, A Source Code Generator based on UML Specification, INTERNATIONAL JOURNAL OF COMPUTERS AND COMMUNICATIONS, vol2, issue1, pp. 10-19,2008
17. Torsten Grust, Manuel Mayr, Jan Rittinger, Sheriff Sakr, Jens Teubner, A SQL:1999 Code Generator for the Pathfinder XQuery Compiler, ACM SIGMOD'07, pp. 1162-1164, Beijing, China
18. Richards T., Walters E.K., Moss J.E.B., Paimer T., Weems C.C., Towards universal code generator generation, IPDPS'08, pp. 1-8, Miami, USA
19. Vu Nguyen, Barry Boehm, Phongphan Danphitsanuphan, A controlled experiment in assessing and estimating software maintenance tasks, Information and Software Technology, Vol53, issue6, pp. 682-691, 2011
20. Barry W. Boehm, Clark, Horowitz, Brown, Reifer, Chulani, Ray Madachy, Bert Steece, Software Cost Estimation with COCOMO II with Cdrom, Prentice Hall PTR, USA, 2000
21. Jorgensen M., Shepperd M., A systematic Review of software Development Cost Estimation Studies, vol33, issue1, pp. 33-53, 2007
22. Ye Yang, Mei He, Mingshu Li, Qing Wang, Barry Boehm, Phase distribution of software development effort, ESEM'08, pp. 61-69, Kaiserslautern, Germany, 2008
23. Lionel C. Briand, Isabella Wieczorek, Resource Estimation in Software Engineering, ISERN Technical Report, 2001(appear in the second edition of the Encyclopedia of Software Engineering, Wiley, 2001)
24. Matson J.E., Barrett B.E., Mellichamp J.M., Software development cost estimation using function points, Software Engineering, Vol20, issue4, pp. 275-287, 1994
25. Boehm Barry W., Software Engineering Economics, Software Engineering, SE-10, issue1, pp. 4-21, 1984
26. Jan Karel Ruzicka, 連結語音辨識系統及應用軟體系統之介面語言之設計與實作，國立交通大學，碩士論文，民國 94 年
27. 彭士榮，蔣加洛，陳登吉，A Generic and Visual Interfacing Framework for Bridging the Interface between Application Systems and Recognizers, Journal of Information Science and Engineering, vol22, issue5, pp. 1077-1091, 2006
28. Florian Brandner, Dietmar Ebner, Andreas Krall, Compiler generation from structural architecture descriptions, CASES'07, pp. 13-22, Salzburg, Austria, 2007
29. Markus Lorenz, Peter Marwedel, Phase Coupled Code Generation for DSPs Using a Genetic Algorithm, DATE'04, pp. 1270-1275, Paris, France, 2004
30. Boris D. Andreev, Edward L. Titlebaum, Eby G. Friedman, Orthogonal code generator for 3G wireless transceivers, GLSVLSI'03, pp. 229-232, Washington D.C., USA, 2003

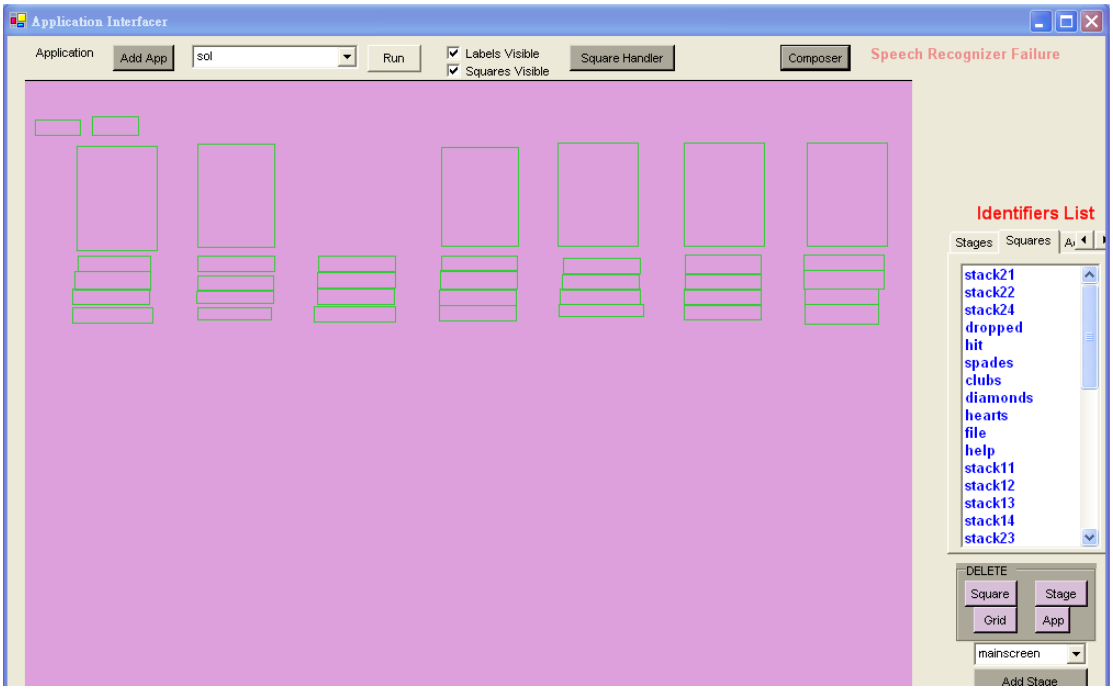
31. Alessandro Bozzon, Sara Comai, Piero Fraternali, Giovanni Toffetti Carughi, Conceptual modeling and code generation for rich internet applications, ICWE'06, Palo Alto, USA, 2006
32. Ben Shneiderman, Catherine Plaisant, Designing the User Interface, 3 edition, Addison Wesley, 1997
33. Ian Sommerville, Software Engineering, 6th edition, Addison Wesley, 2001
34. Wii, http://wii.com/
35. Microsoft Kinect, http://www.xbox.com/zh-TW/kinect
36. G-speak, http://oblong.com/
37. Mgestyk, http://www.mgestyk.com/
38. FingerWorks, http://www.fingerworks.com/
39. HTC sense, http://www.htc.com/tw/htcsense/index.html
40. Simon, http://sourceforge.net/projects/speech2text/
41. Microsoft Speech Recognizer, Microsoft product. http://www.microsoft.com/
42. WiiGLE, http://mm-werkstatt.informatik.uni-augsburg.de/wiigle.html
43. Lee Johnny, Low-Cost Multi-point Interactive Whiteboards Using the Wiimote, http://johnnylee.net/projects/wii/
44. Wiigee, http://www.wiigee.org/
45. Microsoft Visual Studio, Microsoft product, http://www.microsoft.com/
46. Borland C Builder, Embarcadero product, http://www.embarcadero.com/
47. Swig, http://www.swig.org/
48. John R. Levine, Tony Mason, Doug Brwon, YLex & Yacc, 2nd, O'Reilly, 1992

附錄

附錄一 原 GIB 系統限制–畫面展示

限制一圖例

限制二圖例



限制五圖例

附錄二 LL(1) table

	(	)	[	]	move	click	press	then	int	lclick	lDclick	rclick	Keyin
S	-	-	1	-	2	2	2	-	-	-	-	-	-
M	-	-	-	-	3	4	5	-	-	-	-	-	-
V	6	-	-	-	-	-	-	-	-	-	-	-	-
Var	-	-	-	-	-	-	-	-	7	8	-	-	-
mVar	-	-	-	-	-	-	-	-	10	-	-	-	-
cVar	-	-	-	-	-	-	-	-	-	11	12	13	-
pVar	-	-	-	-	-	-	-	-	-	-	-	-	14

Grammar Rule:

S-> [S then M V] | M V

M -> move-to | click-by | press-key

V -> (Var)

Var -> mVar | cVar | pVar

mVar -> int,int

cVar -> lclick | lDclick | rclick | …

pVar -> {keyin}+

附錄三 詢問信件

信件一：

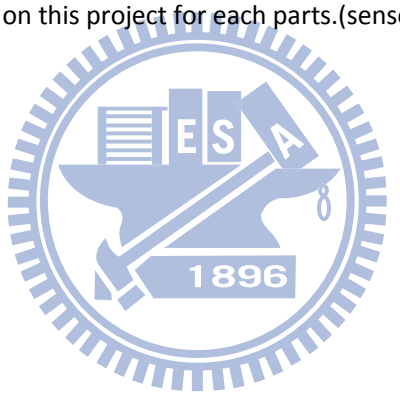
Dear all,

i am Donn, a student of NCTU.

Now i am working on my thesis, it call GIB (generic interface bridge), a kind of interface generator, i hope it to reduce development time if new recognizer publish.
i want to test it with other different recognizer, so i choose gesture recognizer(like wii/kinect...).

i found that you had developed the software with Wii sensor before and use it on rehabilitation.
It's interesting.

may i ask some question?
About how many time you work on this project for each parts.(sensor / interface / software / and others...)



Appreciate for your help.
Best Regards,
Donn

信件二：

Dear all,

I am Donn, a student of NCTU.
Sorry for disturbing again.

Now i am working on my thesis.
And I am interesting on the topic you wrote, Wiihabilitation.
I want to compare the develop time of interface if I do use interface generator or if I do not.

May I ask how much hours or days you spend on develop the interface between Wii and habilitation application?

Appreciate for your help.
Best Regards,
Donn

附錄四 Specification Code

```
<!-- Auto-Generate by XmlWriter for GIB -->
<trainner name="DemoThesis">
    <target_program name="PingPongGame" type="">
        <gesture>RightUp</gesture>
        <command>press-key (e)</command>
        <gesture>RightDown</gesture>
        <command>press-key (c)</command>
        <gesture>LeftUp</gesture>
        <command>press-key (q)</command>
        <gesture>LeftDown</gesture>
        <command>press-key (z)</command>
    </target_program>
</trainner>
```

附錄五 COCOMO II.2000 Scale Factor Value and Calibrated Early Design Model Values[20]

Scale Factors

Scale Factors	Very Low	Low	Nominal	High	Very High	Extra High
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

Calibrated Values

Driver	Extra Low	Very Low	Low	Normal	High	Very High	Extra High
RCPX	0.49	0.60	0.83	1	1.33	1.91	2.72
RUSE			0.95	1	1.07	1.15	1.24
PDIF			0.87	1	1.29	1.81	2.61
PERS	2.12	1.62	1.26	1	0.83	0.63	0.5
PREX	1.59	1.33	1.12	1	0.87	0.74	0.62
FCIL	1.43	1.3	1.1	1	0.87	0.73	0.62
SCED		1.43	1.14	1	1	1	