國立交通大學

資訊科學與工程研究所

博 士 論 文

人機介面及視窗應用程式之通用橋接介面系統

Generic Interface Bridge between HCI and Applications

研 究 生：彭士榮

指導教授：陳登吉教授

中 華 民 國 九 十 八 年 五 月

人機介面及視窗應用程式之通用橋接介面系統

Generic Interface Bridge between HCI and Applications

研 究 生：彭士榮　　Student：Shih-Jung Peng

指導教授：陳登吉教授　　Advisor：Dr. Deng-Jyi Chen

國 立 交 通 大 學

資 訊 學 院

資 訊 科 學 與 工 程 研 究 所

博 士 論 文

A Dissertation Submitted to

Institute of Computer Science and Information Engineering

College of Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy

in

Computer Science and Information Science

May 2009

Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 八 年 五 月

人機介面及視窗應用程式之通用橋接介面系統

研究生：彭士榮　　　　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

摘要

視窗環境下，傳統方法要讓已開發完成的應用程式，具有介面控制能力，以達到操控應用程式之目的，一般常用的方法，是在應用程式設計時，將介面控制系統之控制功能程式，利用函數呼叫以及模組化之方式，直接寫入並包裝在單一的應用程式內。在如此設計開發方式下，存在以下幾個問題：1、使用者必須要對控制介面及程式語言，具備相當豐富的知識背景，才有可能設計出具有人機介面控制功能的應用程式。2、如果要爲現有之應用程式加上人機介面控制功能，設計者必須擁有此應用程式的原始碼，才有可能為此應用程式新增或修改介面控制功能；若沒有應用程式原始碼，我們將很難為應用程式新增或修改介面控制程式。3、即使擁有程式原始碼，設計者亦必須針對應用程式的原理、架構及技術重新加以分析，才有可能撰寫出適合的介面控制程式。如此開發過程顯得非常沒有彈性及效率，且將造成相當大的困難及人力、時間之浪費。基於以上原因，本論文針對軟體工程方法學的應用研究，開發出一個具有通用架構的視覺化介面

平台，做為介面控制裝置與應用程式間的橋樑，使用者只需經由此橋樑，針對應用程式之控制元件做些簡易的描述設定，不需要撰寫任何程式，便可將介面控制裝置與視窗應用程式銜接，輕易地將原先不具有介面控制能力的視窗應用程式，設計成為具有人機介面控制功能的視窗應用程式。

本視覺化通用介面系統研究包括兩大部分，第一：我們提出視覺化通用介面架構觀念，可將開發的視覺化通用橋樑介面，銜接一般的介面控制與視窗應用程式，並且提供使用者簡易之介面操作及指令設定。使用者可以在視窗之任意座標上設定可控制圖格物件（Square Object），並附予每個圖格物件一個名稱。只要在視窗應用程式之相對應位置上設定此圖格物件，然後利用語音將控制命令送至開發之通用橋樑介面(Generic Interface Bridge, GIB)，再經由 GIB 介面解析輸入之指令後，模擬操作滑鼠或鍵盤控制此圖格物件，如此便可達到利用語音來操作視窗應用程式之目的。為了增加操作的彈性及擴充性，我們採用巨集指令（Macro Command）來定義及組合控制指令，使得單一巨集指令可以連續執行數個語音命令，如此可增快指令下達之速度，並可避免因指令過長，造成輸入過程中雜訊進入而影響辨識結果。第二：我們想用 PDA 手機，透過網路操作遠端 PC 或數位電視上的多媒體視窗應用程式。然而想要讓手機具有操作遠端多媒體應用程式功能，需要在手機及多媒體應用程式中，撰寫複雜的程式才可以達到。因此我們採用提出之視覺化通用介面方法，並結合發展之程式碼剖析器

(Parser Generator)，解析特定裝置上的多媒體應用程式，經由通用橋樑介面的操作及應用系統元件之描述設定，程式開發者可以在不需要撰寫手機介面控制程式下，自動且快速地在手機上產生具有控制多媒體應用程式功能的介面環境，達到以手機遙控多媒體應用程式的目的。

為了展現提出方法的適用性及可行性，在論文展示範例中，我們將原先不具有語音或遙控功能之視窗應用程式，在不撰寫任何程式語言之情形下，用簡單、快速、有效率之方法與我們研發的通用橋樑介面結合，經由語音或 PDA 手機輸入控制命令，再由橋接介面系統解析，最後達到輕易控制視窗應用程式之目的。

Generic Interface Bridge between HCI and Applications

Student: Shih-Jung Peng   Advisor: Prof. Deng-Jyi Chen

Institute of Computer Science and Information Engineering
National Chiao-Tung University

Abstract

In a Windows environment, the commonly used traditional method, which allows developed window application programs with Human Computer Interaction (HCI) control ability, is directly to write the control procedures into the application programs while using low-order designing formula to package procedures into single application system. To apply such devising method, the designer must possess certain knowledge about application system designing and programming in order to devise an application system with HCI control functions. Particularly when the design is completed, it is relatively difficult to revise or add any system functions to it without the primitive code.

Three major problems may exist under such development of HCI control procedures. First, system designers must be equipped with abundant knowledge about the design of HCI and programming languages in order to design an application with HCI function. Second, if we want to design an application, which lacked interaction ability before, we need to obtain the primitive code of the particular application due to the difficulty in modifying new programs without the code. Third, even if we have obtained the code, we need to re-analyze the entire structure of application in order to write a suitable control program. These tasks will leave the designer with much trouble and seemingly resulting in less flexibility and efficiency.

In this thesis we will emphasize on the research of Software Engineer Methodology to develop a visual generic interface bridge (GIB) system. Under this GIB system, designers devise the application system with speech or remote HCI control functions in a much easier and efficient manner with the need for defining only some parameters of objects in an application environment whereas the user are not required to write any program code.

The research on GIB system consists of two parts: “Integration of GIB and HCI,” and “GIB-based Application Interface (GAI) generation”. Under the GIB system, we propose the concept of GIB system in connection with HCI and applications. The GIB provides a visual operating interface in which designers draw controllable square objects at any corresponding position on the windows and name each square object. Subsequently, we use speech function to operate actions for mouse and keyboards corresponding to the square object; implying that we can easily control the application with speech as well. In order to increase the operation of applications with more flexibility and expandability, we may use macro command to define and combine the control commands. One macro command may combine with several control commands; this implies that noise effects between long commands can be avoided, making the application control more flexible. Under the GAI system, we use PDA cellular phone to control application programs on the PC in a convenient manner. Nonetheless this process is not simple due to many complicated procedures in and between these systems must be written. For this reason, we use proposed concept of GIB system and adding program parser method, under which the PDA can easily connect with HCI and multimedia applications to achieve the goal of controlling via simple interface operating and setting of AP’s environment.

In regards to demonstration of the feasibility and suitability of the proposed method, we put in practice of developing a GIB system to be used as the bridge between HCI interface and window applications. In the following examples, we literally manipulate the connection of a generic visual interface with application program in a very simple, fast, and effective manner. Speech or remote control HCI ability is implemented more easily and efficiently throughout this GIB system without the need for writing any program code.

Acknowledgement

Foremost, I would like to sincerely give thanks to my advisor, Professor Deng-Jyi Chen, who not only provided me with his creative suggestions and discussions over my studies but also his immense assistance, support, patient and inspiration.

Particularly thanks to Chien-Chao Tseng (Prof. of NCTU) for his tremendous assistance and support. Especially thanks to Shih-Kun Huang (associate Prof. of NCTU), Yeh-Ching Chung (Prof. of NTHU), Koun-Tem Sun (Prof. of NUTN), Wu-Yuin Hwang (associate Prof. of NCU), and Chorng-Shiuh Koong (assistant Prof. of NTHU), who are the members of the oral examination committee for my doctoral dissertation. They have offered me many valuable suggestions to help me with expand a broader scope of research and more important applications than I had initially started with.

I would like to thank gratefully to my friends Chin-Eng Ong and Jan Karel Ruzicka, masters of Science in Computer Science and Information Engineering, for their cooperation and assistance on this dissertation which eventually transformed into applicability and completeness. Thanks to all of my friends who have helped me in my life with researching and studying.

I have many thanks to my wife, Pei-Ling Chen, and my children, Yi-Ping and Yi-Han, parents and parents-in-low for their love, patience, support and always standing by me in my life at all time. Finally, I would like to dedicate this dissertation to my family.

Table of Contents

List of Figures

List of Tables

Introduction

1.1 Introduction

In a typical window-base interface system, users control application programs containing the HCI (Human Computer Interaction) control functions in speech or handheld device such as PDA or cellular phone directly and conveniently. The commonly used traditional method which allows developed window application programs with speech or wireless remote HCI control ability to follow the control commands for achieving the goal of operating application programs, will directly integrate the HCI control function and API procedures of speech or handheld device into the application programs by using low-order designing formula to package procedures into single application program through a direct and intense method. Under such devising method, the designers must be equipped with certain knowledge about application system designing and programming in order to devise an application system with speech or remote HCI control functions. Even if the design is completed, it is relatively difficult to revise or add the interface functions without the primitive code, as shown in figure 1-1.

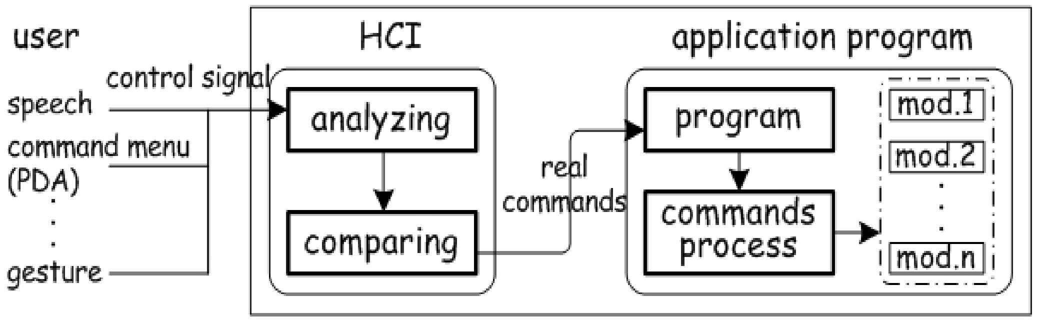
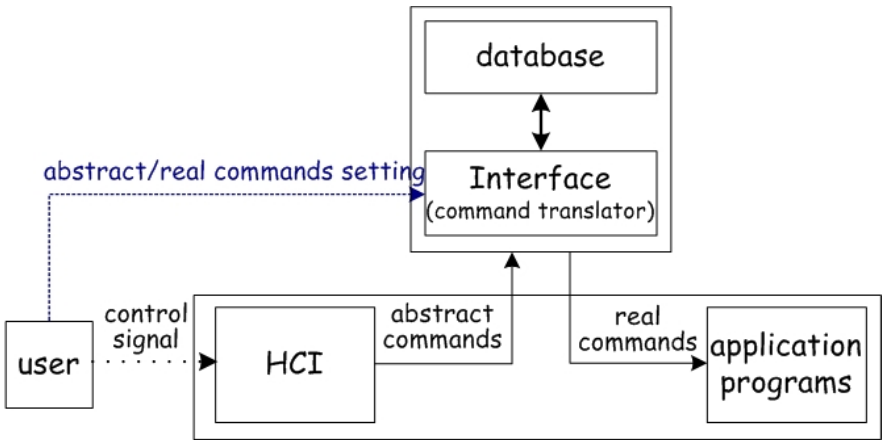


Figure 1-1 General developing method of speech control

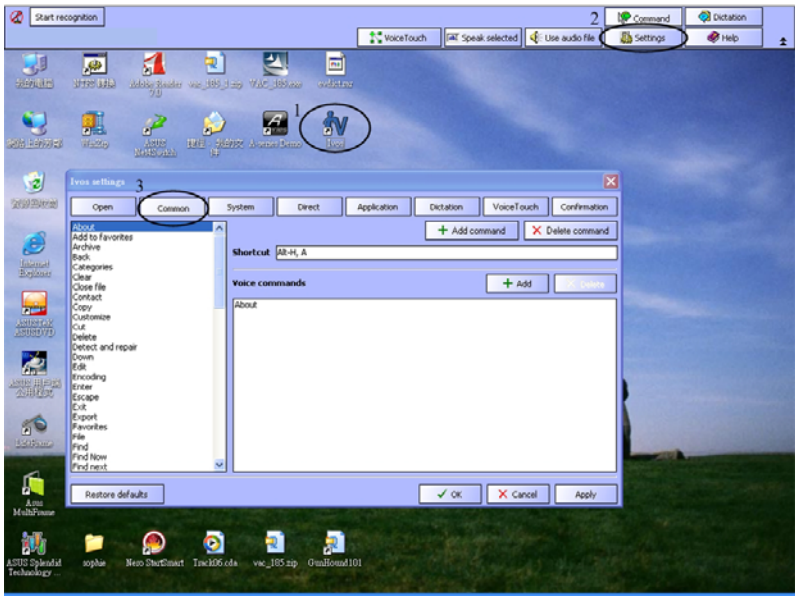
Three major problems exist in such development of speech or remote HCI control procedures: first, system designer are required to possess abundant knowledge about system design for control interface and programming language in order to design an application with speech or remote HCI control functions. Second, if we want to obtain an application program with speech or remote HCI control functions lacking previous interface control ability, we must obtain the primitive code for that application program due to the difficulty in designing without the code. Third, even if the code is attainable, we are still required to reanalyze the entire structure of application in order to write a suitable interface control program. These tasks will leave the designer with much difficulty and seemingly resulting in less flexibility and efficiency.

To overcome these issues, in this thesis we emphasize on the research of Software Engineer Methodology to develop a visual generic interface bridge (GIB) system and introducing this system into two parts: First, the "Integration of GIB and Speech HCI," and secondly, "GIB-based Application Interface (GAI) generation," in which a wireless handheld device is taken as an example. Under the GIB system, designers devise the application system with speech or wireless remote HCI control functions in an easier and more efficient manner with the only requirement for defining some parameters of square objects in a application environment where users do not need to write any program code.

In part one, most of voice control robots employ such development method in designing their voice control interface, such is in the case of AT&T's Speech-Actuated Manipulator (SAM) [1], whereas the voice commands via telephone are comprehended and then performed with respective actions. Under such means of intense design, we discover that if there is any need for modification in speech control function, the primitive source code of application system must be acquired and comprehended before using the low level design method to write application control functions, while the entire program must be recompiled and executed. Such design manner does not offer extreme efficiency, moreover it is hard to design, increase or revise the control function without the original source code of the application.



(a) IVOS system architecture



About
Add to favorites
Archive
Back
Categories
Clear
Close file
Contact
Copy
Customize
Cut
Delete
Detect and repair
Down
Edit
Encoding
Enter
Escape
Exit
Export
Favorites
File
Find
Find Now
Find next

Shortcut: Alt-H, A

Voice commands: About

(b) IVOS system

Figure 1-2 OS integration approach

An OS integration approach allowing one speech recognizer to be intact with the domain of applications such as Vspeech 4.0 [2], Voxx 5.0 [3] and IVOS [4], utilizes the appearance of windows interface, designs some fixed pronunciation operation functions and establishes the name of relative operation instructions. Different instruction names are used to control the same operating function such as "open file" to <Alt>-F and "file open" to <Alt>-F, and then using the designed file name of speech command to achieve the goal of controlling the application system with speech commands, as shown in figure 1-2.

However such development method of speech control system contains two problems: first, the current approach leaves no flexibility for future modifications about control functions. The control functions were already designed steadily in the HCI system and unable to be increase or revise. If some functions we use to control application were not defined, we must get the primitive source code of the application and redesign it. This will increase difficulty and loading on the work. Second, the current approach may only control application functions with hot-key commands. Due to the system only defines the control functions of keyboard, therefore only keyboard actions can be controlled. Nonetheless there are still many applications with control functions of mouse actions, the current approach will not be able to fully handle theses applications.

Based on the reasons states above, we develop a visual generic interface bridge (GIB) system between speech HCI and window-base application programs. The GIB provides visual operating interface, under which designers draw recognizing square object at any corresponding position on the windows and name each square object. Subsequently, we can easily use speech command to control mouse and keyboard actions corresponding to the position of square object. By increasing the operation of application with more flexibility and expandability, we use macro command to define and combine the control commands. One macro command may be combined with several control commands; this will avoid noise effect between long commands and make the application control more flexible with grammar analysis technology.

Through this process we can make any application program which did not have speech HCI control ability previously, and to control easily with speech commands simply after defining simple square object and sets of macro commands without the need for writing any program code, as shown in figure 1-3.

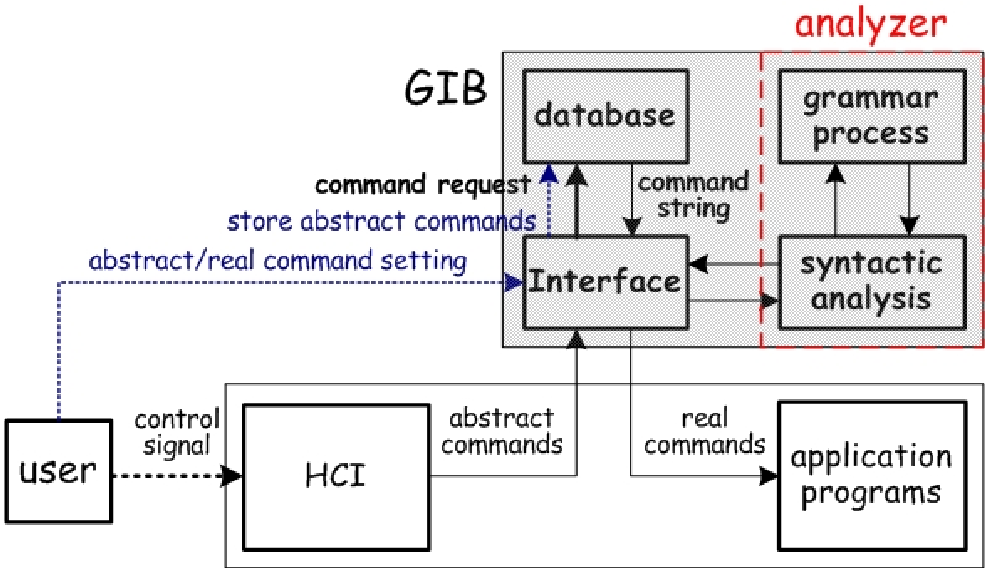


Figure 1-3 Architecture of GIB control system

In part two, due to remote control methods have been discussed in recent years; in 2001, it was proposed that cellular phone to be used in controlling remote HCI running on the PC. Cellular phone is a mobile device containing a small screen without showing the same GUI (Graphical User Interface) as on the PC screen. Therefore we must analyze the GUI and develop some function programs for the cellular phone based on its use in the remote control interaction with GUI on the PC. However along with progressing existing techniques and increasing functions of communication equipments for family use, operation and interaction styles between people and devices have increased in complexity. Most of the multimedia contents can be run and displayed on different kinds of platforms without containing remote control ability originally. Consequently people may believe that if they can simply use some simple instruments, such as cellular phone or PDA, to remotely control the multimedia application module running on the PC or digital TV, subsequently the control will become more vivid and interesting. However due to the variation in control instruments, display devices, and different kinds of methods, it is not easy to make a specified device with remote control ability. To achieve this, we must repeat the design process: (1) Write control command protocols and HTTP wireless protocols into the applications running on the PC; (2)

Write interface program and wireless control protocols into cellular phone, as shown in figure 1-4, for every application system in successive sequence. Despite this designing process is a difficult work in nature; we must write many complex procedures even for simply adding or modifying a new control functions. In addition, there is still one major problem remaining if we lack the original source code of application system, as it becomes impossible to make the specified device with remote control ability. Such repeated design procedures would increase developing application systems a financial burden, waste of time, inflexibility and inefficiency.

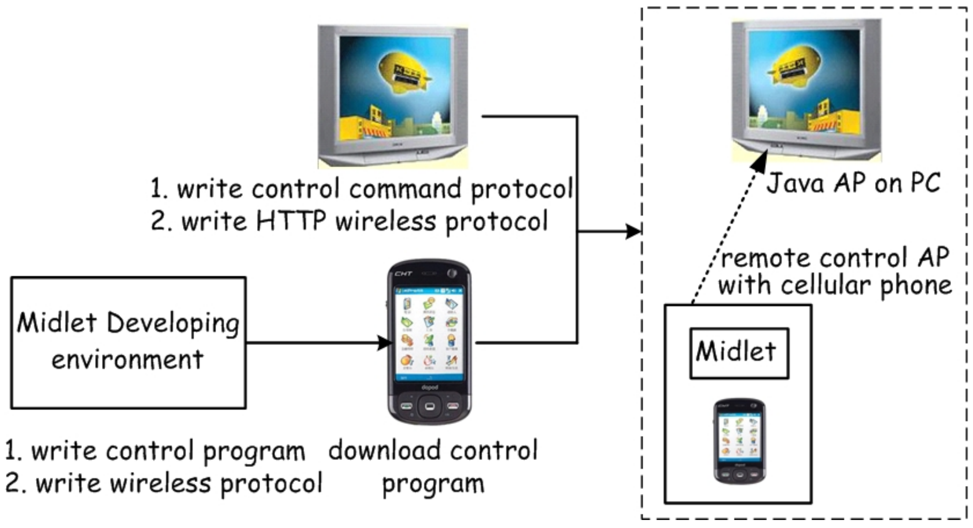

1. write control command protocol
2. write HTTP wireless protocol
Java AP on PC
remote control AP with cellular phone
Midlet Developing environment
Midlet
1. write control program
2. write wireless protocol
download control program

Figure 1-4 General developing method of handheld device control

As the problems reveal, Rajicon System [31] was using cellular phone to control the remote PC, as shown in figure 1-5. It works on a specific device, so, if we want to perform some complex operation functions, we must define a set of interactive rules. Rajicon System uses macro commands to formulate every operating function, therefore, the more operating functions there are, the more likely that more macro commands would be required, as described in table 1.

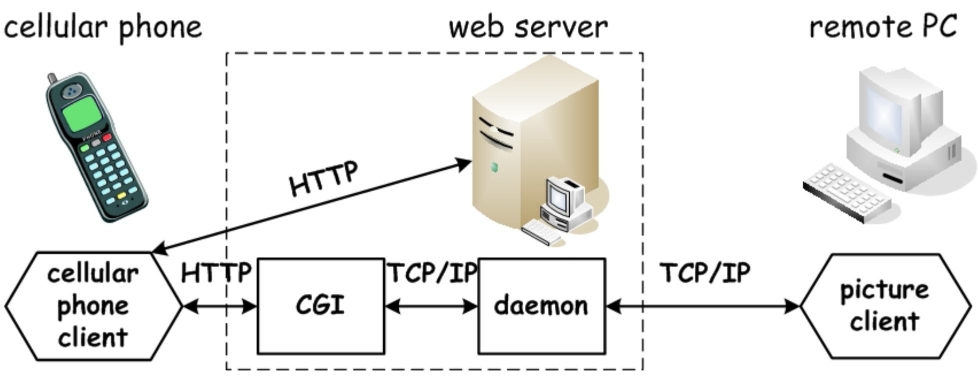


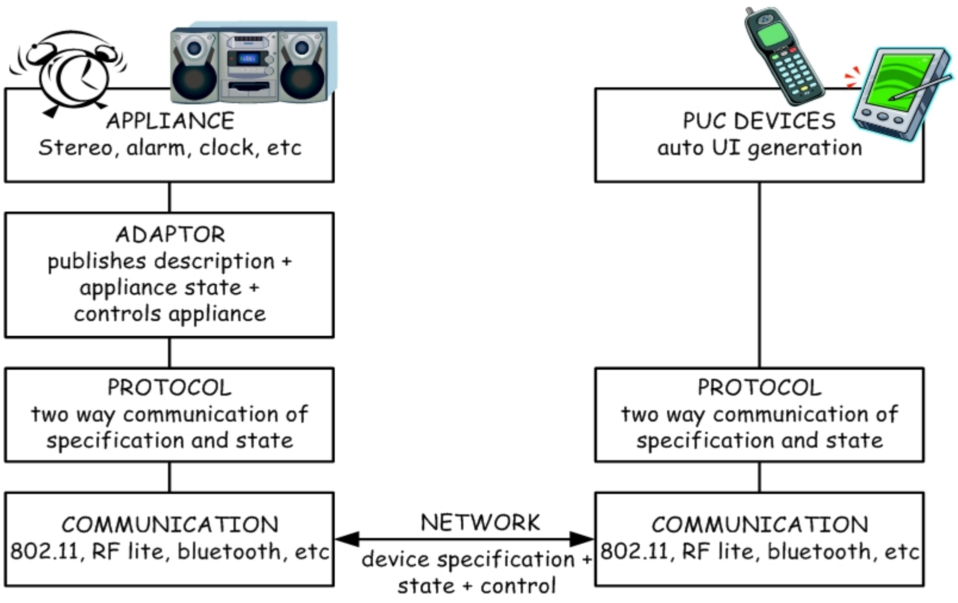
Figure 1-5 Rajicon System Architecture

Table 1-1 remote control functions with cellular phone keypad mapping

Key	Zoom mode	command	result
1	View entire desktop	?a	Hold down Alt key until next key pressed
2	Dec cursor height	?c	Hold down Ctrl key until next key pressed
3	Refresh	?n	Pressed Enter key
4	Dec cursor width	?m	Press left mouse button
5	Execute macro	?^	Maximize the window at the cursor

Through pressing keypad of cellular phone to input command for controlling application running on the PC, these operation methods get complex and disorient, in addition to the difficulties in memorizing these control commands. On the other hand, this system is designed for a specific device when there are some control functions that need to be created or modified; such designed procedures should be restarted repeatedly.

In 2002, Jeffery Nichols and Brad A. Myers published a paper on "Generating remote control interfaces for complex appliances" in [32]. They proposed a method of personal universal control (PUC) with a content describing how to create a control interface with graphics or speech by downloading a functional specification explanation of equipment for family use, as shown in figure 1-6.



APPLIANCE
Stereo, alarm, clock, etc
ADAPTOR
publishes description + appliance state + controls appliance
PROTOCOL
two way communication of specification and state
COMMUNICATION
802.11, RF lite, bluetooth, etc
NETWORK
device specification + state + control
PUC DEVICES
auto UI generation
PROTOCOL
two way communication of specification and state
COMMUNICATION
802.11, RF lite, bluetooth, etc

Figure 1-6 PUC System Architecture

Hence the PUC system analyzes specification document and uses decision tree algorithm to create a specification group tree, as shown in figure 1-7. Then, establish an appropriate control interface according to the structure of this group tree. However there are many design factors which need to be taken into account, including: (1) Download a functional specification explanation for a specific equipment before establishing an interface, however it is not an easy task to establish this specification description; (2) Design a control interface is weary because interface designed algorithm needs to take into account users' requirement with the presentation of control device objects.

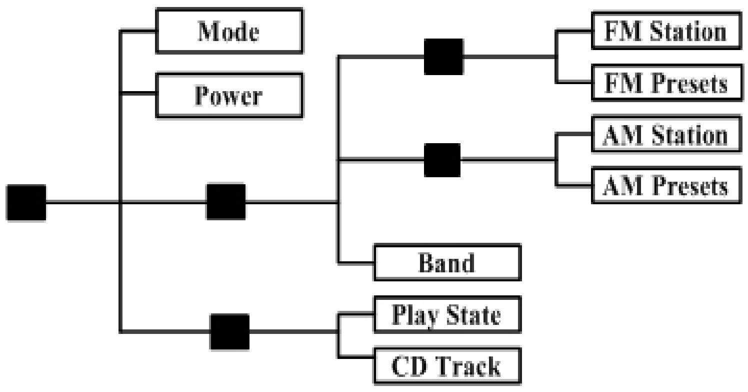


Figure 1-7 Sample group decision tree for a shelf stereo

In [36], because traditional remote control typically allows users to activate the functionality of a single device, consequently qualitative and quantitative results from a study of two promising approaches creating such a remote control are presented: end-user programming and machine learning. In end-user programming, users manually assign the buttons which they believe to be sufficient to accomplish their tasks of graphical “screens”. They then work with a single, handheld remote control that can display those screens. Machine learning also uses a single, handheld remote to display screens; however, the learning uses the recorded history of a user’s actual remote interactions, to infer appropriate groups of buttons for the performed tasks. Some questions arise as described in the following: (1) the end-user programming is too complicated to carry on the design of remote control to the device with graphical screens. User must define the components required for operation on a screen which can control many device interfaces remotely concurrently, and all the control procedures require writing a large number of programs; (2) The ML can record the operation of users automatically and utilizes the performing algorithm to produce the operating component. Perhaps due to the difference in device, it cannot completely define each function of device component; (3) The methods of automatically producing remote control interface are required to follow to the characteristic of the device and the preference of user, but it also needs a large number of procedures to be written in order to design algorithm automatically.

In [38], a new widget and interaction technique known as a “Frisbee,” was described for interacting with areas of a large display that is difficult or impossible to be accessed directly. It consists of a local “telescope” and a remote “target”. This design meets five design principles in: (1) minimizing physical travel, (2) supporting multiple concurrent users, (3) minimizing visual disruption while working, (4) maintaining visual persistence of space, and (5) application independence. However, the design requires writing many procedures for the specific and large-scale showing device. Moreover, because of the limitation of hardware specification and the difficulty in obtaining relevant information, it is comparatively difficult and complicated to design the interface.

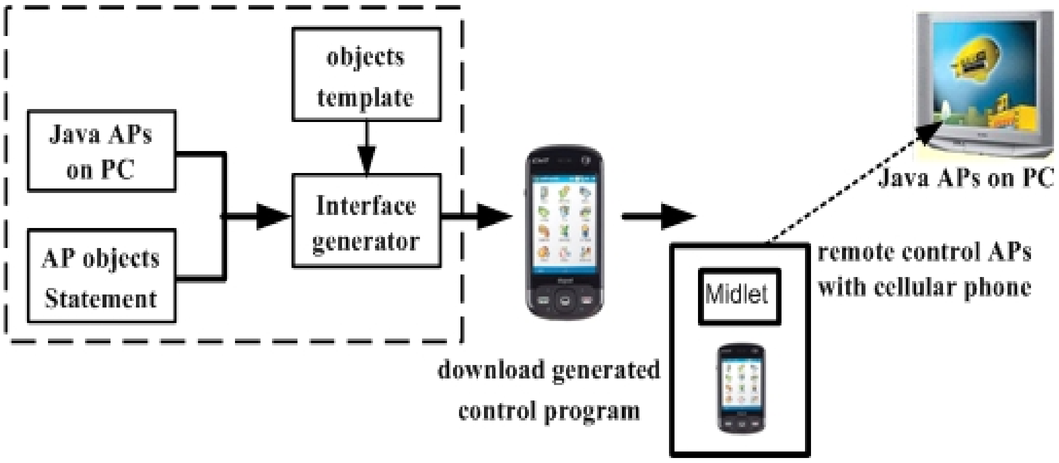


Figure 1-8 Architecture of proposed handheld device control system

Based on the issues mentioned above, our proposal for solving this issue will be to construct an interface generating system; with it the designer can easily develop remote control interface program into cellular phone without even having to write the program of cellular phones. We comply with the concepts of GIB to develop a bridge interface for remote signal control system, under which designer can easily transform from specified control object of Java application system running on PC, such as control buttons and labels, and generating remote control interface with objects, into cellular phone automatically. This will simplify the development process of creating a control interface and makes the control system development and modification more flexible and elastic. Through directly generating controlling objects and functions onto the cellular phone by interface generator, we may directly and easily control the Java application system running on the PC without having to write complex programs into the cellular phone, as shown in figure 1-8.

This thesis emphasizes on the research of Software Engineering Methodology in order to unfold the feasibility and the serviceability. We literally develop a GIB system by adopting a proposed method which is used as the bridge between HCI and window applications, and through which the interaction of HCI and applications will become more vivid and friendly without any requirement of writing any program code from the user.

1.2 Motivation and Goals

In general, if we want to modify HCI (Human Computer Interaction) control functions or increase some new control functions in the application controlled by speech or handheld device, we are required to obtain the original source code of the application system or simply redesign the system. Despite that we might obtain the primitive source code of application system, we might also need to spend much time on analyzing the entire system accordingly; and then use the low level designed way to write and recompile the whole application system; such design method is very difficult and inefficient. The objective of this thesis is to develop a visual GIB platform as a connection bridge between HCI and applications. We can use this GIB system as an interaction interface, to operate applications without previous recognition functions by speech command or PDA device, through simple definition of square objects, parameters description and setting of macro commands without the need of writing any program code.

1.3 Organization of this dissertation

We have briefly introduced the problems of current approach regarding control application with speech or PDA HCI, and have proposed GIB platform to overcome these issues in chapter one. The rest of this dissertation is organized as follows: in chapter 2, introduction of related works regarding proposed GIB system. This discussion contains a touch-panel interface, speech recognition engine, script language, compile grammar definition, Macro command and OS’s API. In chapter 3, we propose integration of GIB and speech HCI system architecture which includes three parts: input module, kernel module and output module. In this chapter, the operations on how to design this bridge interface, and how to define and set control environment of application in order to provide a successful interaction interface between HCI and applications without writing any program code, will be described in details. In chapter 4, applying the concept of proposed GIB system in chapter 3, we propose GIB-based Application Interface (GAI) generation to generate control interface automatically on the PDA through simple object descriptions, in order to control the applications on the PC Windows. In chapter 5, we show real demonstration to explain how to operate applications through speech command or PDA without the need for writing any program under proposed interface generation. Finally, in the last chapter a conclusion is drawn and future work will be described.

2. Related Work

To develop Generic interface bridge (GIB) system, we need some support of technologies in order to allow the system to work in a more generic, flexible, simple, and effective manner. The corresponding technologies include: touch panel interface, speech recognition engine, script language, compiler grammar, macro command, and OS's API. All these technologies would be described briefly in the following sections.

2.1 Touch panel interface

Our visual HCI interface system utilizes the concept of touch panel, as shown in figure 2-1; develops a general transparent sheet of glass between application and mouse cursor, then executes application under this developed circumstance by control mouse cursor actions through controlling drawn and defined square objects above control application on the screen with a name given to each object. That means, through the concept of see-through interface [6-8], if we can use devices such as speech or PDA to control the square objects drawn and defined from the corresponding position of application program on the Windows, we can control this application as well.

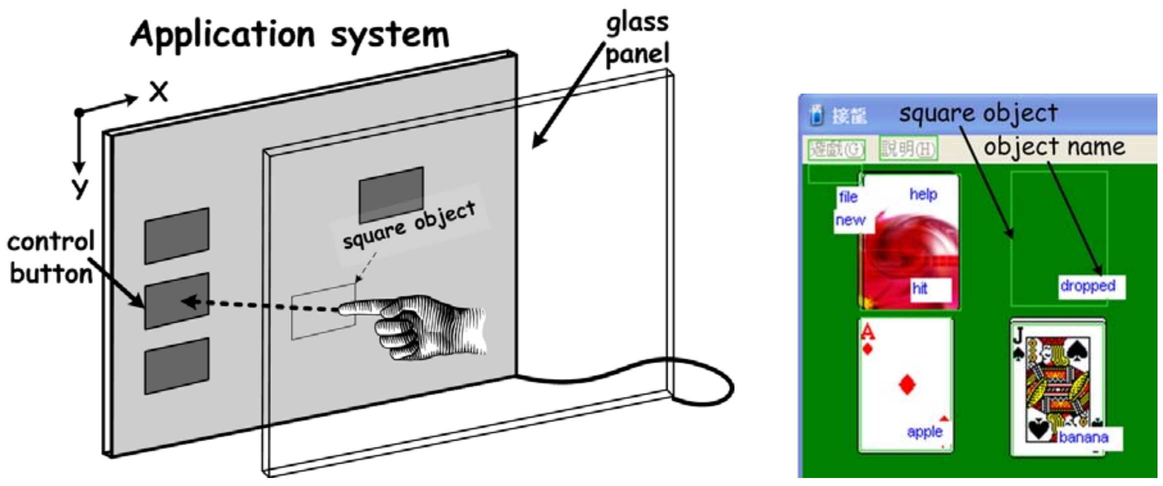


Figure 2-1 Touch panel interface

Applying concepts of touch panel may yield many widgets in visual interface without requiring new offers of extra screen space and allows the control steps become easier and friendlier to follow. An example in figure 2-2 [6] shows that many simple widgets called click-through buttons were used to change the color of objects below them. The user positions the nearby widget and appoints which object for coloring by clicking through buttons with the cursor over that object, as shown in figure 2-2(b). Furthermore buttons in figure 2-2(c) can change the outline colors of objects.

That means, we can use speech or remote control device to control application through controlling mouse actions including mouse moving, mouse events, click, double click, drag, drop, and keyboard press actions by using touch panel concept

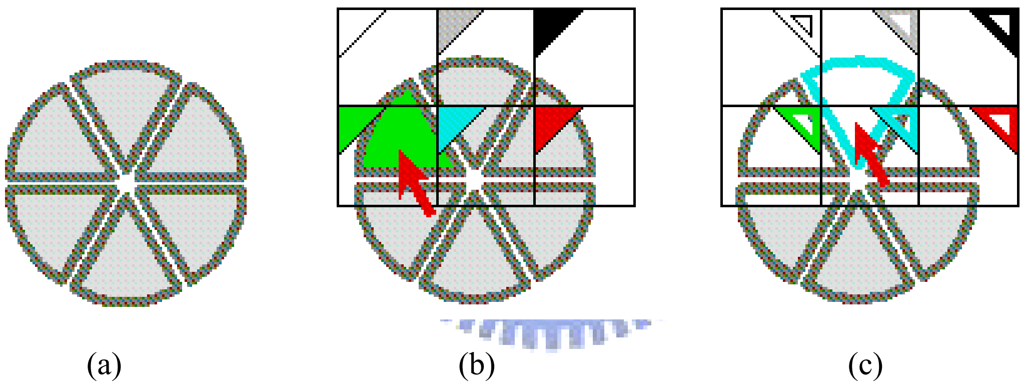
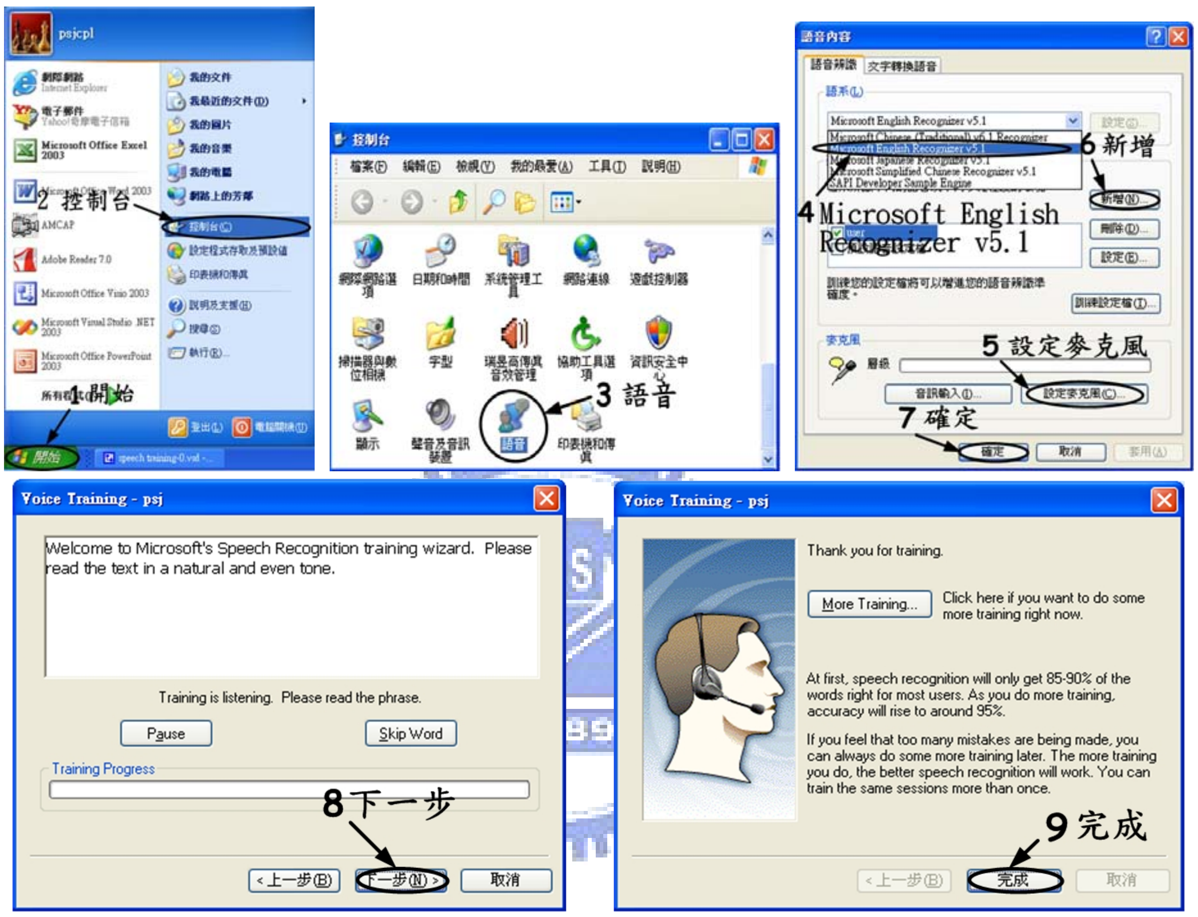


Figure 2-2 Click-through buttons. (a) Six widget objects. (b) Clicking through a green fill-color button. (c) Clicking through a blue outline-color button.

2.2 Speech recognition engine

The pronunciation of every person may differ. Each person must do speech recognition training before using recognizer as a controller to operate the application, resulting in a more correct recognition. In this study, we chose Microsoft Speech Recognizer v5.1 as our system recognition engine and the figures to complete the training steps of speech recognition as shown in figure 2-3. After speech command was inputted and processed by speech recognition engine [16-17], if command was correct, the system would recognize the commands and transfer the recognition result to the system function RecoContext_Recognition( ). The parameters of this function are as below.



Welcome to Microsoft's Speech Recognition training wizard. Please read the text in a natural and even tone.

Training is listening. Please read the phrase.

Thank you for training.

Click here if you want to do some more training right now.

At first, speech recognition will only get 85-90% of the words right for most users. As you do more training, accuracy will rise to around 95%.

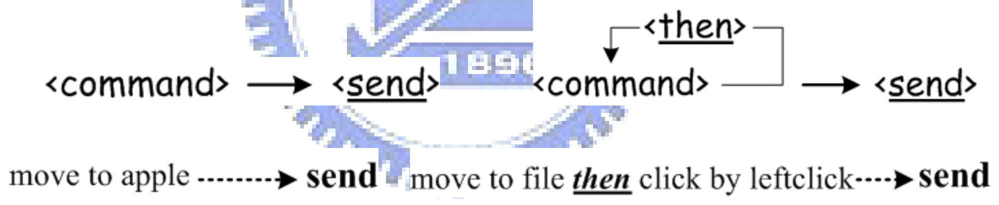
If you feel that too many mistakes are being made, you can always do some more training later. The more training you do, the better speech recognition will work. You can train the same sessions more than once.

Figure 2-3 Recognition training steps of MS speech recognizer V5.1

```
Dim WithEvents RecoContext As SpeechLib.SpSharedRecoContext
Public Sub RecoContext_Recognition (
    ByVal StreamNumber As Integer, _
    ByVal StreamPosition As Object, _
    ByVal RecognitionType As SpeechLib.SpeechRecognitionType, _
    ByVal Result As SpeechLib.ISpeechRecoResult ) Handles RecoContext.Recognition)
```

2.3 Script language

For a more convenient and efficient control with speech commands, we apply script language [11-15] as commands format. Script language is different from system programming language in which it could achieve a higher level programming and a more rapid development than the system programming language. There are some advantages in using script language: 1. it is always stored as a plain text file which users could easily edit or read with any text file editor such as "Word" or "Notepad" application. 2. It is an excellent tool for developing application rapidly. User could edit a simple script file to solve a simple problem with less code and time. Using script language as command format might enable the development and combination of control command string to be easier, rapid and flexible. The speech command input format is as shown in figure 2-4, single command example *move to apple → **send*** for figure 2-4-a, compound command example *move to file **then** click by leftclick → **send*** for figure 2-4-b, and script language syntax tree of "dragsquare" is as shown in figure 2-5.



<command> → <send>
move to apple → send
<command> <then> → <send>
move to file then click by leftclick → send

a. single command input b. compound commands input

Figure 2-4 Speech command input format

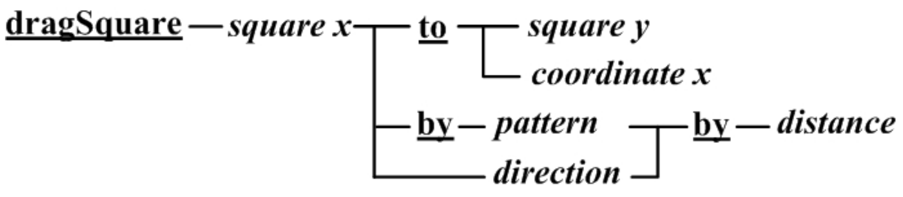


Figure 2-5 Script language syntax tree of command "dragsquare"

2.4 Compile grammar definition

The designed rule of script language follows from the format of function "//Microsoft

Speech SDK 5.1/ Microsoft Speech SDK 5.1 Help/grammar compiler/索引/Designing Grammar Rules". Consequently, we designed our speech recognizing rule of interface system based on the designing grammar rules in "Microsoft Speech SDK 5.1 Help," from file "test.xml".

<GRAMMAR>

The GRAMMAR tag is the outermost container for the XML grammar definition.

<DEFINE>

The DEFINE tag is used for declaring a set of string identifiers for numeric values.

<ID>

The ID tag is used for declaring a string identifier for numeric values.

<RULE>

The RULE tag is the core tag for defining which commands are available for recognition.

<RULEREF>

The RULEREF tag is used for importing rules from the same grammar or another grammar. The RULEREF tag is especially useful in reusing component or off-the-shelf rules and grammars.

<L>

The L tag is used for specifying a list of phrases or transitions, and anything in it should be spoken once but not all times.

<P>

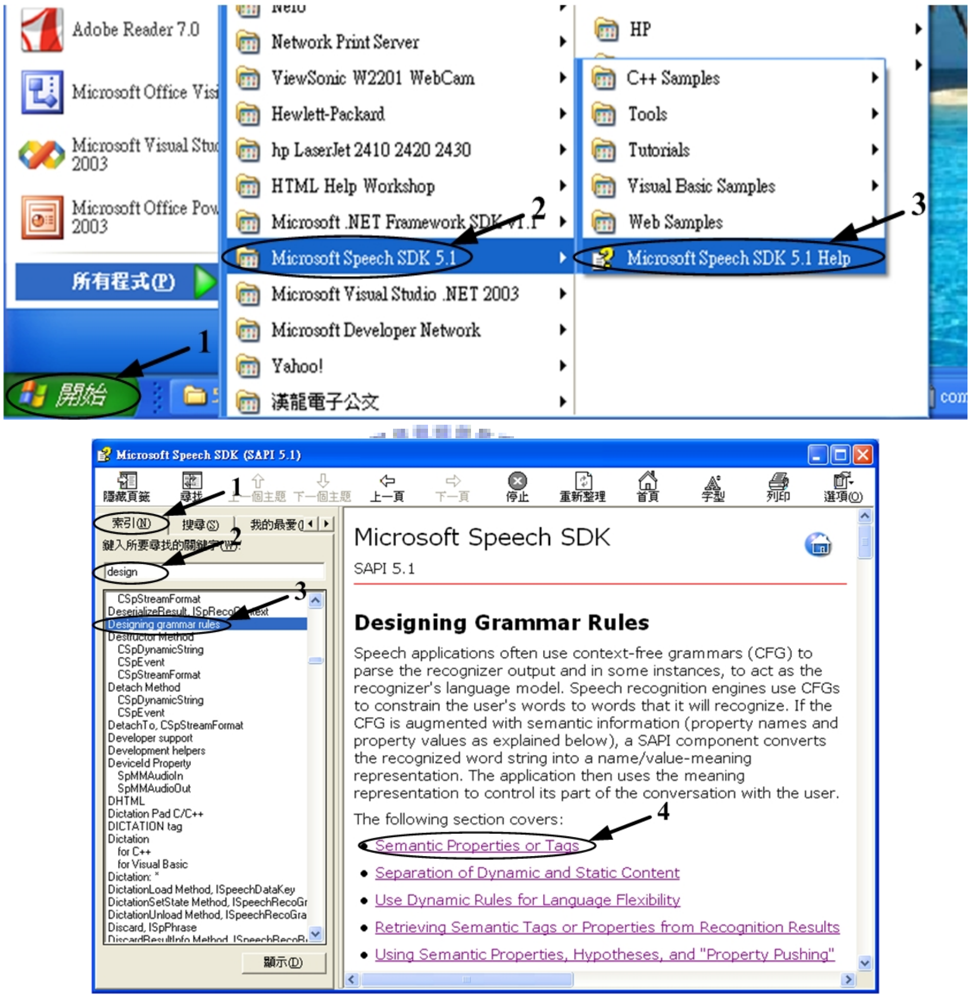
The P tag is used for specifying text to be recognized by the speech recognition engine, or anything in it should be spoken.

<O>

The O tag is used for specifying optional text in a command phrase.

In regards to the help on checking grammar composition, we can use the function "//Microsoft Speech SDK 5.1/tools/grammar compiler/Build" to check the correctness of rules.

If the compiling result is successful, it would show compile successful; else wise it would show compile fail and depict the error reasons. The process figures of Microsoft Speech SDK and Microsoft Grammar Compiler are shown in Figure 2-6 and 2-7.



Microsoft Speech SDK

SAPI 5.1

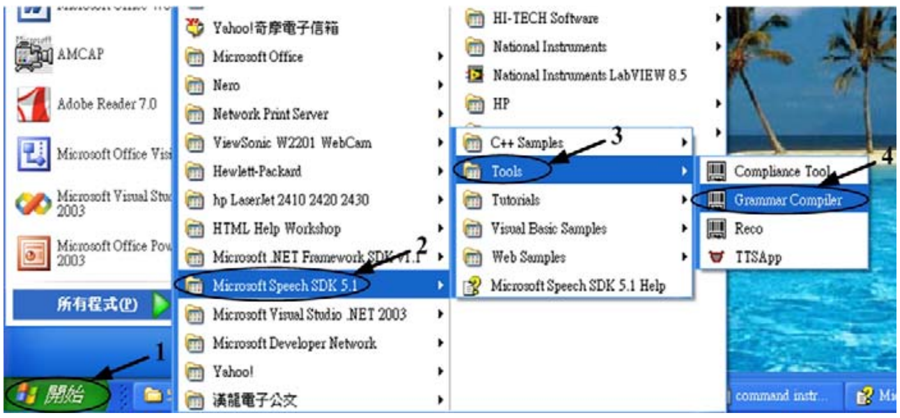
Designing Grammar Rules

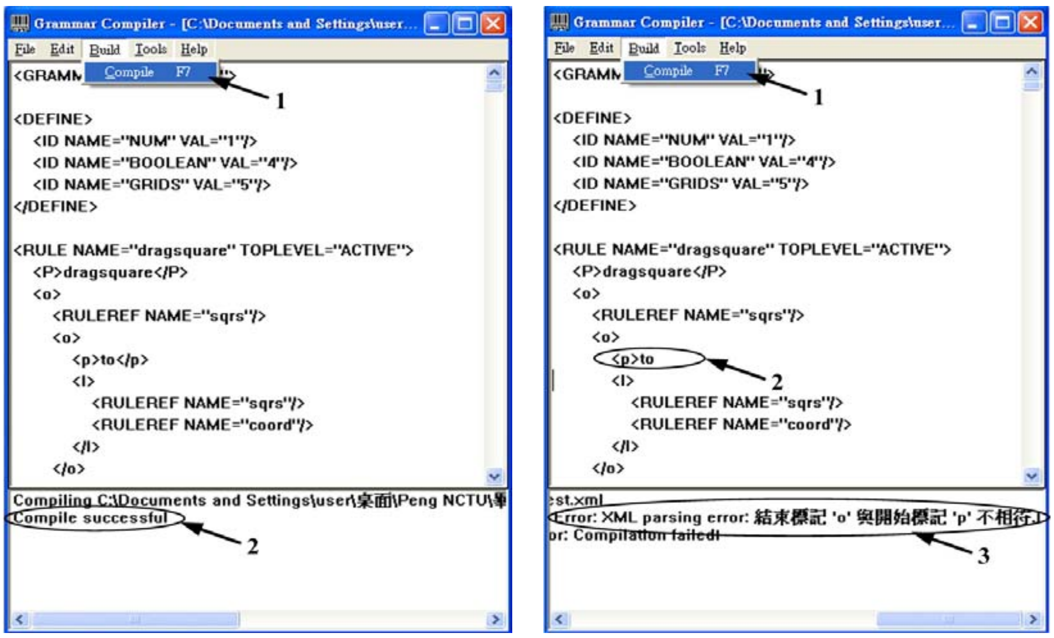
Speech applications often use context-free grammars (CFG) to parse the recognizer output and in some instances, to act as the recognizer's language model. Speech recognition engines use CFGs to constrain the user's words to words that it will recognize. If the CFG is augmented with semantic information (property names and property values as explained below), a SAPI component converts the recognized word string into a name/value-meaning representation. The application then uses the meaning representation to control its part of the conversation with the user.

The following section covers:

- Semantic Properties or Tags
- Separation of Dynamic and Static Content
- Use Dynamic Rules for Language Flexibility
- Retrieving Semantic Tags or Properties from Recognition Results
- Using Semantic Properties, Hypotheses, and "Property Pushing"

Figure 2-6 Microsoft Speech SDK




Grammar Compiler - [C:\Documents and Settings\user...
File Edit Build Tools Help
Compile F7
1
2
Grammar Compiler - [C:\Documents and Settings\user...
File Edit Build Tools Help
Compile F7
1
2
3

```
<GRAMM
<DEFINE>
   <ID NAME="NUM" VAL="1"/>
   <ID NAME="BOOLEAN" VAL="4"/>
   <ID NAME="GRIDS" VAL="5"/>
</DEFINE>

<RULE NAME="dragsquare" TOPLEVEL="ACTIVE">
   <P>dragsquare</P>
   <o>
      <RULEREF NAME="sqrs"/>
      <o>
         <p>to</p>
         <l>
            <RULEREF NAME="sqrs"/>
            <RULEREF NAME="coord"/>
         </l>
      </o>
```

```
Compiling C:\Documents and Settings\user\桌面\Peng NCTU\畢
Compile successful
```

```
<GRAMM
<DEFINE>
   <ID NAME="NUM" VAL="1"/>
   <ID NAME="BOOLEAN" VAL="4"/>
   <ID NAME="GRIDS" VAL="5"/>
</DEFINE>

<RULE NAME="dragsquare" TOPLEVEL="ACTIVE">
   <P>dragsquare</P>
   <o>
      <RULEREF NAME="sqrs"/>
      <o>
         <p>to
         <l>
            <RULEREF NAME="sqrs"/>
            <RULEREF NAME="coord"/>
         </l>
      </o>
```

```
st.xml
Error: XML parsing error: 結束標記 'o' 與開始標記 'p' 不相符
or: Compilation failed!
```

Figure 2-7 Microsoft Grammar Compilers

2.5 Macro command

In the proposed system, we use speech as one of the control tools; if speech commands are too long, the correctness of recognition result would be affected due to the pronunciation of people or the noise of environment between command and commands. According for easy interaction with GIB (Generic Interface Bridge) system, we adopt macro command method to simplify and combine longer or complex commands into single context-free command. For users this method is much easier and more flexible to input commands and to increase the recognition correctness and efficiency. The command format of macro command and relative complex or compound commands are as following.

Macro command	Relative commands
Clickfile	move to file then click by leftclick

; move mouse cursor to defined position named "file," and then left click the mouse

Start menu	*g02-@cm

; move mouse cursor to the position stored before, then left click the mouse. The command “then” and “-” is used as a separator of commands.

2.6 OS’s API

Upon obtaining abstract commands from HCI, GIB translates command lexical and analyzes the syntactic, validating those tokens and then manipulating the actions of mouse and keyboard through calling OS’s API. In the proceeding section, we will describe the parameters of mouse API and keyboard API.

```
Dllimport (“user32.dll”)
```

Mouse API:

```
Private Shared sub mouse_event (
    Byval dwFlags as mouse_event_flags,
    Byval dx as integer,
    Byval dy as integer,
    Byval dwData as integer,
    Byval dwextrainfo as integer)
End
```

dwFlags:

MOUSEEVENTF_ABSOLUTE: Specifies that the *dx* and *dy* parameters contain normalized absolute coordinates. If not set, those parameters will contain relative data: the changes in position since the last reported position.

MOUSEEVENTF_MOVE: Specifies that movement occurred.

MOUSEEVENTF_LEFTDOWN: Specifies that the left button is down.

MOUSEEVENTF_LEFTUP: Specifies that the left button is up.

MOUSEEVENTF_RIGHTDOWN: Specifies that the right button is down.

MOUSEEVENTF_RIGHTUP: Specifies that the right button is up.

MOUSEEVENTF_MIDDLEDOWN: Specifies that the middle button is down.

MOUSEEVENTF_MIDDLEUP: Specifies that the middle button is up.

dx: It specifies the absolute position of mouse along the x-axis or its amount of motion since the last mouse event was generated, depending on the setting of MOUSEEVENTF_ABSOLUTE.

dy: It specifies the absolute position of mouse along the y-axis or its amount of motions since the last mouse event was generated, depending on the setting of MOUSEEVENTF_ABSOLUTE.

dwData: If dwFlags contains MOUSEEVENTF_WHEEL, then dwData specifies the amount of wheel movement. A positive value indicates that the wheel was rotated forward and away from the user; a negative value indicates that the wheel was rotating backward and towards the user.

dwExtraInfo: It specifies an additional value associated with the mouse event.

Keyboard API:

```
Private Shared sub keybd_event (
    Byval bVk as byte,
    Byval bScan as byte,
    Byval dwFlags as long,
    Byval dwExtraInfo as integer)
End sub
```

bVk: It specifies a virtual-key code defined in MSDN. The code value is from 1 to 254.

bScan: This parameter is not used.

dwFlags: It specifies various aspects of function operation.

dwExtraInfo: It specifies an additional value associated with the key stroke..

3. Integration GIB with HCI

The architecture of the proposed GIB system for bridging HCI and applications is as shown in figure 3-1. This GIB system includes three parts: input module, kernel module and output module. The main functions of input module include the reception of controlling signal from user into the system. When controlling signal is valid, the system would translate input control-signal to relative abstract command string of script language from the command database. The kernel module breaks down the abstract command string into token sets, and makes syntactic analysis for each token set. Finally, the output module would call OS's API, to simulate mouse and keyboard control according to the real command request.

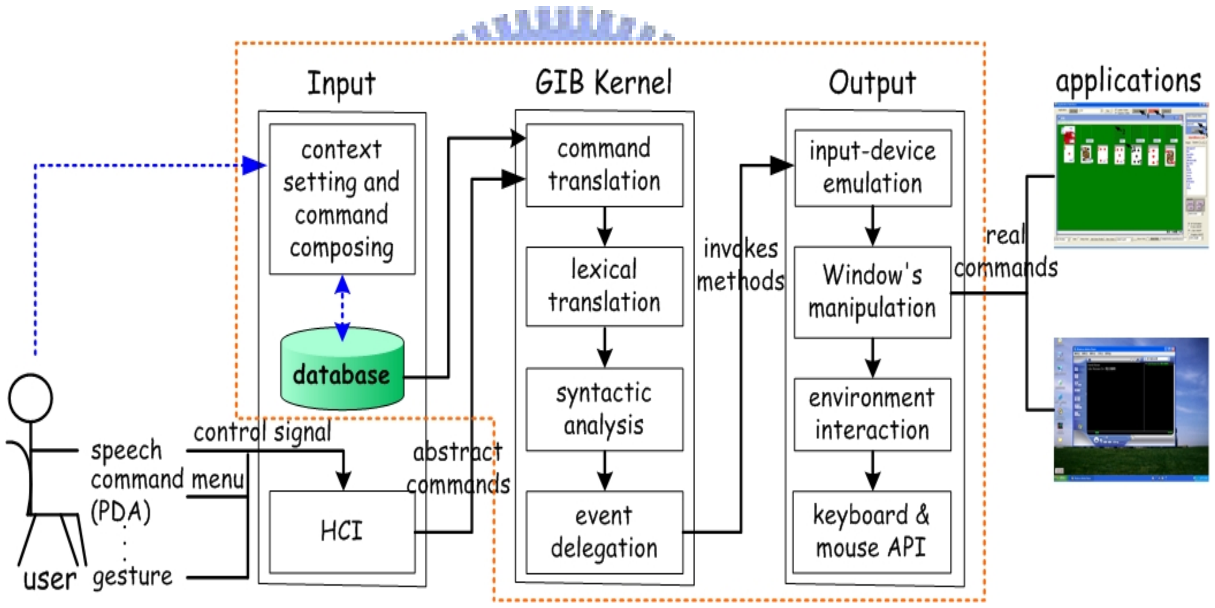
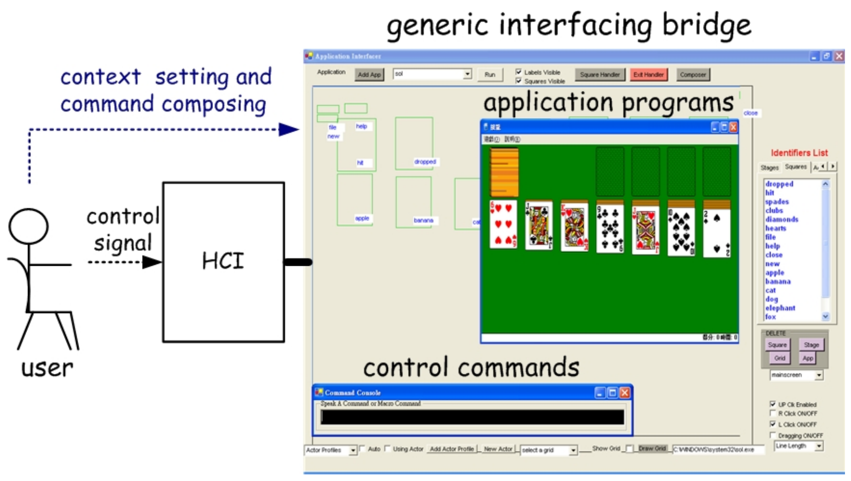
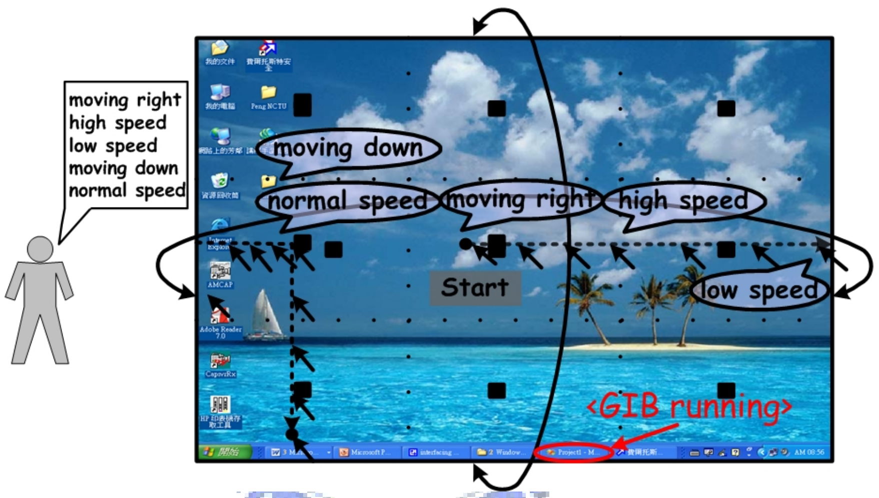


Figure 3-1 System architecture of GIB

To conveniently introduce the advantage of the GIB system, we classify GIB system architectures into two approaches: 1. Run Application in GIB (RAGIB) Windows, 2. Separate Application and GIB (SAGIB) Windows, as shown in figure 3-2 (a) and (b).



(a) Run application in GIB Windows



(b) Separate application and GIB Windows

Figure 3-2 Run application under GIB system

In RAGIB system, application must be run under the GIB system. This implies that, every time if we want to control an application under RAGIB system, we must select the application and execute it under the proposed GIB system. For a more convenient control, we proposed a SAGIB system. In SAGIB, GIB system can be run behind the Windows and would not occupy the extra working space of Windows, and consequently makes the control of application more conveniently. That means we can control all the applications which are executed on the Windows by controlling mouse and keyboard through control signal from user, under the SAGIB system.

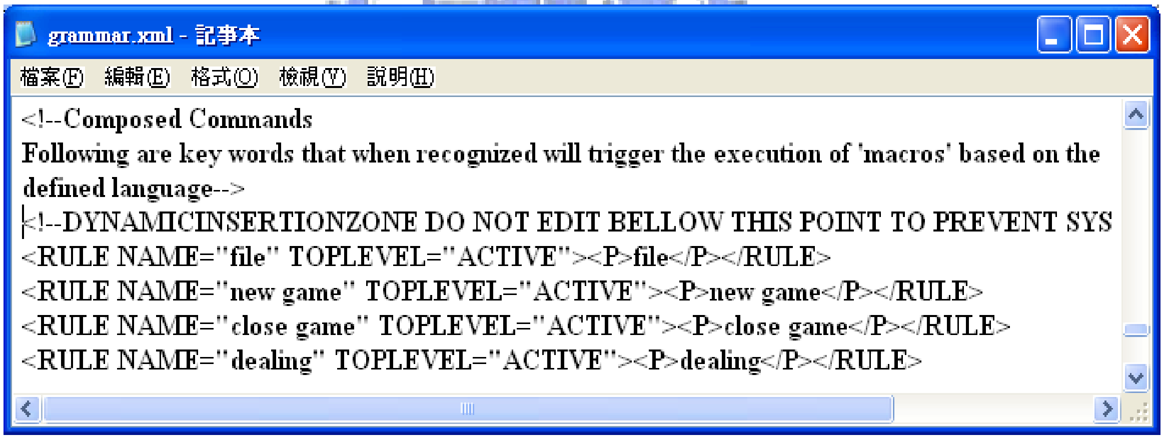
3.1 Run Application in GIB (RAGIB) Windows

3.1.1 Input module

The main functions of input module include speech command recognizer, context setting and commands composing. We can use speech recognizer to recognize inputting speech command, defined control objects and environment setting such as objects defining, grids setting, macro command setting and stage registering.

3.1.1.1 Speech command

We use Microsoft's Speech SDK V5.1 as the speech input recognizer of this GIB system, and the speech commands must be stored under the file "grammar.xml," as shown in figure 3-3, and if not, the commands would not be recognized.

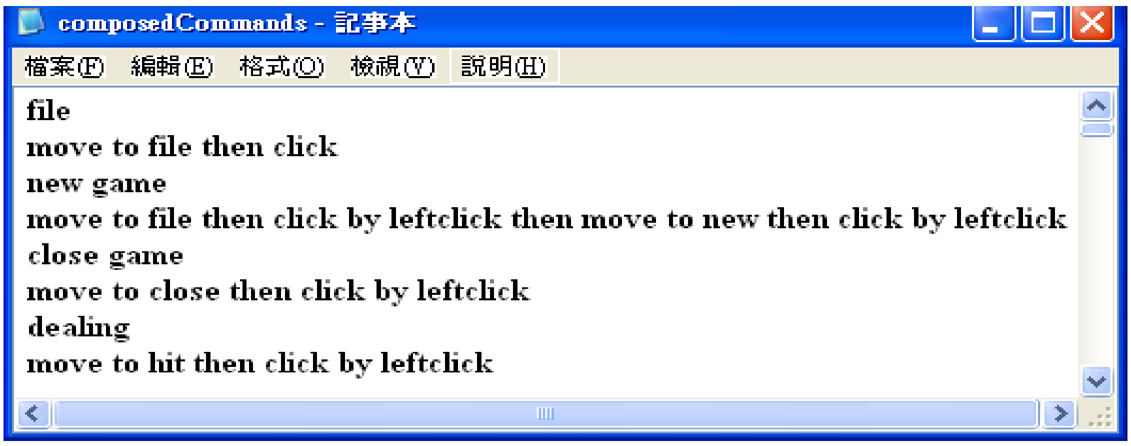


```
<!--Composed Commands
Following are key words that when recognized will trigger the execution of 'macros' based on the
defined language-->
<!--DYNAMICINSERTIONZONE DO NOT EDIT BELLOW THIS POINT TO PREVENT SYS
<RULE NAME="file" TOPLEVEL="ACTIVE"><P>file</P></RULE>
<RULE NAME="new game" TOPLEVEL="ACTIVE"><P>new game</P></RULE>
<RULE NAME="close game" TOPLEVEL="ACTIVE"><P>close game</P></RULE>
<RULE NAME="dealing" TOPLEVEL="ACTIVE"><P>dealing</P></RULE>
```

Figure 3-3 Format of file Grammar.xml

3.1.1.2 Composed command

The relative command string of macro commands are stored into the file "composed command," the file format is as shown in figure 3-4. If the command spoken valid, this system would get the relative commands string from the database.

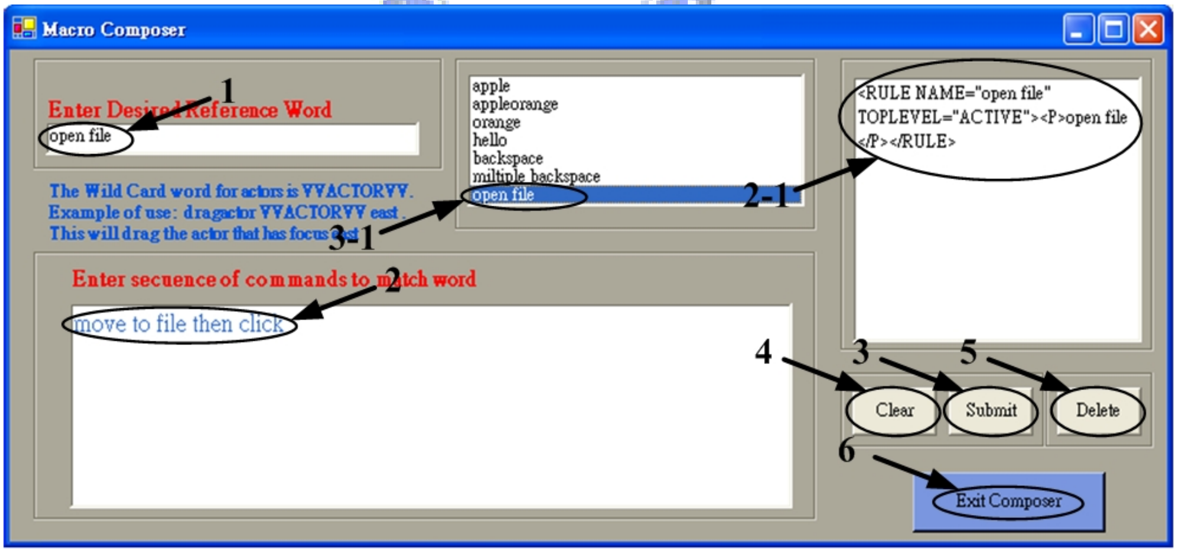

composedCommands - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)

```
file
move to file then click
new game
move to file then click by leftclick then move to new then click by leftclick
close game
move to close then click by leftclick
dealing
move to hit then click by leftclick
```

Figure 3-4 Format of composed commands

3.1.1.3 Macro command registration

For using this system in a more convenient manner, we develop a GUI interface for user to register macro commands and to store the commands string to file “grammar.xml” and “composed command” automatically. The registration interface of macro commands is as shown in figure 3-5.



Enter Desired Reference Word

open file

The Wild Card word for actors is VVACTORVV.
Example of use: dragactor VVACTORVV east.
This will drag the actor that has focus east

```
apple
appleorange
orange
hello
backspace
miltiple backspace
open file
```

```
<RULE NAME="open file" TOPLEVEL="ACTIVE"><P>open file</P></RULE>
```

Enter secuence of commands to match word

move to file then click

Figure 3-5 Macro command registration

3.1.1.4 Command translation flow

For example, if the user speaks a macro command "open file," this would be compared with the commands set in the file "grammar.xml". If the command is found valid, it would get and translate relative command string "move to close then click by left click" from the file "composed command," as in figure 3-6.

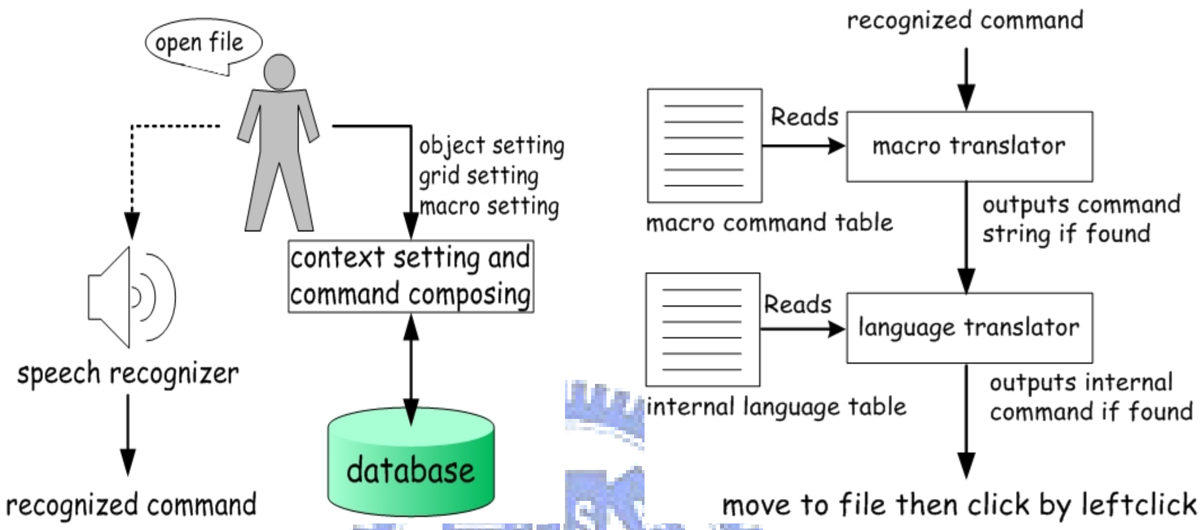


Figure 3-6 Translation flow of Speech command

3.1.2 Kernel module

The main functions of Process module include separating command string into token sets and analyzing its syntax. If the command is valid after analyzing, it would be executed by calling OS's API. The processing flow is shown in figure 3-7.

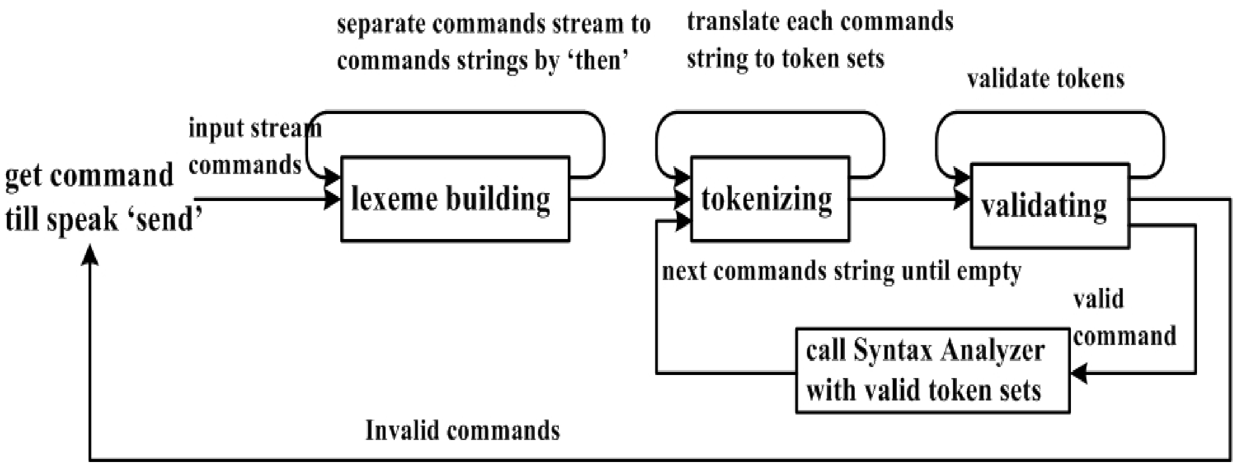

separate commands stream to commands strings by 'then'
translate each commands string to token sets
validate tokens
input stream commands
get command till speak 'send'
lexeme building
tokenizing
validating
next commands string until empty
valid command
call Syntax Analyzer with valid token sets
Invalid commands

Figure 3-7 Command processing flow

3.1.2.1 Lexical translator

The lexical translator separates command string into token sets according to the separator command “then,” as shown in figure 3-8. Each token set represents a stand-along command that is sent to syntax analyzer.

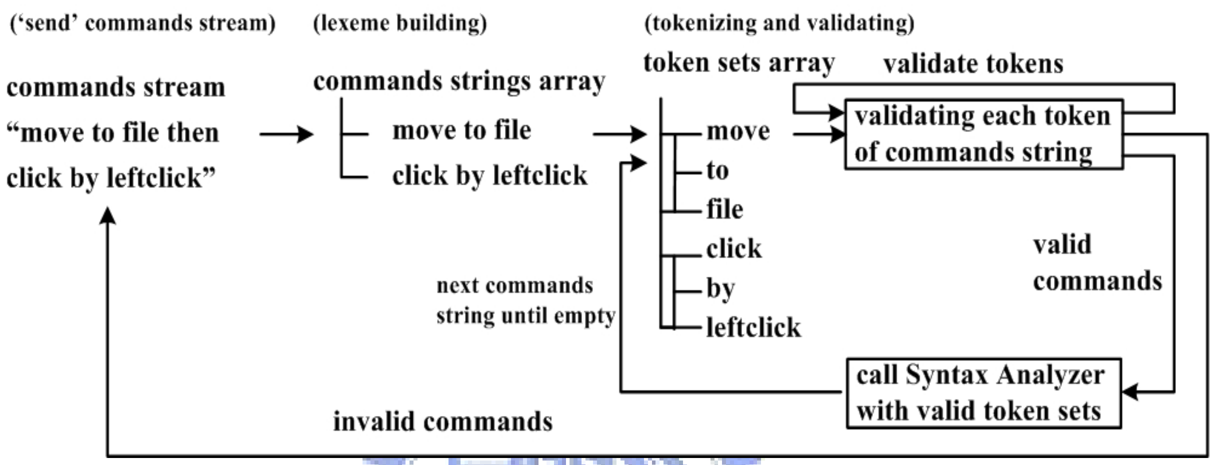


Figure 3-8 Lexical translating

3.1.2.2 Syntax analyzer

After receiving one token set at a time sent from section 3.1.2.1 lexical translator, it would check into their syntax by following the grammatical rules defined in the file “grammar.xml,” as in figure 3-9.

```
<RULE NAME="move"
TOPLEVEL="ACTIVE">
    <P> move </P>
    <o>
        <P> to </P>
        <l>
            <RULEREF NAME="sqrs"/>
            <RULEREF NAME="coord"/>
        </l>
    </o>
    <o> <RULEREF NAME="mse"/>
        <o>
            <p> by </p>
            <RULEREF NAME="mse"/>
        </o>
    </o>
    <o> <RULEREF NAME="then"/> </o>
</RULE>
```

```
<RULE NAME="click"
TOPLEVEL="ACTIVE">
    <P> click </P>
    <o>
        <P> by </P>
        <RULEREF NAME="mse"/>
    </o>
    <o> <RULEREF NAME="then"/> </o>
</RULE>
```

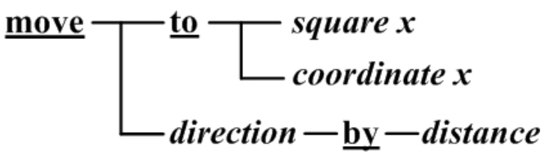


Figure 3-9 Grammatical rules in file “grammar.xml”

3.1.2.3 Command analyzer flow

For example, a command string “move to close then click by left click” obtained from section 3.1.1, would be broken into token sets of “move to close” and “click by left click” by lexical translator. After analyzing, it would be parsed into target function events and then call the corresponding OS’s API:

1. Move cursor to square object which named close,

2. Perform a left click of mouse button, as shown in figure 3-10.

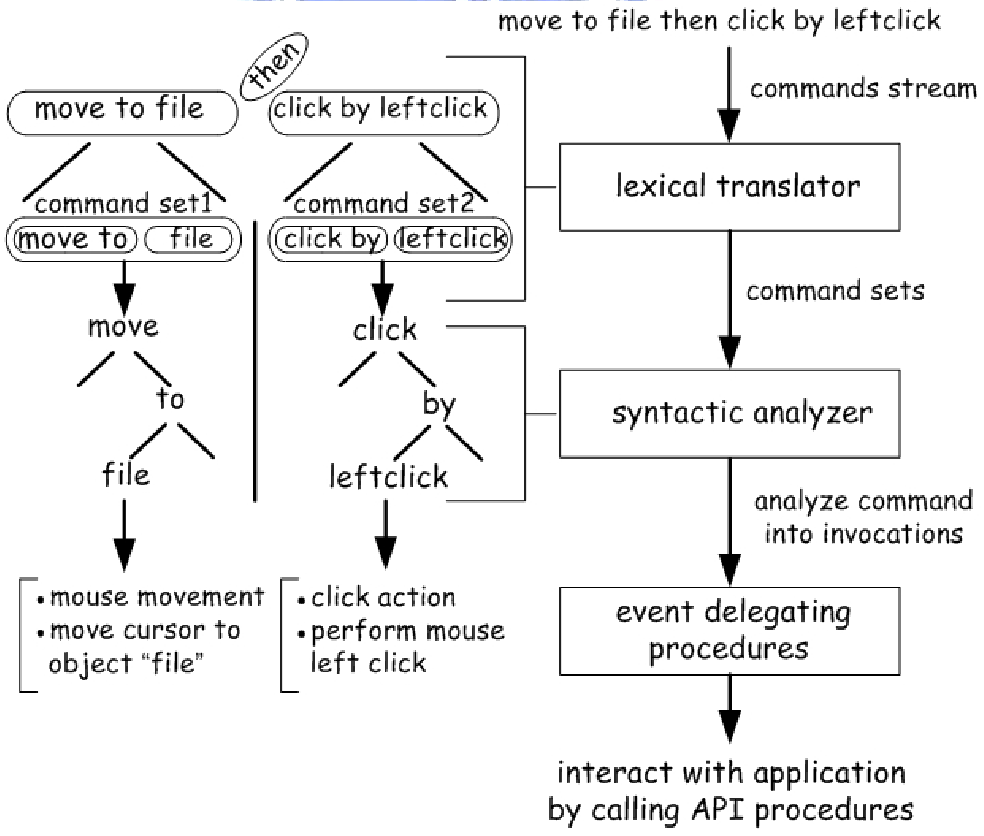


Figure 3-10 Grammatical rule flow in file “grammar.xml”

3.1.3 Output module

The main function of this part is to execute the recognition result from the syntax analysis by calling OS's API. To perform the related actions of mouse and keyboard easier, we have written functions to emulate the actions, as in shown in figure 3-11. The mouse control functions are shown in the following: "mouse-left-click," "mouse-right-click," "mouse-left-double-click," "mouse drag," "mouse drop," "mouse spring," "mouse jumping" to pre-defined position, "mouse move" with different speed, "move stop," and "mouse moving speed". Whereas the keyboard control functions include the following: single key press (0-9, a-z, and @, #, etc), compound key (Alt-, Shift-, and Ctrl-), Chinese/English input interchange, article input. A more detailed illustration is discussed in next section.

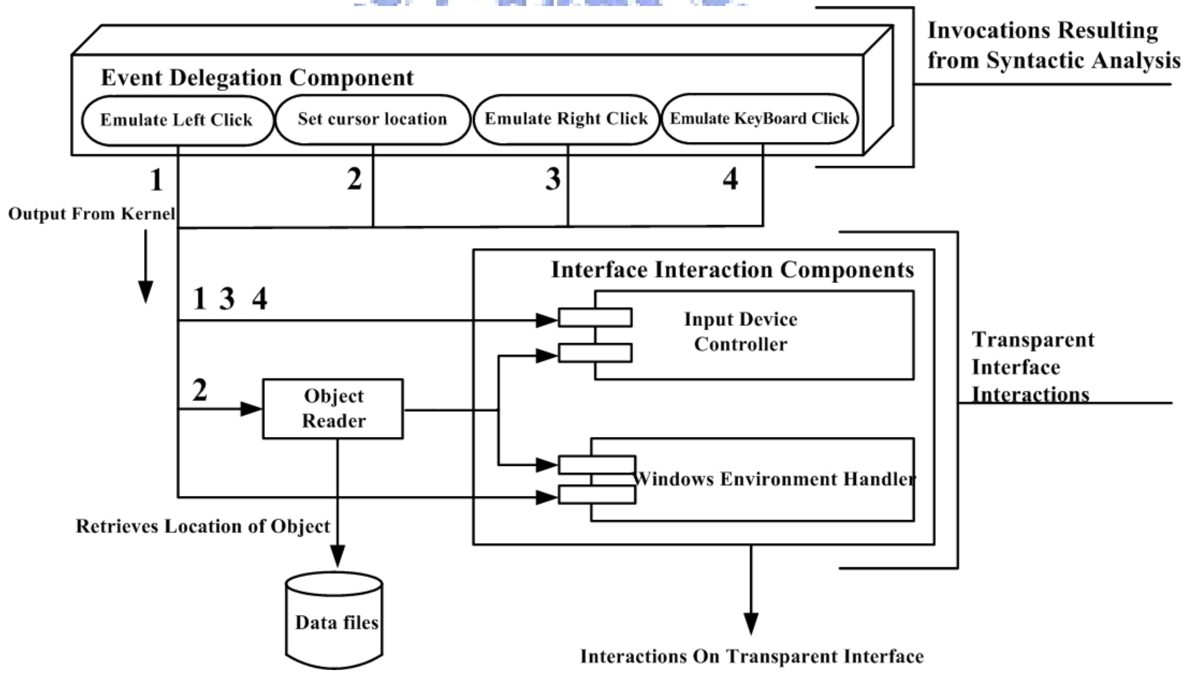

Invocations Resulting from Syntactic Analysis
Event Delegation Component
Emulate Left Click
Set cursor location
Emulate Right Click
Emulate KeyBoard Click
1
2
3
4
Output From Kernel
Interface Interaction Components
1 3 4
Input Device Controller
Transparent Interface Interactions
2
Object Reader
Windows Environment Handler
Retrieves Location of Object
Data files
Interactions On Transparent Interface

Figure 3-11 Event delegation

The application program and information, such as defined squares, grids, locations, macro commands, and parameters, are stored into a hierarchical directory structure, as in figure 3-12.

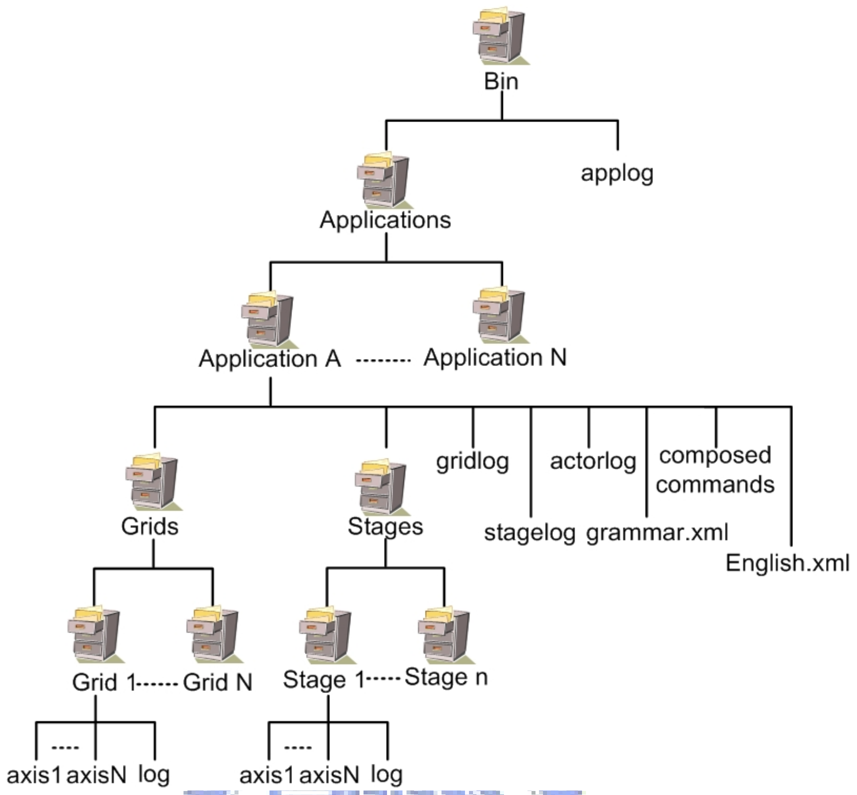


Figure 3-12 File hierarchical organization

3.1.4 Identifiers

3.1.4.1 Command class

The control commands can be classified as some sections according to their operation functions: "Selection commands," "Setting commands, "Mouse action commands," "Keyboard action commands," "File action commands" and "System action commands," and will be described in the following sections.

3.1.4.2 Command parameter

Square:

The "square" object is a graphical rectangle drawn by the user from the direction left-up to right-down at static desired location. The variable of square is associated with locations (integer, integer).

Coordinate:

The "coordinate" object is a graphical rectangle grid, which is auto-generated by the system, and drawn by the user from direction left-up to right-down at static desired location. The variable of each grid in the "coordinate" is associated with locations (integer, integer).

Stage:

The "stage" is a file that stores group sets of squares.

Grid:

The "grid" is a file that stores group sets of grids squares.

Number:

The "number" operand is an integer and used with commands "coordinate" or "loop."

Text:

The "text" operand is a text string. It is used with command "sendkey" for emulating typing on the keyboard.

3.1.4.3 Connector

Then:

The connector "then" is used for conjunction with statement x and statement y, and is delimited from one statement to others.

3.1.4.4 Terminator

Times:

The terminator "times" is used as the end of loop command.

Separator

Comma:

The separator "," separates the coordinate x and y of grid.

3.1.4.5 Operators

To:

The "to" operator is used for conjunction with the action or assignment commands. It is strictly used for commands "move," "dragsquare" and "dragcoordinate," for use with right operands of data "*square x*" and "*coordinate x*," and for use with left operands of data "*square x*" and "*coordinate x*".

By:

The "by" operator is used for conjunction with the assignment commands. It is strictly used for commands "move," "drag," "dragsquare," "dragcoordinate," "click," "clicksquare" and "clickcoordinate," for use with right operands of data "*distance*" and "*clicktype*," and for use with left operands of data "*square x*," "*coordinate x*," "*direction*" and "*pattern*".

Loop:

The "loop" operator is used for executing repeatedly pre-spoken commands for "*munber*" times. Its left operand is always a command, and its right operand is always a "*number*".

3.1.4.6 Constant parameters

clickType:

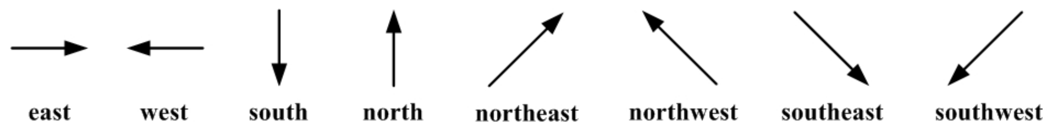
The types of "clicktype" include: *leftclick*, *rightclick* and *doubleclick*. This "clicktype" type is used for triggering a mouse click.

Distance:

The types of "distance" include: *very short, short, normal, long and very long.* This distance type is used for restricting the moving "distance" of mouse cursor.

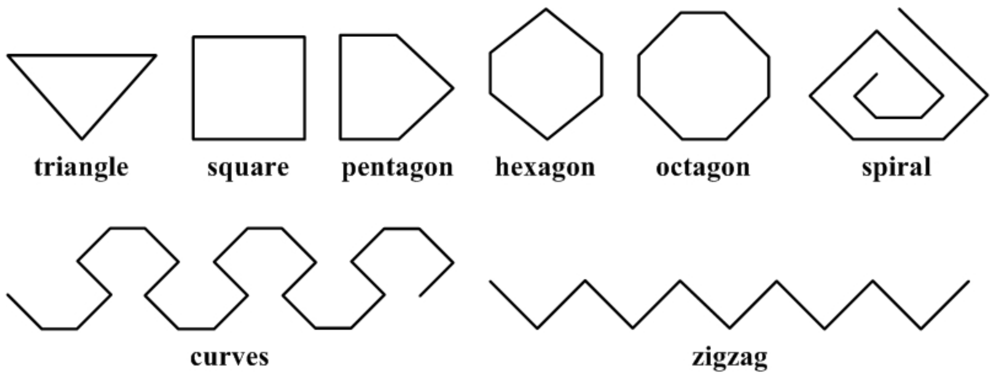
Direction:

The types of "direction" include: *east, west, south,* and *north, northeast, northwest,* *southeast* and *southwest*. This “direction” type is used for setting the moving direction of mouse cursor.



Pattern:

The types of geometric moving “pattern” include: *triangle, square, pentagon, hexagon, octagon, curves, zigzag,* and *spiral*. This “pattern” type is used for setting the moving manner of mouse cursor.



Speed:

The types of “speed” include: *very slow, slow, normal, fast* and *very fast*. This “speed” type is used for restricting the moving speed of mouse cursor.

Boolean:

The types of “boolean” include: *true* and *false*. This “boolean” type is used for setting the command “setdrop”.

3.1.4.7 Selection commands

selectApplication:

The “selectApplication” command is used for selecting an application program into the GIB system to execute.

selectStage:

The “selectStage” command is used for selecting a stage file and loading all its defined squares into the executing application.

selectGrid:

The “selectGrid” command is used for selecting a grid file and loading all its defined grids into the running application.

3.1.4.8 Setting commands

setDragSpeed:

The “setdragspeed” command is used for setting the mouse cursor to move speed according to the parameter “speed”.

setDistance:

The “setDistance” command is used for setting the mouse cursor moving distance according to the parameter “distance”.

setDrop:

The “setDrop” command is used for disabling the drop of drag command according to the parameter “boolean”.

showGrid:

The “showGrid” command is used to show or hide defined grids according to the parameter “boolean”.

3.1.4.9 Mouse action commands

Move:

The “Move” command is used for controlling the mouse cursor moving to the specified location, according to the parameters “direction” and “distance”. If the “distance” is omitted, it would follow the setting parameter via command “setDistance”

Drag:

The “Drag” command is used for dragging the mouse cursor moving to the specified location, according to the parameters “direction” and “distance”.

dragSquare:

The “dragsquare” command is used for dragging the specified square x moving to the specified location square y or coordinate x, according to the parameters “direction” ,”distance” and “pattern”.

dragCoordinate:

The “dragCoordinate” command is used for dragging the specified coordinate x moving to the specified location coordinate y or square x, according to the parameters “direction,” “distance” and “pattern”.

Click:

The “Click” command is used for triggering a mouse click under the current mouse position, according to the parameter “clicktype”.

clickSquare:

The “clickSquare” command is used for moving cursor to the specified square location first and then triggering a mouse click under the current position according to the parameter “clicktype”.

clickCoordinate:

The “clickCoordinate” command is first used for moving cursor to the specified coordinate location and secondly triggering a mouse click under the current coordinate position according to the parameter “clicktype”.

3.1.4.10 Keyboard action commands

sendKey:

The “sendKey” command is used for emulating keyboard stroke, and is limited to the combination of number 0 to 9 and character “a” to “z”.

clearText:

The "clearText" command is used for emulating keyboard <Backspace> stroke once.

3.1.4.11 System action commands

Send:

The "Send" command is a stand-along command and used for sending all the spoken commands to lexical translator, in order to analyze and process spoken commands.

clearConsole:

The "clearConsole" command is a stand-along command, and used for clearing all the spoken commands in the input command buffer.

undoPhrase:

The "undoPhrase" command is a stand-along command and used for deleting the preceding spoken command stored in the input command buffer.

storeCursor:

The "storeCursor" command is a stand-along command and used for storing the current cursor position into position buffer.

recoverCursor:

The "recoverCursor" command is a stand-along command and used for moving cursor to the previous stored position of mouse cursor from position buffer.

captureIt:

The "captureIt" command is a stand-along command and used for capturing the most front application program windows into the GIB interface system.

3.1.5 Process description

The process description of GIB system is illustrated in the following:

Step1. Waiting for control commands input

Step2. Recognizing inputted commands

```
Step3. Analyzing whether the inputted commands are valid and defined in the macro
       composer
       Select Case (inputted command)
           Case (the command is macro)
                  Translate inputted commands according to command table
                  Store translated command to command buffer, go to Step1
           Case "system action command"
                  Examples:
                  "Send"; store speech commands to command buffer, then go to Step4
           Case (the command is invalid)
                  Discard inputted speech command, and then go to Step1.
        End Select
Step4. Lexical translation:
         Parsing command string to token set according to separator "then"
         Store translated token sets to command string array
Step5. Syntactic analysis:
       pointer = 0
       Do while (commands string array [pointer] < > empty)
           Select case (command string)
               Case "mouse action command"
                     Examples:
                     "Move to file"; move mouse cursor to defined square object "file"
               Case "keyboard control"
                     Examples:
                     "SendKey hello"; press keys "hello"
               Case "selection commands"
```

```
            Examples:
            "SelectApplication mspaint"; selecting application "mspaint" into system to
              run
        Case "setting commands"
            Examples:
            "SetDragSpeed slow"; set mouse cursor drag moving speed slow
        Case else
            Commands are un-defined; discard it and clear command buffer
      End select
      pointer = pointer + 1
    End do
Step6. Repeat Step1, until system end.
```

The corresponding operation flow chart of GIB system is described in figure 3-13.

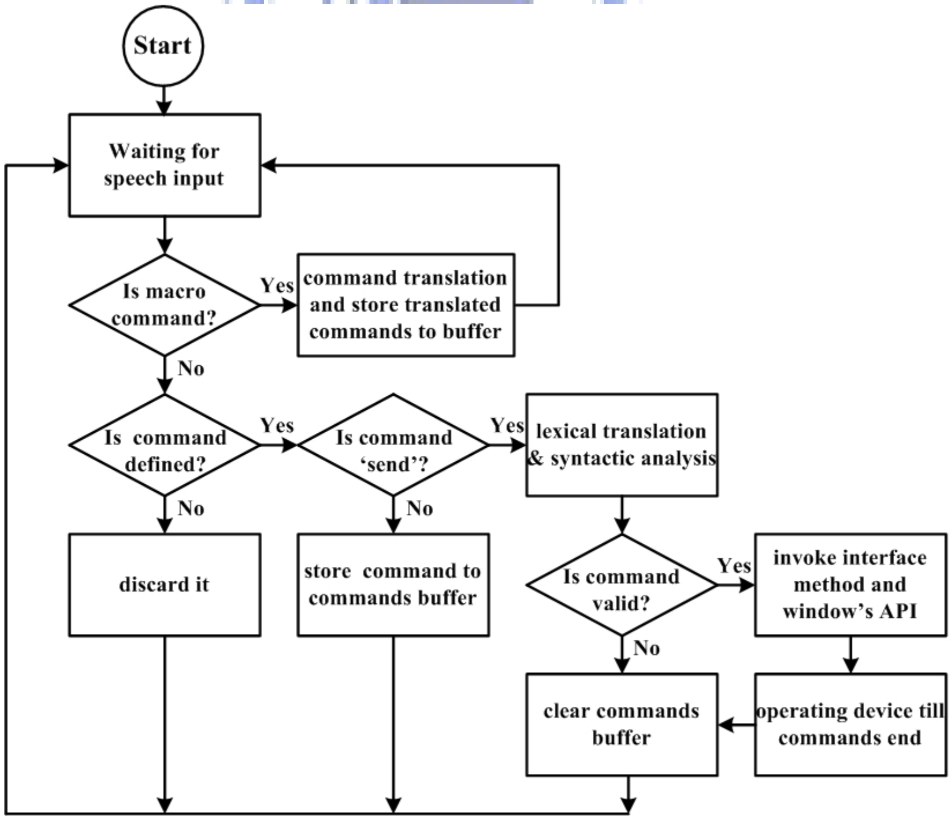

Start
Waiting for speech input
Is macro command?
Yes
command translation and store translated commands to buffer
No
Is command defined?
Yes
Is command 'send'?
Yes
lexical translation & syntactic analysis
No
discard it
No
store command to commands buffer
Is command valid?
Yes
invoke interface method and window's API
No
clear commands buffer
operating device till commands end

Figure 3-13 Operating flow chart of GIB system

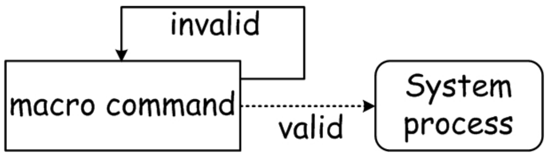
3.1.6 Limitation

In the method of “Run Application in GIB Windows,” we can easily control the application which has no prior speech recognition control ability, via speech commands under the RAGIB system. But even so, there are still some limitations under this system:

1. Application programs must be executed under the proposed RAGIB environment.
2. The input commands would be analyzed after command “send” has been spoken.
3. User would be unable control mouse moving to undefined position with speech command.
4. The label name of defined square objects would affect the action control of mouse cursor.

3.2 Separate Application and GIB (SAGIB) Windows

For solving problems mentioned in section 3.1, we will make some modification to the GIB system and separate application from the GIB Windows. It implies that GIB would be executed behind the Windows and applications need not to be executed under the SAGIB system. The speech command would be sent to the SAGIB system after the validation of automatic recognition, without have to wait for the speech command “send”. This would increase the correctness of recognition without disturbance from the noise of the environment during time space of input commands and command “send,” while in turn this process would also make the system control more user-friendly.



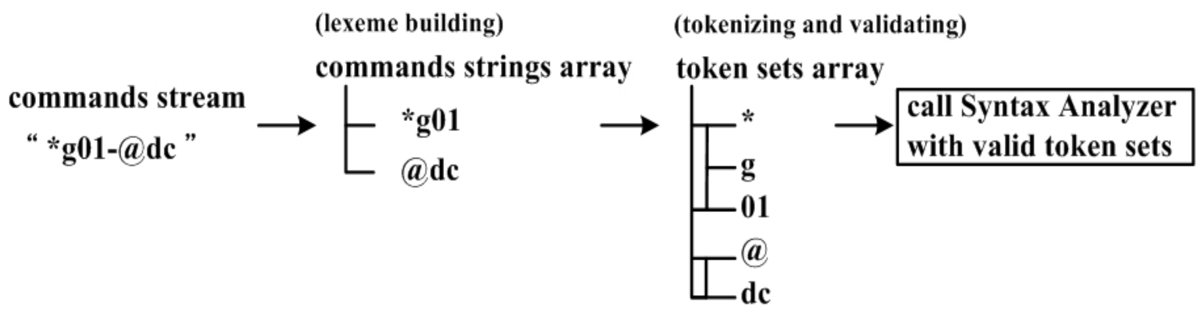


Figure 3-14 Rule composition flow

For accelerating movement control of mouse cursor, we virtually cut the Windows into 3*3 big grid space and 10*8 small grid space, and pre-define each center coordinate of small grid space a name (x, y), x: 1~10 and y: 1~8, such as (2, 4), and named the grids of big space as “left-up,” “center-left,” “left-down,” “center-up,” “center-axis,” “center-down,” “right-up,” “center-right,” and “right-down.” For a more precise mouse cursor movement, we define the whole screen as working space with wrapped-around pixel reference. The flow of rule composition is shown in figure 3-14, and the mouse control environment is shown in figure 3-15.

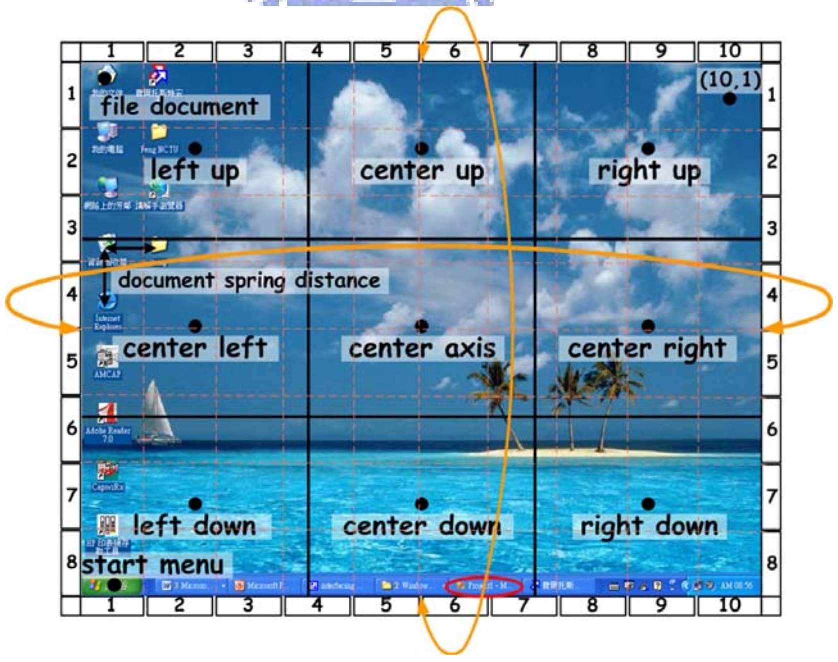
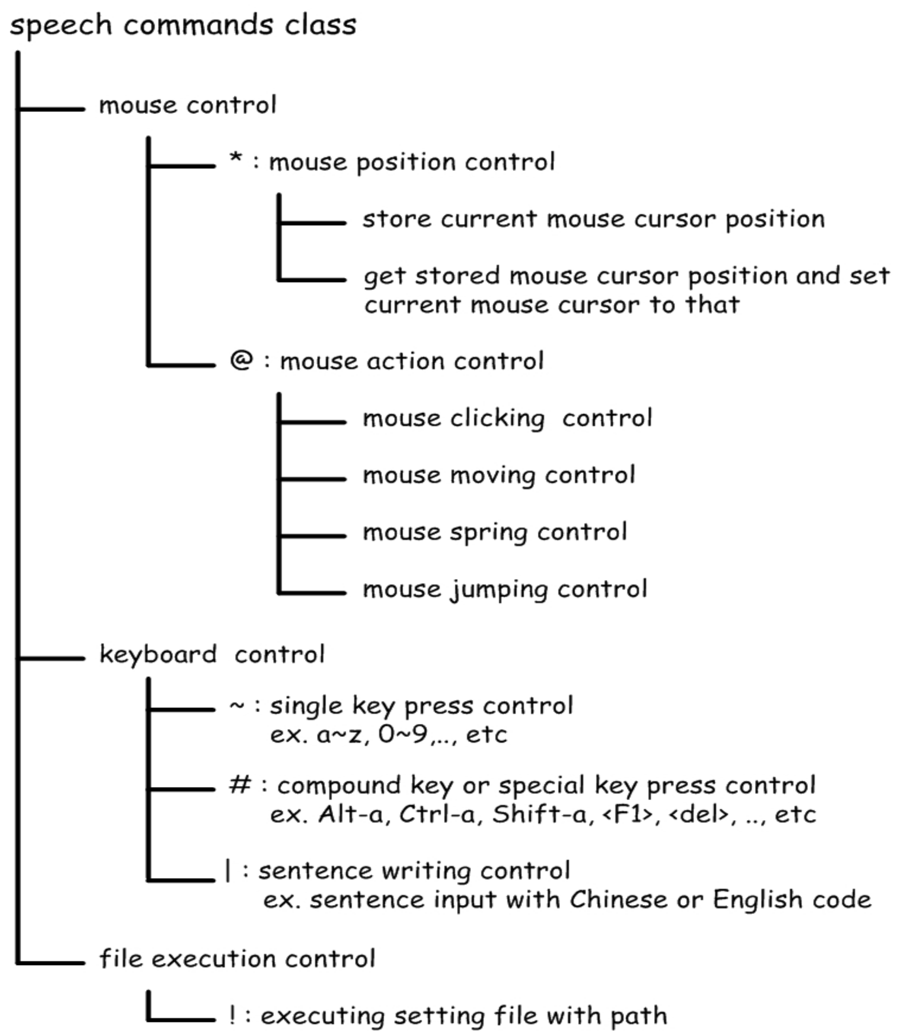


Figure 3-15 Mouse action environments

3.2.1 Identifiers

3.2.1.1 Command class

We have classified the speech commands into sections: "mouse position control," "mouse action control," "keyboard control" and "file execution control," as shown in the following.


speech commands class
mouse control
* : mouse position control
store current mouse cursor position
get stored mouse cursor position and set current mouse cursor to that
@ : mouse action control
mouse clicking control
mouse moving control
mouse spring control
mouse jumping control
keyboard control
~ : single key press control ex. a~z, 0~9,.., etc
# : compound key or special key press control ex. Alt-a, Ctrl-a, Shift-a, <F1>, <del>, .., etc
| : sentence writing control ex. sentence input with Chinese or English code
file execution control
! : executing setting file with path

Command input format:

<command$_1$> - <command$_2$> - ‧ ‧ <command$_n$>

"-": The connector "-" is used as a separator with statement x and statement y in composed commands, and is delimited from one statement to the others.

3.2.1.2 Mouse position control

Mouse position control includes:

1. Store position of current mouse cursor as an object,

2. Set current mouse cursor to that gotten position stored before

*sn	; s: store current mouse cursor position
*gn	; g: get stored mouse cursor position before and set current mouse cursor position to the gotten position
	; n: 01~99, number of stored position (or defined) in program

3.2.1.3 Mouse action control

Mouse action control includes:

1. Mouse clicking control: control mouse button action via commands “click mouse,” “right click,” “double click,” “drag” and “release,”

2. Mouse moving control: control mouse cursor moving continuously or stop with different speed via one command,

3. Mouse spring control: control mouse cursor spring once to the directions, “up,” “down,” “left,” and “right” with distance via system default or set by the user,

4. Mouse jumping control: control mouse cursor jumping to the default locations, “right up,” “center down,” (2, 3), etc, pre-defined by the system.

Mouse clicking control:

Example:

@cm	; click mouse left button once
@dc	; double click mouse left button

Mouse moving control:

Example:

@ms ; control mouse cursor moving stop

@mu ; mouse cursor moving up

Mouse spring control:

Example:

@su ; mouse spring up

@v01 ; set Vertical spring distance y1, jumpstepx=|y2-y1|

@v02 ; set Vertical spring distance y2, jumpstepx=|y2-y1|

Mouse jumping control:

Example:

@ru ; mouse jumping to the pre-defined location "right up"

move to 2,5 send ; mouse jumping to the pre-defined object coordinate "2, 5"

3.2.1.4 Keyboard control

Keyboard control includes the follows:

1. Single key press control: control single keyboard pressed actions
2. Compound key press control: control compound keyboard pressed action
3. Special key press control: control special command keyboard pressed action
4. Sentence writing control: sentence writing with English or Chinese code

Single key press control:

~<char> ; manipulating single key pressed, <char>: a~z, 0~9

Ex. "~f" means "f" key pressed, etc.

Compound key press control:

Example:

#a<char> ; manipulating compound keys <Alt>-<char> pressed, <char>: a~z, 0~9

Special key press control: control special command keyboard pressed

Example:

#f<num>	; manipulating function key pressed, <num>: 1~12

Sentence writing control: control sentence writing with English or Chinese code

#il	; compound key "$\boxed{\text{Ctrl}}$ $\boxed{\text{Space}}$" pressed, "change language"
#ii	; compound key "$\boxed{\text{Ctrl}}$ $\boxed{\text{Shift}}$" pressed, "change input"
	|<En char string>]<CH char string>:
#ie	; set parameter emode=1, "english code"
#ic	; set parameter emode=0, "chinese code"

3.2.1.5 File execution control

File execution control: executing file by setting application file path in macro.

File execution control:

![file path]	; execute file stored in the system for path=[file path]
	; Ex. "!c:\program files\microsoft office\office11\execel.exe"

3.2.2 Process description

The process description of SAGIB system is illustrated in the following:

```
Step1. Waiting for speech commands input
Step2. Recognizing control commands
Step3. Analyzing whether the inputting commands are valid and defined in the macro
       composer
       If (inputted command is invalid) then
           Discard inputted command and clear commands buffer, then go to Step 1.
       Else
           Translate inputted commands according to command table
           Store translated command to command buffer
       End If
Step4. Lexical translation:
    Parsing command string to token set according to separator “-”
    Store translated token sets to command string array
Step5. Syntactic analysis:
    pointer = 0
    Do while (commands string [pointer] < > empty)
    Select case (command string)
      Case “mouse position control”
          Examples:
          “*$s_n$”; store current cursor position to database record n;
          “*$g_n$”; set mouse current cursor to the position n stored before.
      Case “mouse action control”
          Examples:
          “@dc”; double click mouse left button,
          “@mu”; mouse cursor moving up continuously
      Case “keyboard control”
          Examples:
          “~niceday”; press keyboards “niceday” continuously.
          “#aa”; press $\boxed{\text{Alt}}$ $\boxed{\text{a}}$ simultaneously
      Case “sentence writing control”, commands format is as “|***]###”
          If (variable enmode=0) then
              write out characters “***” with Chinese mode;
          Else if (enmode=1) then
              write out characters “###” with English mode
          End if
      Case “file execution control”
          Ex. “!c:\program files\microsoft office\office11\execel.exe”
      Case else
            Commands are un-defined; discard it and clear command buffer
   End select
   pointer = pointer + 1
  End do
```

Step6. Repeat Step1, until system end.

The corresponding operation flow chart of modified GIB system is described in figure 3-16.

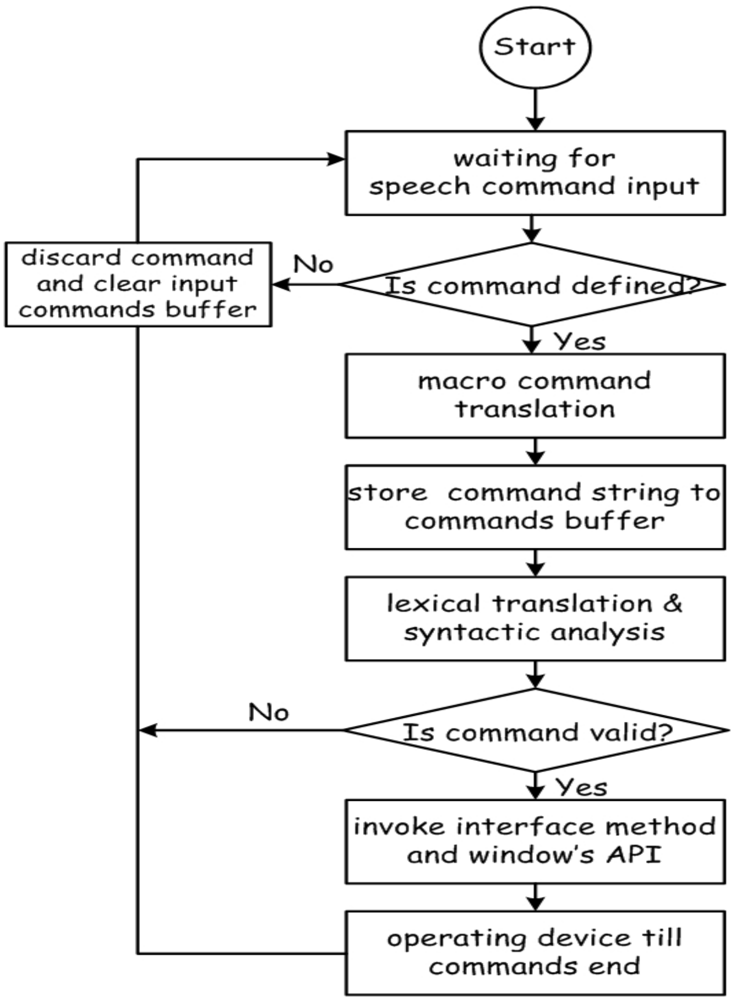

Start
waiting for speech command input
Is command defined?
No
discard command and clear input commands buffer
Yes
macro command translation
store command string to commands buffer
lexical translation & syntactic analysis
Is command valid?
No
Yes
invoke interface method and window's API
operating device till commands end

Figure 3-16 Operating flow chart of modified GIB system

4. GIB-based Application Interface (GAI) Generation

4.1 interface

On the traditional developing of remote control functions into the cellular phone, the designer needed to write the complex Midlet program into cellular phone at first, and then write the JAVA AP controlling interaction statement between PC and the cellular phone. It must redesign and recompile the whole application while adding or deleting functions. This designed process was a hard job, which wasted much time and was highly inefficient, as shown in figure 4-1.

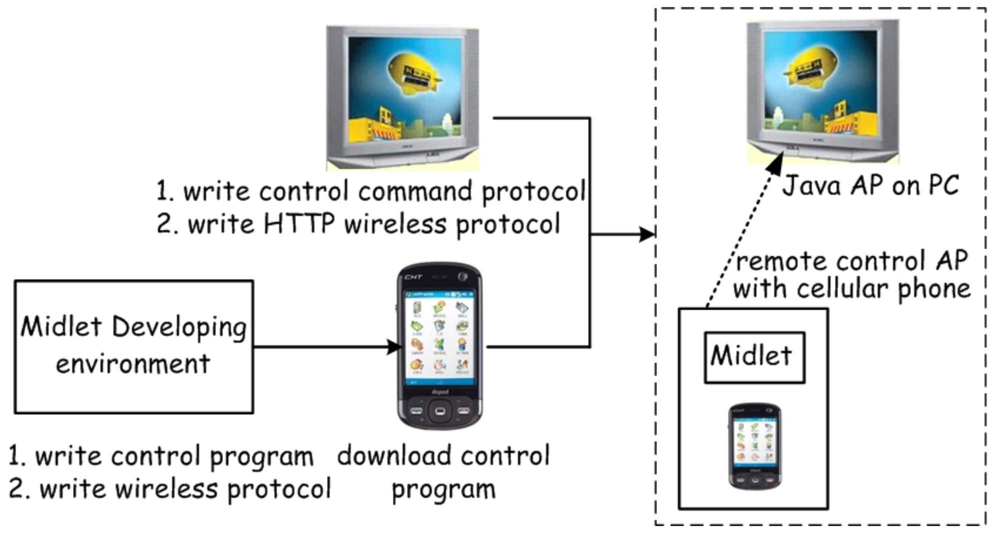

1. write control command protocol
2. write HTTP wireless protocol
Java AP on PC
remote control AP with cellular phone
Midlet Developing environment
Midlet
1. write control program
2. write wireless protocol
download control program

Figure 4-1 Framework of traditional cellular phone interface system

Our proposed solution for solving this problem is to construct an interface generating system; with which the designer can easily develop remote control interface program into cellular phone without the need to write any program code of cellular phones. Finally, using this cellular phone as a remote controller, users may interact with the JAVA application running on the PC directly and easily. The framework of remote control interface system is as shown in figure 4-2.

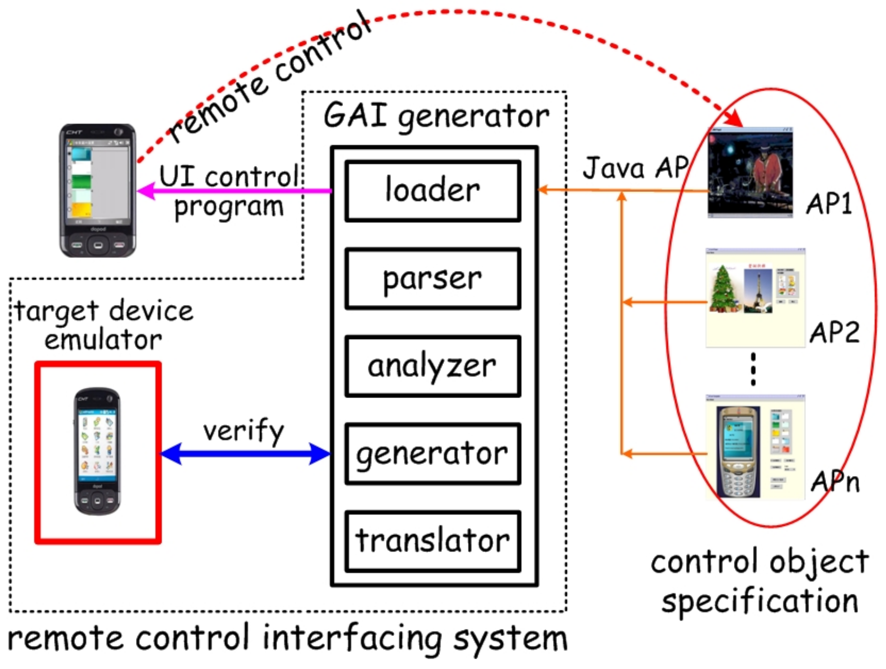
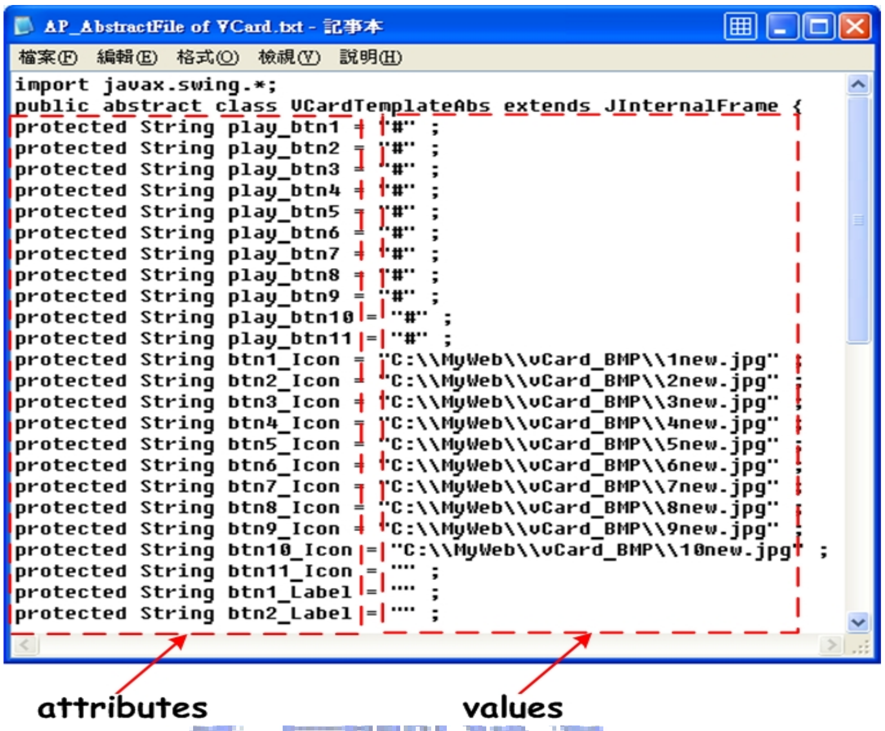


Figure 4-2 Framework of proposed cellular phone control interface system

4.2 Procedure

The GAI generation modules includes JAVA contents, AP interface loader, parser, analyzer, interface generator, and translator, as shown in figure 4-2. Module "JAVA content" refers to the JAVA applications running on the PC controlled with cellular phone. Module AP interface loader includes "AP loader," which loads JAVA AP specification file and UI into the generation system, while "packs remote control statement," and "UI command parser" process the interface control command of cellular phone. Finally, module "interface generator" includes "code template parser," which analyzes abstract classes of AP, and according to these classes to generate operating script file and control command table, and "remote control statement," which processes HTTP linking and transfer control command and "code generation" to generate Midlet of cellular phone automatically according to the operating script file. Finally, it executes the Wireless ToolKit (WTK) compiler and packs program into jar file, then the designer downloads this jar file into the cellular phone. The sample of specification file of AP includes attributes and values of control objects on the AP, as described in figure 4-3.


AP_AbstractFile of VCard.txt - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)
attributes
values

```
import javax.swing.*;
public abstract class VCardTemplateAbs extends JInternalFrame {
protected String play_btn1 = "#" ;
protected String play_btn2 = "#" ;
protected String play_btn3 = "#" ;
protected String play_btn4 = "#" ;
protected String play_btn5 = "#" ;
protected String play_btn6 = "#" ;
protected String play_btn7 = "#" ;
protected String play_btn8 = "#" ;
protected String play_btn9 = "#" ;
protected String play_btn10 = "#" ;
protected String play_btn11 = "#" ;
protected String btn1_Icon = "C:\\MyWeb\\vCard_BMP\\1new.jpg" ;
protected String btn2_Icon = "C:\\MyWeb\\vCard_BMP\\2new.jpg" ;
protected String btn3_Icon = "C:\\MyWeb\\vCard_BMP\\3new.jpg" ;
protected String btn4_Icon = "C:\\MyWeb\\vCard_BMP\\4new.jpg" ;
protected String btn5_Icon = "C:\\MyWeb\\vCard_BMP\\5new.jpg" ;
protected String btn6_Icon = "C:\\MyWeb\\vCard_BMP\\6new.jpg" ;
protected String btn7_Icon = "C:\\MyWeb\\vCard_BMP\\7new.jpg" ;
protected String btn8_Icon = "C:\\MyWeb\\vCard_BMP\\8new.jpg" ;
protected String btn9_Icon = "C:\\MyWeb\\vCard_BMP\\9new.jpg" ;
protected String btn10_Icon = "C:\\MyWeb\\vCard_BMP\\10new.jpg" ;
protected String btn11_Icon = "" ;
protected String btn1_Label = "" ;
protected String btn2_Label = "" ;
```

Figure 4-3 sample of specification file of AP "VCard"

In the figure 4-4, it shows the interface modules of cellular phone. We describe its process procedures as the follows:

AP interface loader:

Load AP UI

1. loading JAVA AP UI into system
2. loading specification file of AP

UI command parser

3. parsing HTTP command
4. process control command of UI

AP interface generator:

Code template parsing module

1. analyzing abstract classes of AP

2. parsing code template

3. generating operation script file and control table

Remote control statement module

1. processing HTTP link

2. transferring control command

Code generating module

1. generating remote control table

2. generating Midlet program

3. executing WTK compiler

Deploy Java Midlet application:

Download PACK JAR into cellular phone

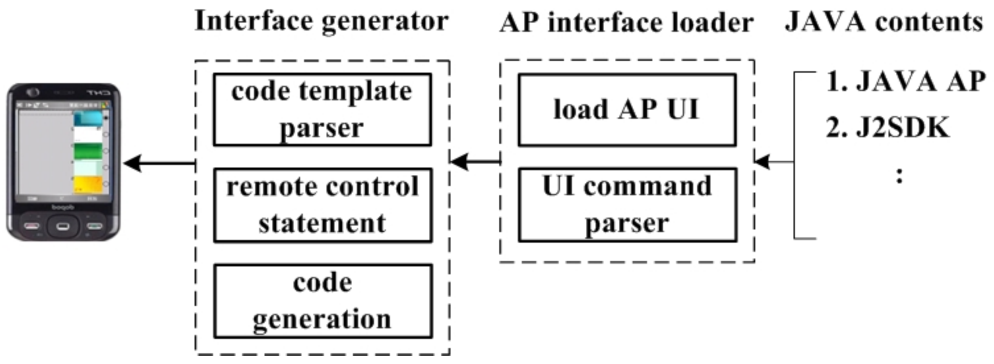


Figure 4-4 Cellular phone interface modules

Figure 4-5 shows the interface generated procedures of cellular phone. We describe its process procedures as the follows:

1. Input AP control object specification file

2. Generate target code for cellular phone

3. Verify target code by running in the emulator

4. Port verified target code to cellular phone

5. Use cellular phone to control AP running on remote host

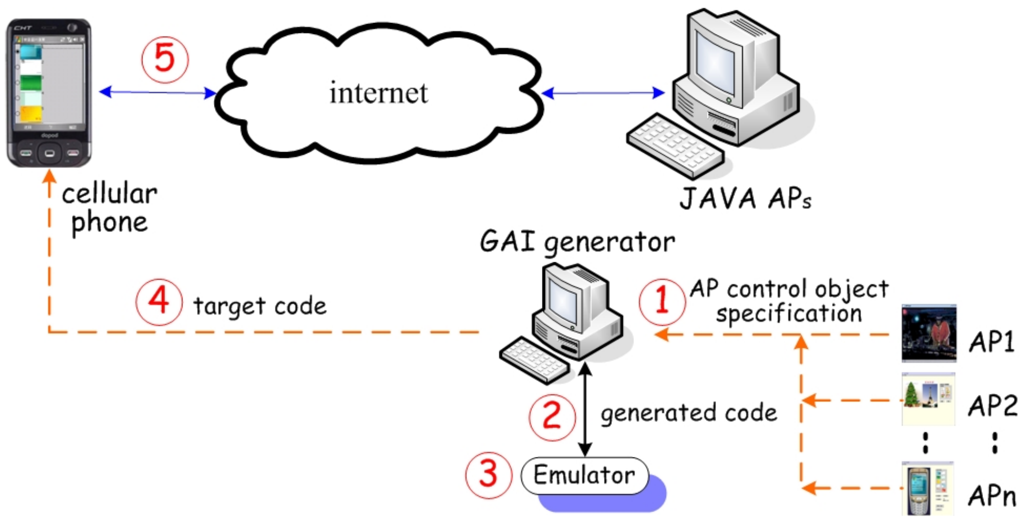


Figure 4-5 Cellular phone interface generated procedures

4.3 Algorithm

Control interface systematic framework is composed of application program interface loader, cellular phone interface generator, and a Java application program. Application program interface loader is a graphic user interface (GUI) for designers, which faciliates loading JAVA applications running on the PC into interface generator. It is a JAVA server procedure; whereas system user makes JAVA application program template and provides this template to the interface loader. After loading application interface program, it links and transforms operation objects, such as Javax.swing.JButton and Javax.swing.JReadButton, into commands ID, such as "001#" for play_btn_1 and "002#" for play_btn_2. In the following section, we will describe algorithms of these modules respectively.

4.3.1 Algorithm of application interface loader

Algorithm 1-1 Loading AP UI

1. Initializing JAVA swing GUI component, such as JFrame, JSplitPane, JScrollPane, JMenuBar, JMenu, JRadioButtonMenuItem and JDesktopPane.

2. Create look and feel properties by calling setLookandFee (String) function.
3. Add all kinds of GUI components into JFrame by calling JFrame.add (Component).
4. Set default size and look&feel of JFrame.
5. Waiting for launching Java Application Program.
6. Initializing Java Application Program.
7. Loading the JInternalFrame of the target Java Application Program.

```
If (ActionListener of JMenu Received JMenuItem ActionEvent)
    {
    Switch (ActionEvent)
        {
        Case "Launch Java _AP_X":                    //X:1~N
            LoadAP (Java_AP_X);
                    :
        }
    }
```

8. Parsing the code template (abstract class) of the target Java Application Program by calling BufferedReader (FileReader) and FileReader (abstract class name: String).

```
BufferedReader br = new BufferedReader (new FileReader ("JavaAPName"));
While (br.ready ())
    {
    String s=br.readLine ();
    Parsing s and retrieve the String Token;
    Switch (String Token)
        {
        Case "play_btnN":                              //N:1~n
            Get the command ID value of play_btnN;   //ex: play_btn1 = "001#"
        Case "btnN_Icon":
            Get the Icon path of play_btnN;              //ex: btn1_Icon = "1new.jpg";
                        :
        }
    }
```

9. Generating the Operation File "AP_ControlTable.txt" by calling BufferedWriter (FileWriter), FileWriter (AP_ControlTable.txt: String).

```
BufferedWriter bw = new BufferedWriter (new FileWriter "AP_ControlTable.txt"));
While (br.ready ())
    {
    String s=br.readLine ();
    If (s.startsWith ("protected String"))
        {
        st=new StringTokenizer (s, "");
        ValueOfCmd=st.nextToken ();
        bw.write (Command ID+""+ ValueOfCmd);
        bw.newLine ();
        }
    }
```

10. Load the JTable of the target Java Application Program.

```
br = new BufferedReader (newFileReader ("AP_ControlTable.txt"))
While (br.ready ())
    {
    String content = st.nextToken ();
    Switch (content)
        {
```

```
        Case “TYPE”:
          Assign content to CurrentTableData[X] [0];    //X:0~n
        Case “ICON” | “LABEL”:
        Assign content to CurrentTableData[X] [1];
                              :
        }
      }
```

According to system framework of this paper, each Java application operating screen can be loaded in by application program interface loader, which is designed by adopting JAVA swing groupware. From line 1 to line 4, before loading applications, we first initialize actuating devices of interface loader which is similar to a vessel containing application interfaces, such as Menu, Menu Bar, Radio Button, Split Pane, and Scroll Pane. When application is loaded on, we parse program code and find out all command IDs of actuating devices on this application at line 8, then record the searching results into file AP_ ControlTable.txt as at line 9.

Algorithm 1-2 Parsing UI Command

```
1. Waiting for HTTP URL request from the MIDlet Program in mobile phone.
2. Parsing the HTTP parameters to retrieve the command ID.
   response.setContentType (“text/html”);
   PrintWriter out = response.getWriter ();
   String command = request.getParameter (“message”);
3. Retrieve command value in JTable by calling getValueAt (row index, column index);
4. Compare HTTP command ID and value in JTable by String.equals (String).
   For (int i=0, i<row count, i++)
     {
```

```
    If (command.equals (JTable.getValueAt (i, column index)))
        rc_setXXXActionPerformed (command);
    }
```

5. If command is identical, calls control functions "rc_setXXXActionPerformed".

In fact, interface loader precisely plays a server role. Upon recording all operating project and command ID, interface loader then waits for a HTTP connection from remote cellular phone procedures. From line 1 to 4, it illustrates that while interface loader received HTTP command, it would compare with the field value in JTable to find what kind of remote control command was delivered. This was named as rc_setXXXActionPerformed which supplied programmer with ability to define and write its functions, as at line 5.

4.3.2 Algorithm of control interface generator

Algorithm 2-1 Code Generation

1. Generate the Remote Displayable Form.

```
  Import javax.microedition.lcdui.*;
  Import java.io.*;
  Public class RemoteDisplayable extends Form implements CommandListener
```

2. Generate MIDlet class.

```
  Public class remoteMIDlet extends MIDlet
    {
    static remoteMIDlet instance;
    RemoteDisplayable displayable = new RemoteDisplayable2 ();
    Display display;
    }
```

3. Decide the Component Types (ex: Button/RadioBtn) in the command table.

4. Generate the Main List included possible list items of Component Types.

```
If (parsing the command types in table = BUTTON|RADIOBTN|COMBOBOX)
    {
    String Component_Array [] =new String [] {"Button", "RadioBtn", "ComboBox"};
    mainList = new List ("main menu", List.EXCLUSIVE, Component_Array, null);
    mainList.addCommand (mainList_OK_cmd);
    mainList.addCommand (mainList_EXIT_cmd);
    mainList.setCommandListener (this);
    }
```

5. Generate the operation items in operation list; a main list may include several operation lists- Button/RadioBtn/Combobox).

```
For (int i=0; i<NumOfButton; i++)
    {
    form btn_form[i] = new form ();
    ChoiceGroup choiceGroup[i] = new ChoiceGroup ();
    For (NumOfOperations_IN_ ith_Button)
        choiceGroup[i].add ("the i-th Button Operations");
    form[i].append (choiceGroup[i]);
    form[i].addCommand (opForm_OK_cmd);
    form[i].addCommand (opForm_EXIT_cmd);
    form[i].setCommandListener (this);
    add choiceGroup[i] into the i-th button operation form - form[i];
    add all button operation forms into mainList;
}
```

6. Send HTTP Commands according to the command table.

```
Protected void sendCommand (String command)
    {
```

```
    set url = "http://140.113.208.118:8080/MyWeb/ResponseTest?message=";
    set url = url + command;
    call readyConnect ();
    }

  Public void readyConnect ()
    {
    connectThread = new Thread (this);
    Try
      {
      call the HTTP connection thread and execute run () method;
      connectThread.start ();
      }
    Catch (Exception ex);
    }

  Public void run ()
    {
    Try
      {
      send HTTP connection request with a command by calling connect (url);
      }
    Catch (Exception ex);
    }
```

After computing mathematical process of algorithms 1-1 and 1-2, we fully understand the sample version and class definition of Java application program. Consequently users are not required to develop remote control program on cellular phone, as they can easily and directly get remote MIDlet procedures of cellular phone by analyzing program code through interface generator. In algorithm 2-1, MIDlet procedure inherits Form class and CommandListener in order to control the entire mobile interface. Line 1 and 2 take charge of loading relative groupware, such as Button, Radio Button, and ComboBox, and analyzes the kinds of operation projects used in application program. At line 4, it would then generate a main list which belongs to cellular phone interface, such as Button, Radio Button, and ComboBox, producing sub-project on cellular phone screen at line 5. Finally, we need HTTP connection procedure to send remote control commands from cellular phone to GUI system, which is performed at line 6.

5. Demonstration

5.1 Integration of GIB and speech recognizer

5.1.1 Running an Application

The operation steps for executing application under GIB system are described as the follows and are labeled in figure 5-1:

1. choosing target language for English
2. selecting an application program
3. pressing button "run" to execute program selected under developing environment
4. selecting pre-defined stage file

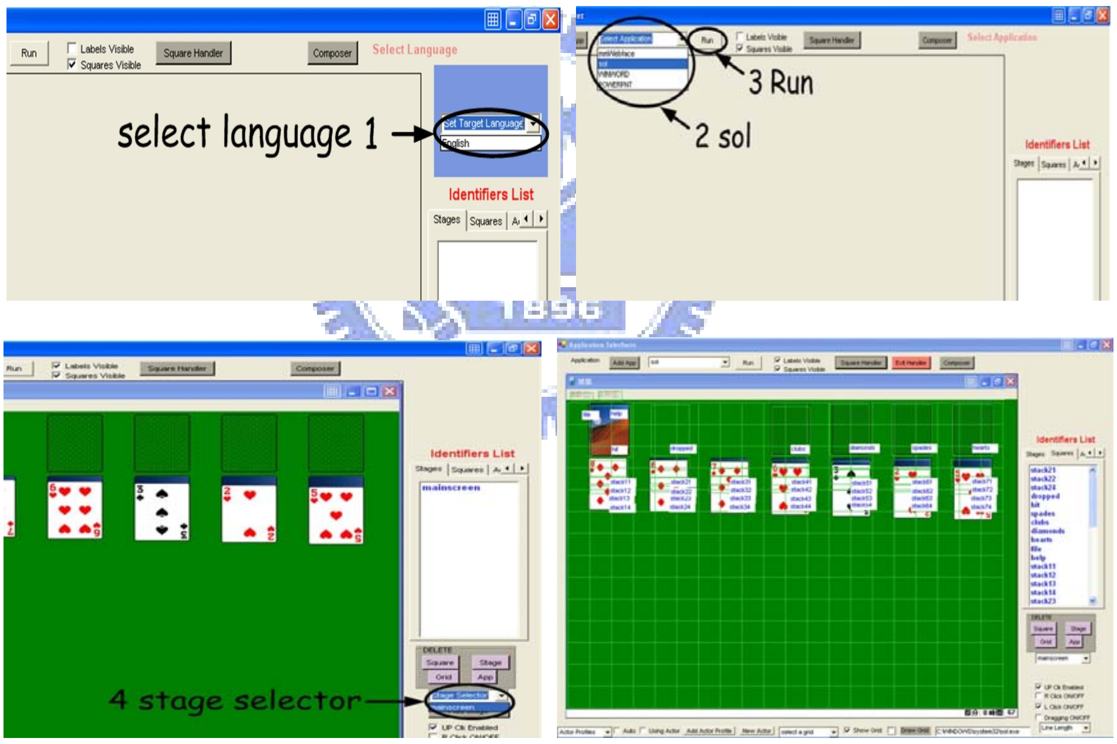


Figure 5-1 Operation steps of running application

5.1.2 Registering an application

This developing system allows choosing multiple applications running under this interface environment, and for convenience, user may load the desired applications into our proposed system. The operation steps of registering application into interface bridge system are as described in the following and labeled in figure 5-2:

1. pressing button "Add App" to open folder
2. selecting a file to register
3. pressing the button "開啓" to confirm the file selection
4. giving a relative name to the chosen application program
5. pressing the button "submit" to submit selected file into system

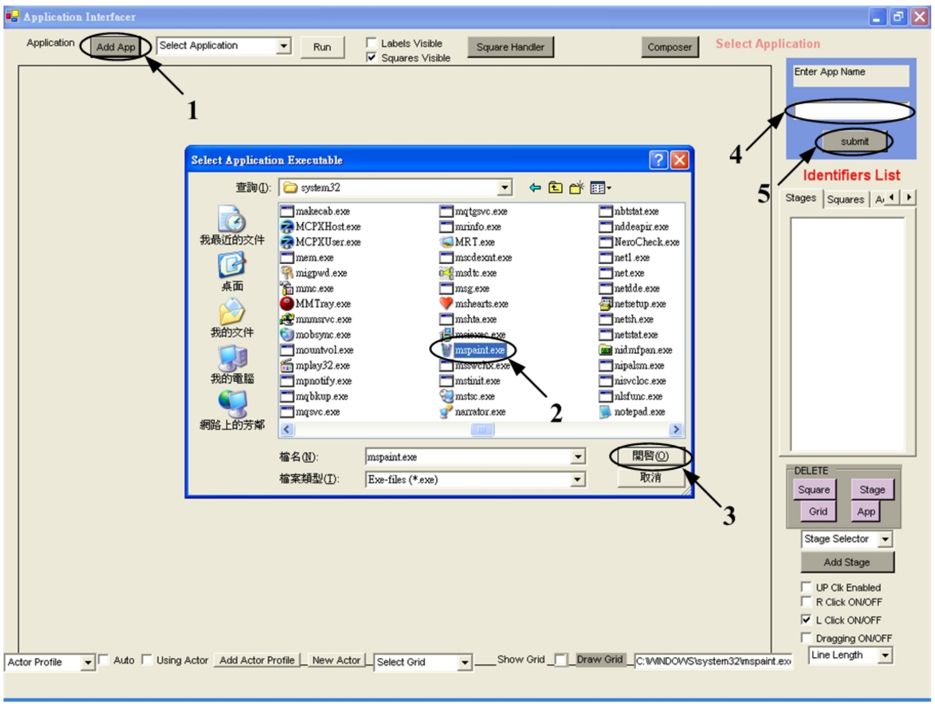


Figure 5-2 Operation steps of registering an application

5.1.3 Registering a square object

To easily move the mouse cursor to the desired position in order to perform a button pressed action, we may draw square objects as the reference zones of application. The operation steps of registering a square object are as the follows and labeled in figure 5-3.

1. Click the button "square handler"

2. Move mouse cursor to label2, then drag mouse cursor to label3.

   - Store position label2 as coordinate1, (x1, y1)

3. Stop and drop mouse cursor when mouse cursor has arrived to label3

   - Store position label3 as coordinate2, (x2, y2)

4. Give drawn square object a name "elephant"

   - Give square object a name from coordinate1 to coordinate2

5. Click the button "submit"

   a. Repeat step 2 until square objects setting is done.

   b. Go to step 6

6. Click the button "exit handler" to complete the objects setting steps

After all steps are completed, system will draw a square from coordinate (x1, y1) to (x2, y1) to (x2, y2) to (x1, y2) and back to (x1, y1), then store coordinate (x, y) as square object control point, where x = x1 + (x2 - x1) / 2, y = y1 + (y2 - y1) / 2.

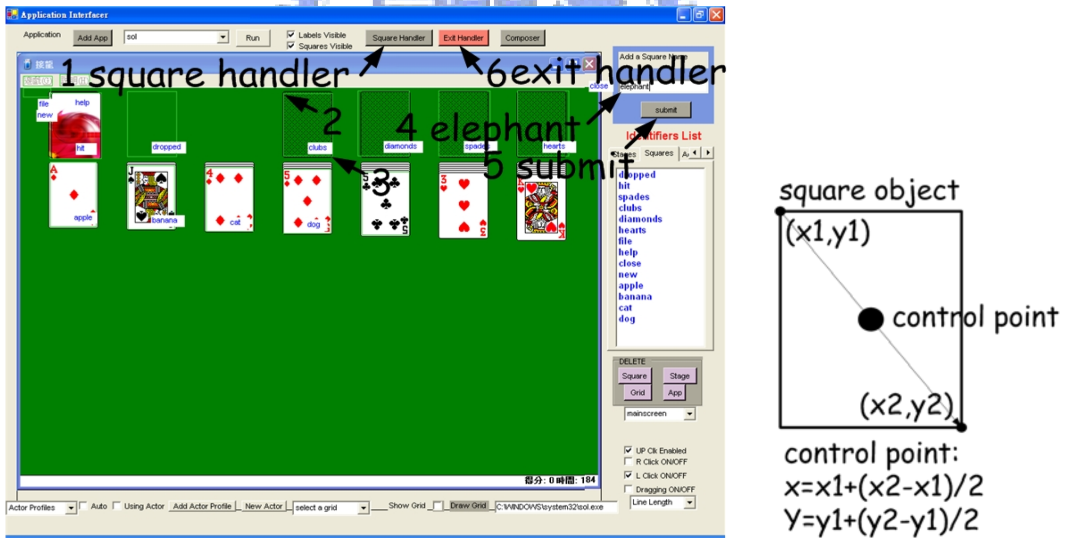


Figure 5-3 Operation steps of registering a square

5.1.4 Registering grids

In section 5.1.3, we can register a square object one step at a time. For convenience, we draw a big square on the Window screen, the system will automatically cut it into many small grids, and each grid has a coordinate name (x, y) given by the system. The operation steps of registering grids are as described in the following and labeled in figure 5-4.

1. Click the button “draw grid”

2. Give grids object a name “position”

3. Click the button “submit” to submit the name of drawn grids into system

4. Click the button “square handler”

5. Moving mouse cursor to label5, and then drag mouse cursor to label6.

   - Store position label5 as coordinate1, (x1, y1)

6. Stop and drop mouse cursor when mouse cursor has arrived to label6

   - Store position label6 as coordinate2, (x2, y2)

7. Click the button “exit handler” to complete the action steps

After square drawn, system will draw a big square from coordinate (x1, y1) to (x2, y1) to (x2, y2) to (x1, y2) and back to (x1, y1), cut this big square into many small square grids, and the area is pre-defined in system as 40 * 40 pixels. Finally, system would set each grid a name (x, y) according to their coordinate, such as (2, 1).

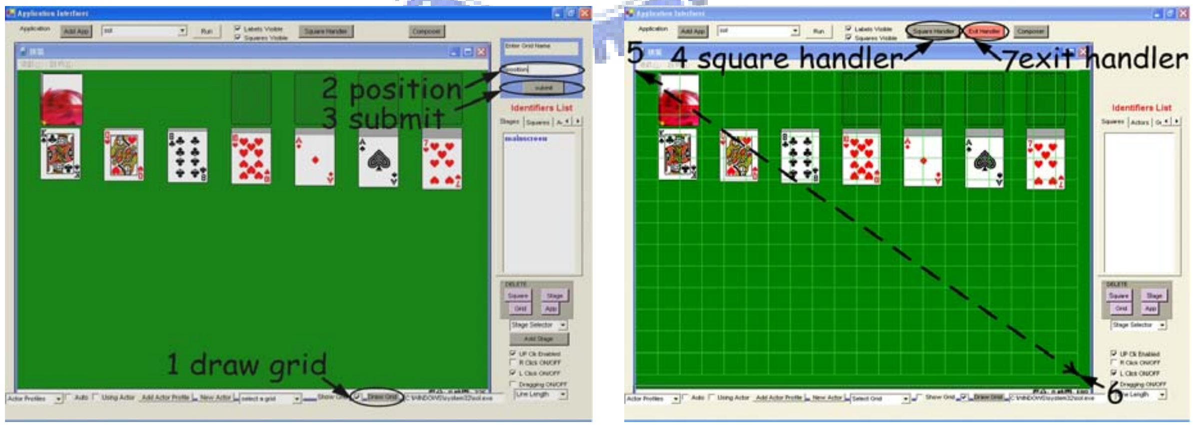


Figure 5-4 Operation steps of registering grids

5.1.5 Registering a stage

Application may have many control layers on the screen. When we draw and set many squares and grids on the application screen, and for an easy usage next time, we could store these setting into a stage which can be named by the user. The operation steps of registering a stage are as demonstrated in the following and labeled in figure 5-5.

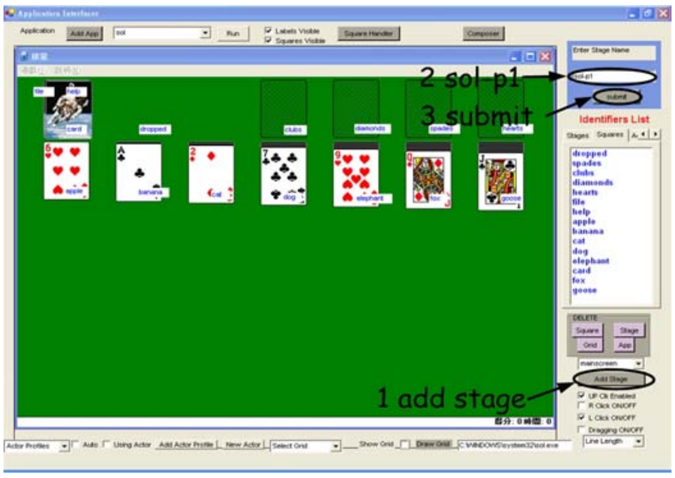


Figure 5-5 Operation steps of registering a stage

1. Click the button "add stage"
2. Give setting environment as a stage named "sol-p1"
3. Click the button "submit"

    - Store setting stage named "sol-p1"

5.1.6 Registering a macro command

For easy and convenient interaction with interface bridge system, we design macro commands procedure in the specification program. The operation steps of registering a macro commands are demonstrated in the following and labeled in figure 5-6.

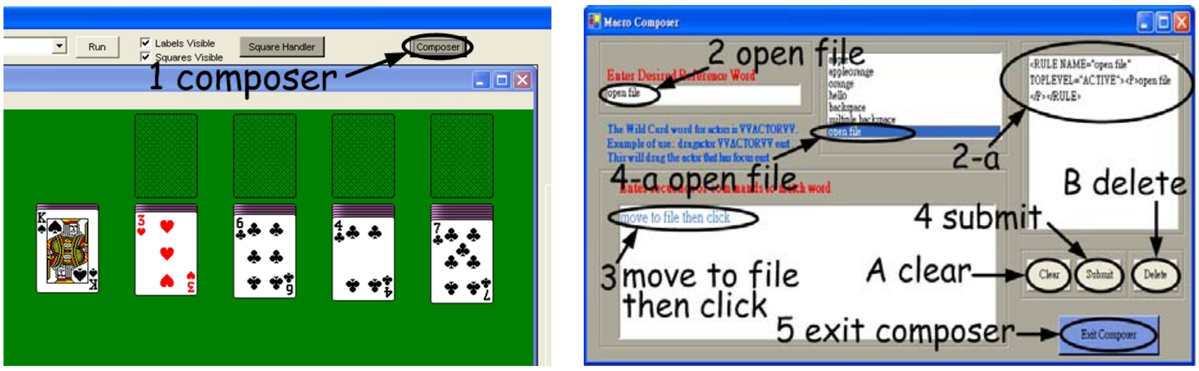


Figure 5-6 Operation steps of registering a macro command

1. Click the button “composer”

2. Give composed commands a macro named “open file”

    - Macro name would be translated into system format simultaneously

3. Key in related command stream “move to file then click”

4. Click the button “submit” to submit defined macro command into system

    a. Store macro into database, repeat step 2 till macro setting is done.

    b. Go to step 6

5. Click the button “exit composer” to exit macro defined system

A. Clear

    - Clear un-submit information

B. Delete

    - Delete the chosen macro command

When a macro command is submitted, the Macro Command Composer stores this structure into the speech grammar definition. After the users spoke a command, the human-interface will recognize and sent these to the macro interpreter to translate and get relative commands.

5.1.7 Examples of application “Sol”

In the past, we could not operate the application “sol.exe” with speech commands, because it was not equipped with speech recognition control ability. Under our GIB system, we pre-defined some square objects through a simple way as described from section 5.1.3 to section 5.1.6, which named “apple,” “banana,” “cat,” “dog,” “elephant,” “fox,” “goose,” “clubs,” “spades,” “hearts,” “diamonds,” “hit,” “dropped,” “file,” “new,” “help,” “close,” accordingly as labeled in figure 5-7, while some sample macro commands are described in table 5-1 below.

Table 5-1 sample macro command of control application “Sol”

Macro commands	**Relative commands**
dealing	move to hit then click by leftclick
apple to banana	dragsquare apple to banana
banana to apple	dragsquare banana to apple
open apple	move to apple then click by leftclick
open banana	move to banana then click by leftclick
apple doubleclick	move to apple then click by doubleclick
banana doubleclick	move to banana then click by doubleclick
clubs to diamonds	dragsquare clubs to diamonds
no	sendkey n
yes	sendkey y
File	move to file then click
close game	move to close then click by leftclick
new game	move to file then click by leftclick then
:	:

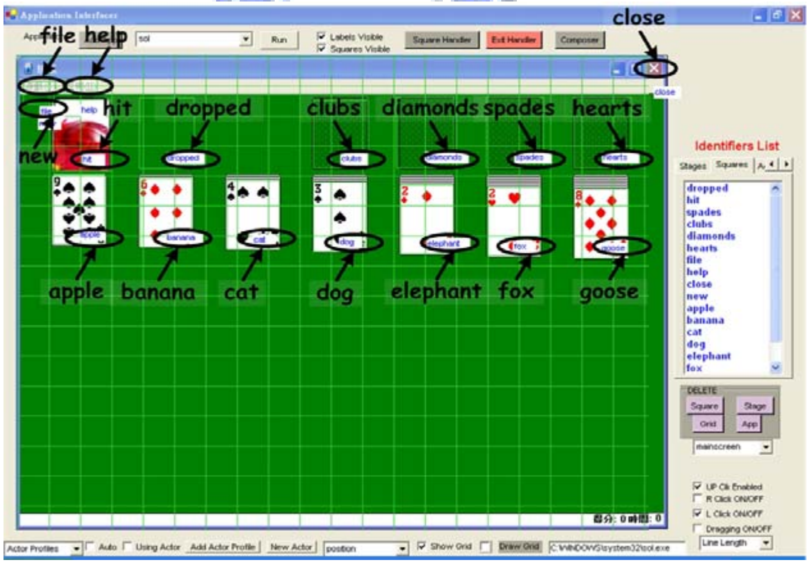


Figure 5-7 Defined control objects of application “sol”

After a simple defining and setting of the control environment of application, we will make a game with speech recognizing ability and operate this application with speech commands easily. First, we speak a macro command, “elephant to dog,” the composed command defined in table 5-1 is “dragsquare elephant to dog,” following this procedure, we speak command “send” to send the macro command spoken earlier to the system. After that, the system receives and recognizes the macro command, translates it into relative commands as define in table 5-1, then allows the system to analyze it to delegate API events if the command is valid. It will first move the mouse cursor to the location of square object named “elephant,” and then drags the object “elephant” and moves it to the location of square object named “dog,” after those procedures, then drops the object “elephant” at the location of object “dog,” as shown in figure 5-8. If there are some wrong or unwanted commands in the command console, we may use speech command “clear console” to clear it.

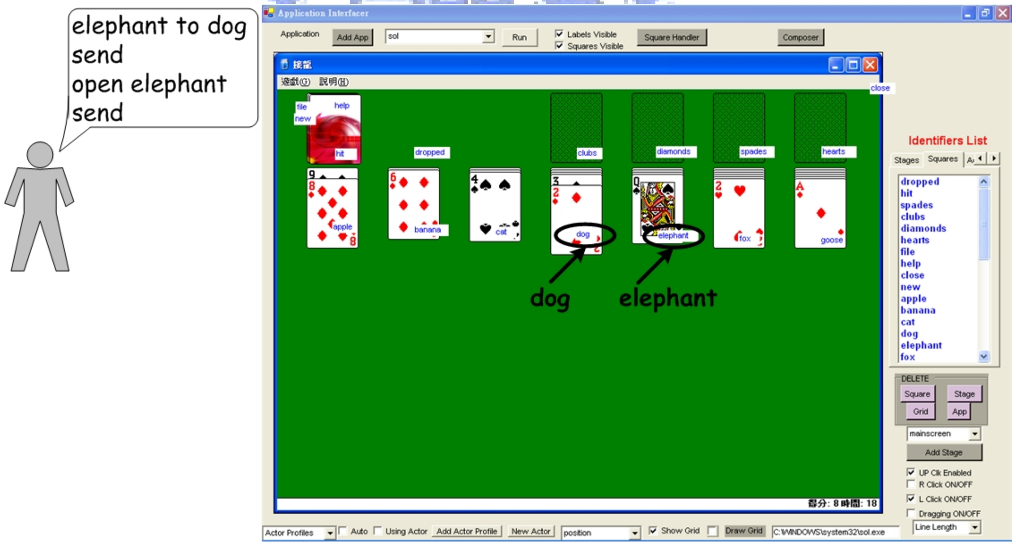


Figure 5-8 Operation in application “sol”

Finally, if the game is over or we want to play a new game, we may speak a macro command “new game,” and the composed command defined in table 5-1 is “move to file then click by leftclick then move to new then click by leftclick”. Following that, we speak command “send” to send the macro command spoken earlier to the system. After that, the system recognizes and analyzes the command, and will delegate API events if the command is valid. The corresponding steps of system process are: first, moving the mouse cursor to the location of square object named “file”. Second, click mouse left button. Third, move the mouse cursor to the location of square object named “new”. Finally, click mouse left button to replay a new game.

5.2 Separate GIB and Application Windows

5.2.1 Application running

The operation steps of executing the application separated from GIB system are described in the following and shown in figure 5-9:

1. running GIB system platform
2. executing defined macro commands interface
3. defining steps of macro commands

A. end the execution for GIB system

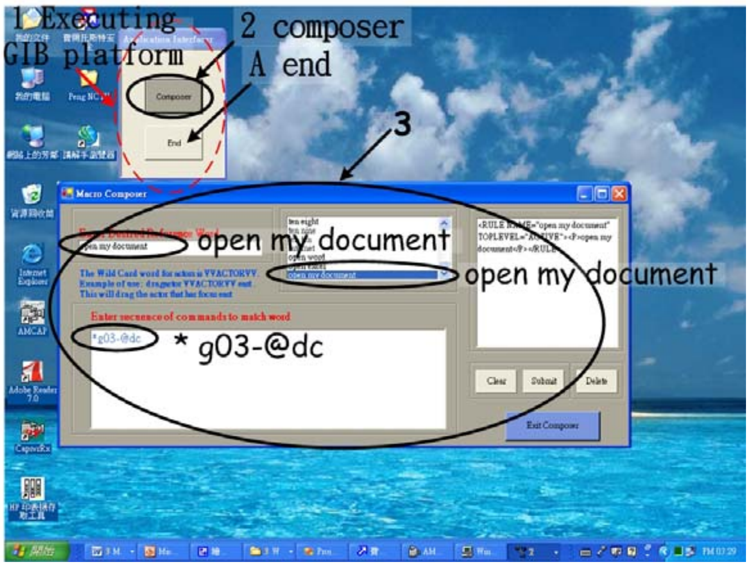


Figure 5-9 Set command stream “open my document” in macro designed folder

5.2.2 Setting spring distance of mouse cursor

For a more flexible way to control mouse cursor spring to right, left, up or down with regular distance, we can use speech commands “h previous” and “h next” to define this regular distance of spring right and left, and use commands “v previous” and “v next” to define this regular distance of spring up and down. First, we use speech command “h previous” to obtain the reference coordinate (x1, y1) of current mouse cursor position, and move mouse cursor to another position. Second, we use speech command “h next” to get the reference coordinate (x2, y2) of current mouse cursor position. We can use the same operating procedures with speech commands “v previous” and “v next” to obtain the reference coordinates (x3, y3) and (x4, y4) of mouse cursor position moving to the up and down. The operating steps are shown in figures 5-10 and 5-11.

After these operations, the spring distance of mouse cursor springing to the right or left would be Δx, and the spring distance of mouse cursor springing to the up or down would be Δy, as described below:

Horizontal spring distance: $\Delta x = | x2 - x1 |$,

Vertical spring distance: $\Delta y = | y3 - y4 |$.

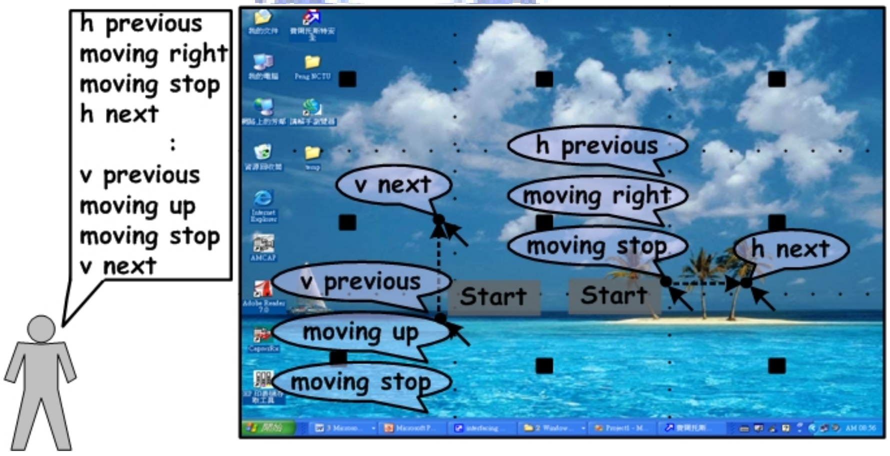


Figure 5-10 mouse cursor one-step moving distance setting

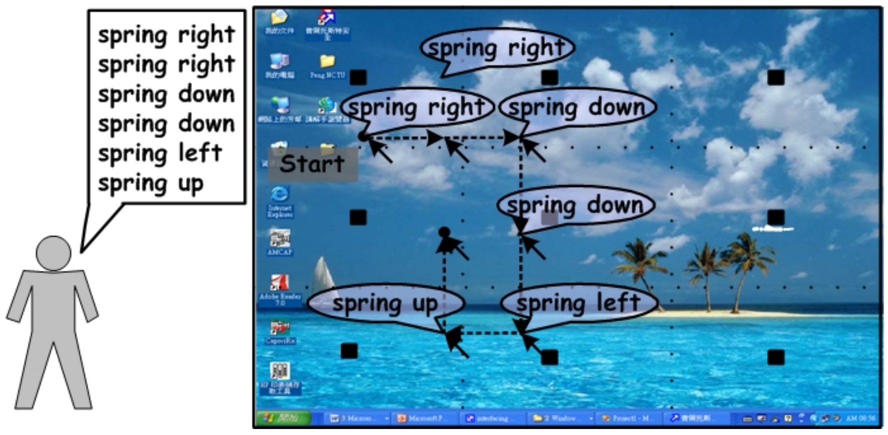


Figure 5-11 mouse cursor one-step moving control

5.2.3 Jumping and moving control of mouse cursor

For a more convenient way to control mouse cursor moving next to a wanted position, we can use speech commands “left up,” “left down,” “right up,” “right down,” “center up,” “center down,” “center left,” “center right,” and “center axis” to control mouse cursor jumping to the pre-defined grids position, as shown in figure 5-12.

To control mouse cursor more precisely moving on the top of control object, we use speech commands “moving left,” “moving right,” “moving up,” and “moving down” to control mouse cursor’s moving direction, and with commands “low speed,” “normal speed,” and “high speed” to control mouse cursor’s moving speed, as shown in figure 5-13.

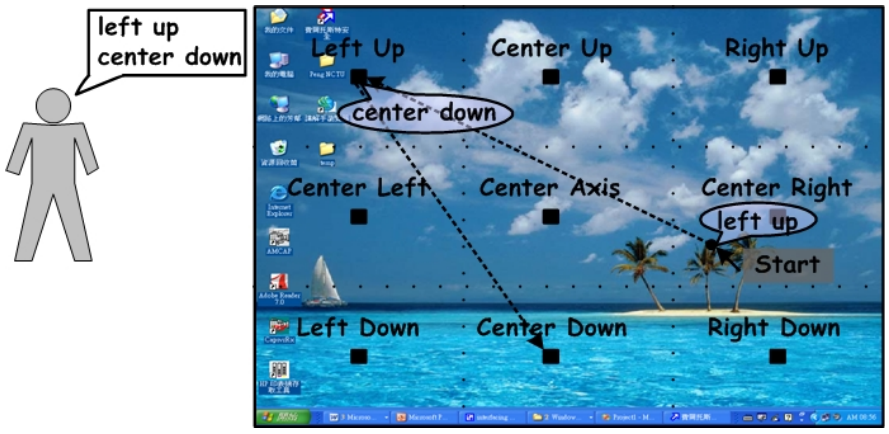


Figure 5-12 mouse cursor jumping control

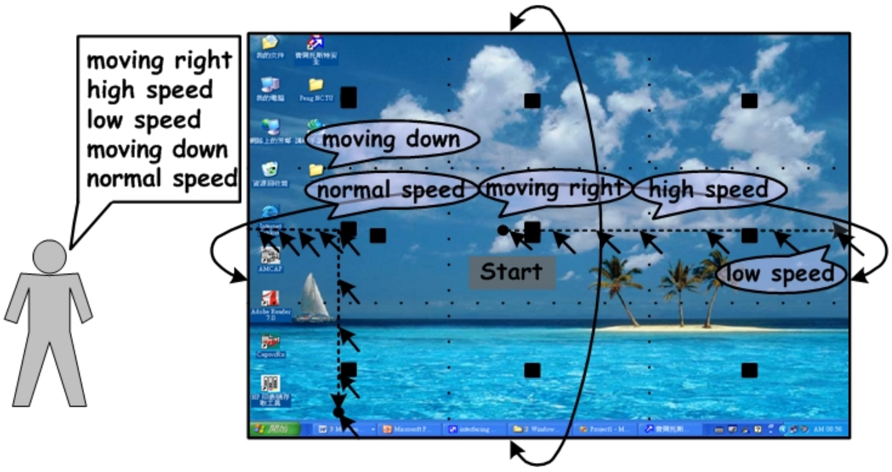


Figure 5-13 mouse cursor moving control

After these operations are completed, if the current mouse cursor position is (x1, y1) and the second position after mouse cursor moving is (x2, y2), then the parameters of mouse cursor moving with different direction and speed become:

Moving right: $x2 = x1 + \Delta x$, $y2 = y1$, $\Delta x > 0$,

Moving left: $x2 = x1 - \Delta x$, $y2 = y1$, $\Delta x > 0$,

Moving up: $x2 = x1$, $y2 = y1 - \Delta y$, $\Delta y > 0$,

Moving down: $x2 = x1$, $y2 = y1 + \Delta y$, $\Delta y > 0$,

High speed: $\Delta x=\Delta x1$,

Normal speed: $\Delta x=\Delta x2$,

Low speed: $\Delta x=\Delta x3$; in that, $\Delta x1>\Delta x2>\Delta x3$

5.2.4 Example of application “Wmplayer”

In Figure 5-14 (a) - (d), we can use the macro commands to control application “media player”. The operation steps are:

1. Users can control mouse cursor moving to the top of component object by speech commands “moving up,” “moving down,” “moving right,” “moving left” and “moving stop,” controlling mouse cursor’s moving direction; and use “low speed,” “normal speed” and “high

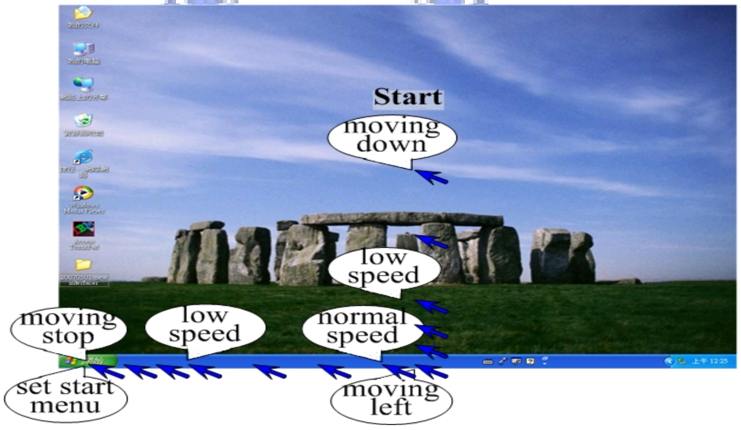
speed" to control mouse cursor moving speed.

2. Using command "set start menu" to set position of current mouse cursor with a name "start menu".

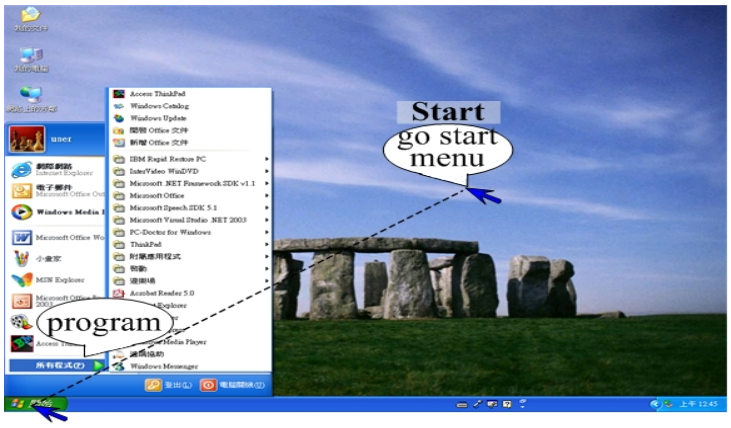
3. Using "go start menu" to move mouse cursor jumping to the stored position named "start menu".

4. We can also do the work and get the same result with a macro command "start," which is equivalent to the interface command "#as" and submitted as a short command $\boxed{\text{Alt}}\boxed{\text{s}}$.

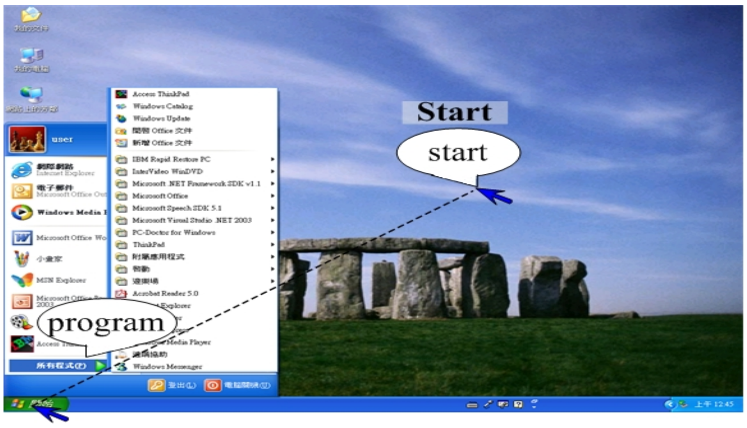
5. Finally, using commands "up," "down," "right," "left," "enter" to manipulate the 2nd or multi-layer button commands, we can run the application system "Media Player" simply with speech commands. The corresponding steps are listed below:



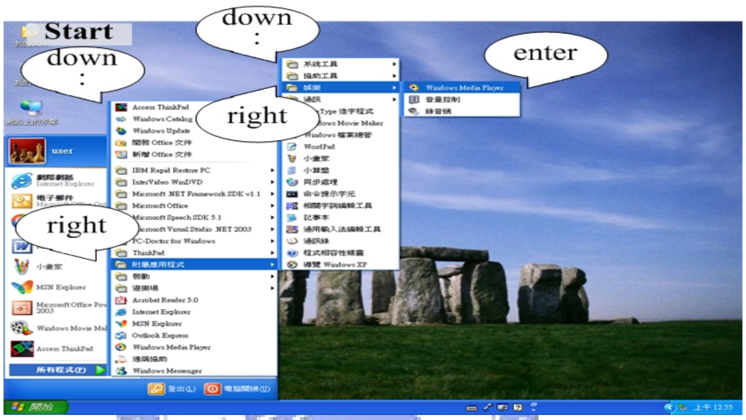
(a) Controlling cursor to move to "start button" and then set it a named "start menu."



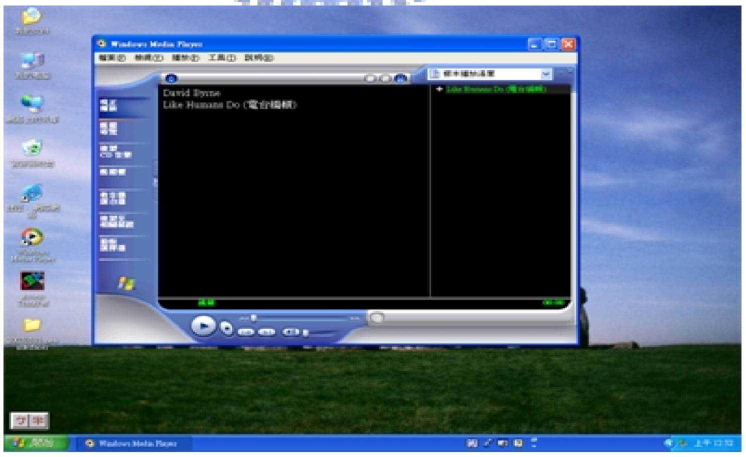
(b) Using "go start menu" command to move cursor to "start button" and click it



(c) Using macro command "start"="#as" to move cursor to "start button"



(d) Control ←, ↑, ↓, → and ↵ by commands "left," "up," "down," "right" and "enter"

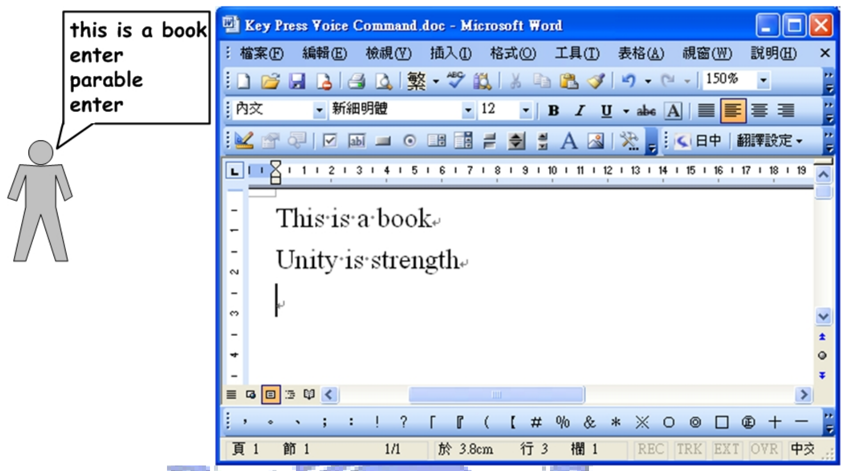


(e) Running application system "Media Player"

Figure 5-14 Executing steps of application "Media Player"

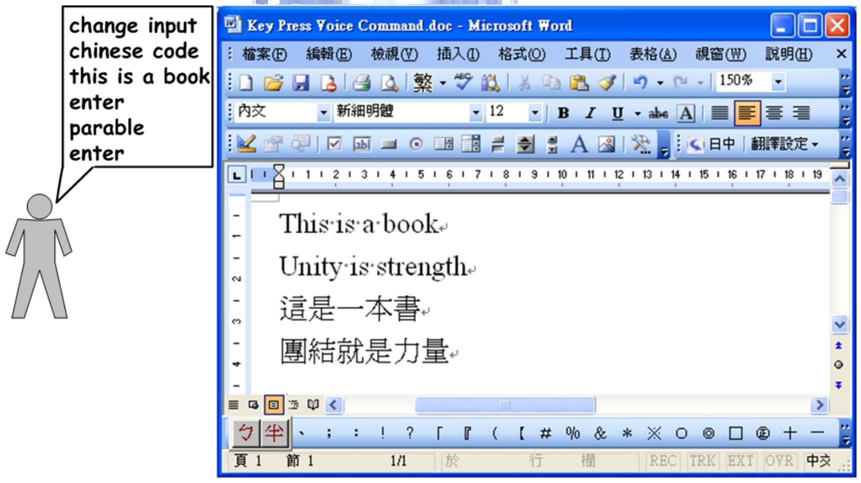
5.2.5 Example of application "Word"

In Figure 5-15 (a), (b), we can use macro command “this is a book” and “enter” to write alphabet, word, and sentence in “WORD” system. Furthermore, through speech commands, such as “change input,” interface code #ci; “English code,” interface code #ie; “Chinese code,” interface code #ic, we switch the environment input method and we will get different output results in “this is a book” in English input method and “這是一本書” in Chinese input method individually through using the same speech command “This is a book”.



This is a book
Unity is strength

(a) Writing word “This is a book” and “parable” by speech command



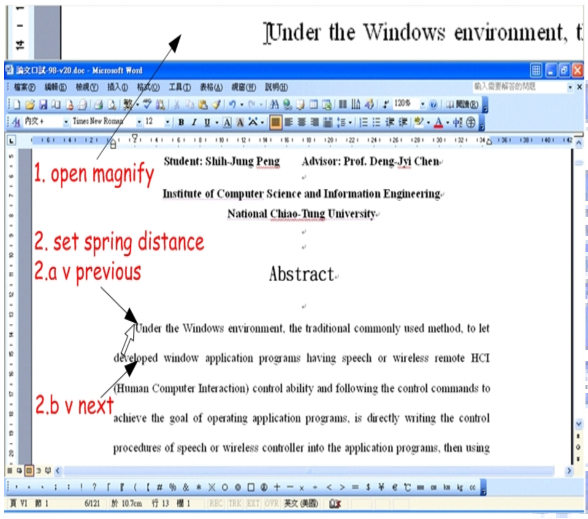
This is a book
Unity is strength
這是一本書
團結就是力量

(b) Change input code by commands “change input” and “chinese code”

Figure 5-15 Writing English or Chinese alphabet with speech commands

Example of document reading

In Figure 5-16 (a), (b), (c), we can use GIB system to facilitate people with reading more easily and friendly. Step 1, we can use macro speech command “open magnify” to execute application “magnify.exe” which will magnify the text under the mouse cursor. Step 2, using speech commands “v previous” and “v next,” users can set the vertical spring distance of mouse cursor as shown in figure 5-16 (a). Step 3, we can use command “moving right” to control mouse cursor’s moving direction to the right continuously as shown in figure 5-16 (b). Step 4, we can use command “spring down” to control mouse cursor jumping to the next line according to the distance set in step 2 as shown in figure 5-16 (c).



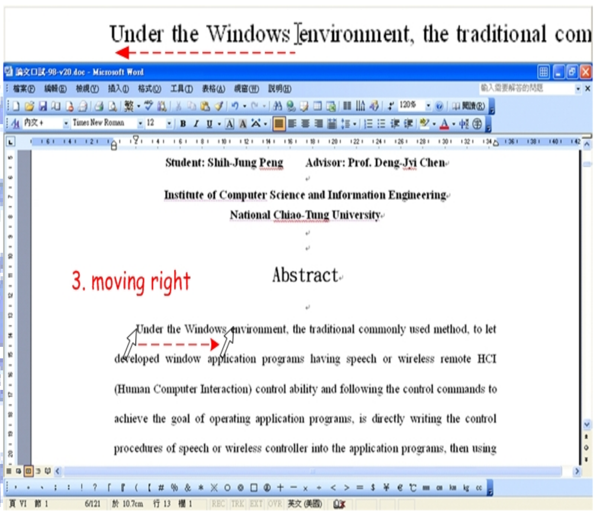
Student: Shih-Jung Peng Advisor: Prof. Deng-Jyi Chen

Institute of Computer Science and Information Engineering

National Chiao-Tung University

Abstract

Under the Windows environment, the traditional commonly used method, to let developed window application programs having speech or wireless remote HCI (Human Computer Interaction) control ability and following the control commands to achieve the goal of operating application programs, is directly writing the control procedures of speech or wireless controller into the application programs, then using



Student: Shih-Jung Peng Advisor: Prof. Deng-Jyi Chen

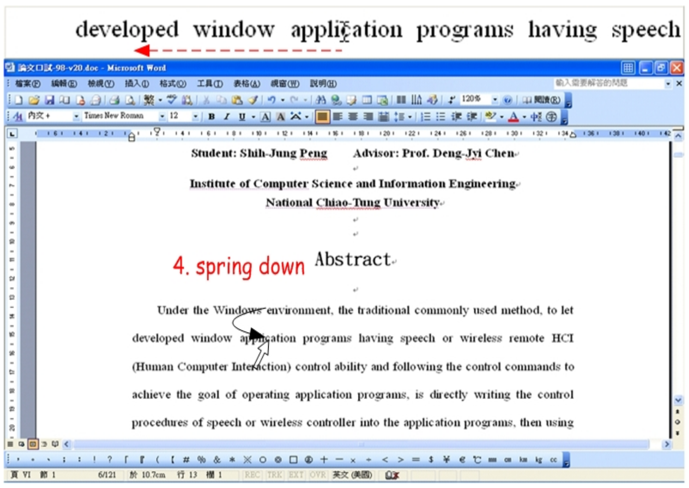
Institute of Computer Science and Information Engineering

National Chiao-Tung University

Abstract

Under the Windows environment, the traditional commonly used method, to let developed window application programs having speech or wireless remote HCI (Human Computer Interaction) control ability and following the control commands to achieve the goal of operating application programs, is directly writing the control procedures of speech or wireless controller into the application programs, then using

(a) Execute application “magnify.exe” (b) control mouse cursor's moving to right



Student: Shih-Jung Peng Advisor: Prof. Deng-Jyi Chen

Institute of Computer Science and Information Engineering

National Chiao-Tung University

Abstract

Under the Windows environment, the traditional commonly used method, to let developed window application programs having speech or wireless remote HCI (Human Computer Interaction) control ability and following the control commands to achieve the goal of operating application programs, is directly writing the control procedures of speech or wireless controller into the application programs, then using

(c) Spring mouse cursor to next text line

Figure 5-16 Using speech command to help user reading document

5.3 GAI generation

5.3.1 Example of application "VCard"

If there is a JAVA program "VCard" running on the PC environment, we can use mouse and keyboard to control the operating functions of this program; the main function of this program is to edit the name card format with different background, logo, and card format. And the operating procedures are as labeled in figure 5-17.

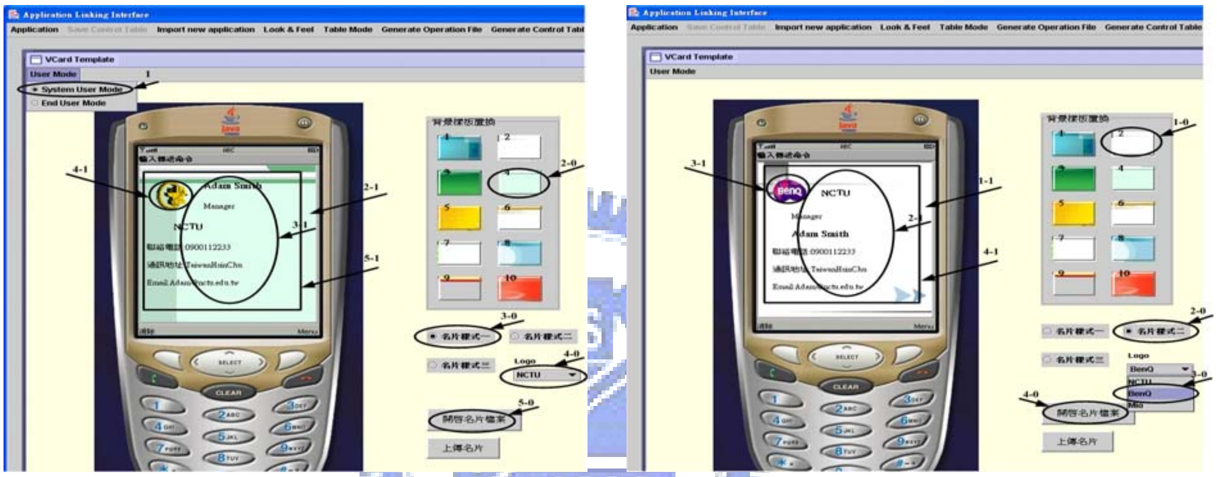


Figure 5-17 Executing application program "VCard"

In this case, we intend to use a cellular phone which has GUI function generated automatically by proposed interface generator system, as a remote controller to control the JAVA AP on the PC environment through HTTP wireless network. The interface generator procedures are separated into multiple steps as depicted in figure 5-18, 5-19, and 5-20:

Step 1: Loading Target Application Description

The first step of loading a JAVA application into an interface generator system is to choose a desired application for the proposed system by file selection menu. After that, the target application can be executed under the interface generator environment and placed on top of the application's GUI, which is labeled as step 1 and 2 in figure 5-18.

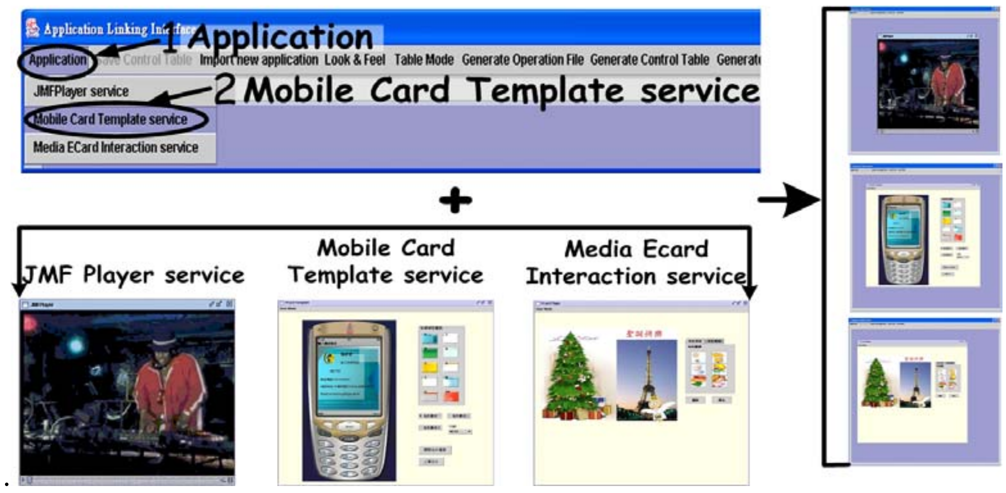
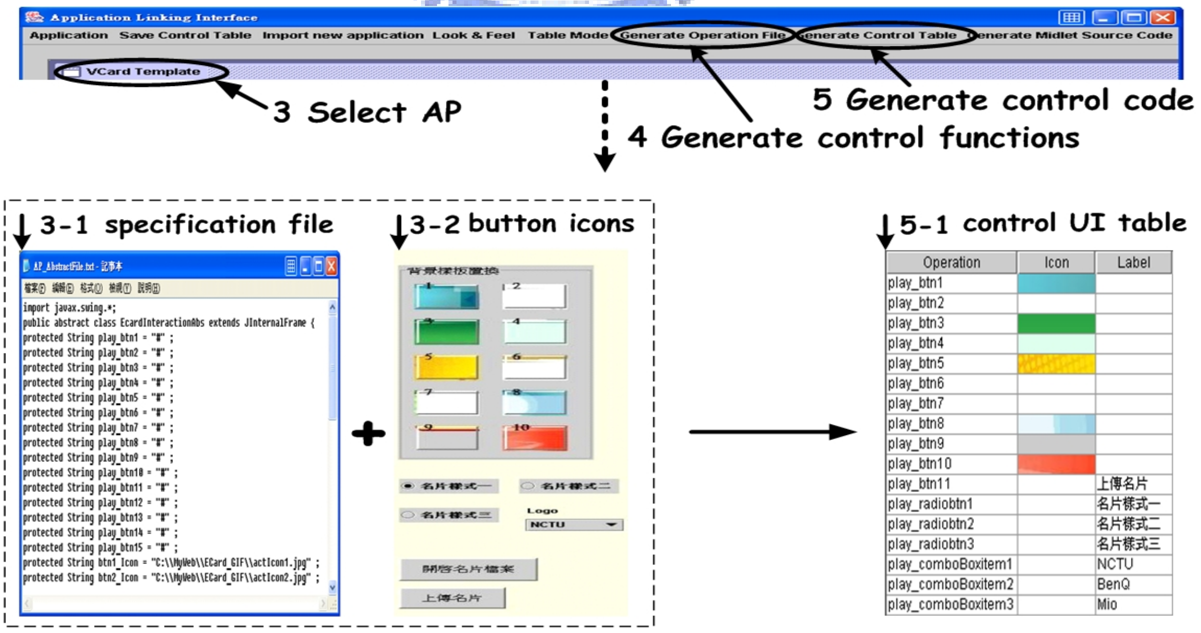

Application Linking Interface
1 Application
Application
Save Control Table
Import new application
Look & Feel
Table Mode
Generate Operation File
Generate Control Table
JMFPlayer service
2 Mobile Card Template service
Mobile Card Template service
Media ECard Interaction service
JMF Player service
Mobile Card Template service
Media Ecard Interaction service

Figure 5-18 Loading and running JAVA AP on the system

Step 2: Generate JAVA Midlet file

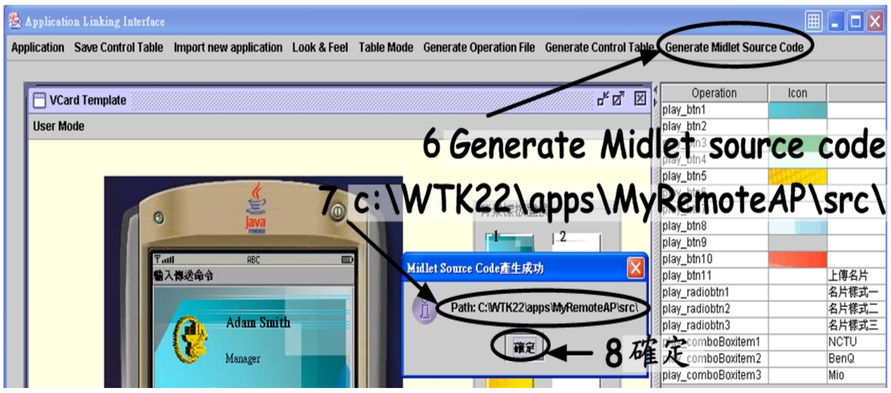
The second step is to automatically generate Midlet program of control interface from the control button of chosen JAVA application into cellular phone. After loading the application into system, interface designer can just press command button "generate operation file," as step 4 of figure 5-19, and the system would create operation file of application.



```
import javax.swing.*;
public abstract class EcardInteractionAbs extends JInternalFrame {
protected String play_btn1 = "#" ;
protected String play_btn2 = "#" ;
protected String play_btn3 = "#" ;
protected String play_btn4 = "#" ;
protected String play_btn5 = "#" ;
protected String play_btn6 = "#" ;
protected String play_btn7 = "#" ;
protected String play_btn8 = "#" ;
protected String play_btn9 = "#" ;
protected String play_btn10 = "#" ;
protected String play_btn11 = "#" ;
protected String play_btn12 = "#" ;
protected String play_btn13 = "#" ;
protected String play_btn14 = "#" ;
protected String play_btn15 = "#" ;
protected String btn1_Icon = "C:\\MyWeb\\ECard_GIF\\actIcon1.jpg" ;
protected String btn2_Icon = "C:\\MyWeb\\ECard_GIF\\actIcon2.jpg" ;
```

Operation	Icon	Label
play_btn1	🟦	
play_btn2	⬜	
play_btn3	🟩	
play_btn4	⬜	
play_btn5	🟨	
play_btn6	⬜	
play_btn7	⬜	
play_btn8	🟦	
play_btn9	⬜	
play_btn10	🟥	
play_btn11		上傳名片
play_radiobtn1		名片樣式一
play_radiobtn2		名片樣式二
play_radiobtn3		名片樣式三
play_comboBoxitem1		NCTU
play_comboBoxitem2		BenQ
play_comboBoxitem3		Mio

Figure 5-19 Flow of generating control table of AP "VCard"


Application Linking Interface
Application
Save Control Table
Import new application
Look & Feel
Table Mode
Generate Operation File
Generate Control Table
Generate Midlet Source Code
VCard Template
User Mode
6 Generate Midlet source code
7 c:\WTK22\apps\MyRemoteAP\src\
Midlet Source Code產生成功
Path: C:\WTK22\apps\MyRemoteAP\src\
確定
8 確定
Adam Smith
Manager

Operation	Icon	
play_btn1	🟦	
play_btn2		
play_btn3	🟩	
play_btn4		
play_btn5	🟨	
play_btn6		
play_btn7		
play_btn8	🟦	
play_btn9	▨	
play_btn10	🟥	
play_btn11		上傳名片
play_radiobtn1		名片樣式一
play_radiobtn2		名片樣式二
play_radiobtn3		名片樣式三
play_comboBoxitem1		NCTU
play_comboBoxitem2		BenQ
play_comboBoxitem3		Mio

Figure 5-20 Flow of Generating Midlet code of AP "VCard"

Similarly, designer can press command button "create control table," as step 5 of figure 5-19, and the system will create a control table of application as step 5-1 of figure 5-19. Finally, designers press command button "generate Midlet source code," and the system generates Midlet source code of control interface of application VCard on the cellular phone, as the step 6 - 8 labeled in figure 5-20.

Step 3: Download executed file into cellular phone

After downloading the executed file into cellular phone and run it, the control interface of application VCard on the cellular phone under different objects view such as "Main Menu," "Button," "Radio," and "ComboBox" are shown as in figure 5-21.

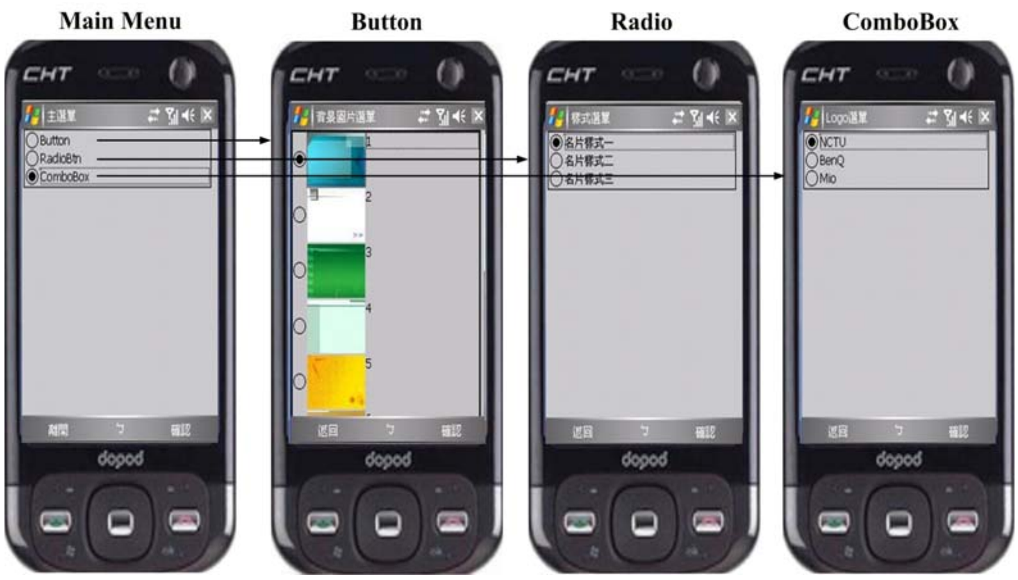


Figure 5-21 Generated control interface of "VCard" to cellular phon

Step 4: Control AP program on the PC with cellular phone

Finally, this cellular phone would become a remote controller; we can use it to remotely control the application program running on the PC through HTTP wireless network by directly touching the command objects, as we have seen, on the control interface of cellular phone, while we need not to memorize complex compound commands or complex operating procedures.

6. Conclusions and Future work

6.1 Conclusion

In this research we have overcome some obstacles and developed a visual Generic interface bridge (GIB) to make applications without prior recognition control ability to have recognition control ability of speech or cellular phone by employing several techniques such as the “Touch-Panel Interface,” object oriented design pattern, and incorporated into a script language definition together with a parsing technique through a simple and easy way without the need to write any program code.

This research includes two parts: first, “Integration GIB with HCI” with the use of speech to control applications on the PC, second, “GIB-based Application Interface (GAI) generation” with the use of cellular phone to control applications running on the PC through internet. In part one, we propose a flexible and effective visual platform as an interface for the target application by defining reference positions and giving each position a name on a virtual transparent interface environment. Once the position objects are defined on the Window screen and the macro commands are set completely, users can manipulate the application simply with speech commands through processing input commands and calling the mouse and keyboard API events according to the analyzed result. In part two, with the same analyzing concepts of GIB, we develop an interface generation system to bridge a wireless control system of cellular phone with applications running on PC. This system can automatically generate a human control interface into cellular phone for a controlled PC application through simple objects specification of application. Finally it produces and compiles the Midlet program and wraps the result of jar file into cellular phone.

The major contributions of this research include: first, we proposed a visual GIB between HCI and applications under which the users can control applications more easily after defining simple object. Second, the GIB system offers a simple, easy and effective way for users to control applications without the need to write any program code. Third, users can increase, delete or modify the commands of HCI, and give defined object a name during run time without the need to access, write or re-compile any source code of the application. Fourth, under GIB systems, users can easily define speech commands with macro command in order to simplify commands input and increase recognition correctness. Fifth, users can completely control mouse and keyboard combined actions under proposed GIB system.

6.2 Future work

The future work in this research includes:

1. **special education application**

Not all users are convenient in motions such as hand, body, or eyes, therefore they cannot easily have access to the computer and acquire acknowledge. The major complexity of special education applications is the design of user control interface. Under GIB system, we may consider developing some particular manipulating actions or control signals for those who have inconvenience in motions, so that the most common applications may also be manipulated by them easily.

2. **multi touch technique**

Traditional touch panel is a commonplace for single point of contact. Multi-touch system allows user to interact with applications with more than one finger at a time. Thus, we may consider integrating GIB system with multi touch techniques to change the way we interact with computer and make the manipulation more natural and vivid.

Reference

[1] Speech-Actuated Mainpulator.

Available: "http://www.research.att.com/history/89robot.html"

[2] VSpeech 4.0, BK02 product. Available: "http://www.bk02.net/vspeech/index2.htm"

[3] Voxx 5.0 Speech recognition project, Sourceforge product,

Available: "http://sourceforge.net/projects/voxxopensource/"

[4] IVOS, ComunX product., Available: "http://ivos.comunx.com/"

[5] Microsoft's Speech Recognizer V.5.1, Microsoft product,

Available: "http://www.microsoft.com"

[6] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, Tony D. DeRose, "Toolglass and Magic Lenses: The See-Through Interface," Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304, 1993.

Available: "http://www2.parc.com/istl/projects/MagicLenses/93Siggraph.html"

[7] Yves Boussemart, Francois Rioux, Frank Rudzicz, Michale Wozniewski, Jeremy R. Cooperstock "A Framework for 3D Visualization and Manipulation in an Immersive Space Using an Untethered Bimanual Gestural Interface"; Centre For Intelligent Machines 3480 University Street Montreal, Quebec, Canada, 2004.

Available: "http://www.cim.mcgill.ca/sre/publications/vrst.pdf"

[8] W. LI, W. Wang, I. Marsic, "Collaboration Transparency in the DISCIPLE Frame Work"; In Proceedings of the ACM International Conference on Supporting Group Work (GROUP'99) November 14-17, 1999, Phoenix, AZ.

[9] BestWise International Computing Company. Available: "http://www.caidiy.com.tw"

[10] Christofer R. Wren, Carson J. Reynolds, "Parsimony & Transparency in Ubiquitous Interface Design"; Media Laboratory, Massachusetts Institute of Technology.

Available: "http://affect.media.mit.edu/pdfs/02.wren-reynolds-abstract.pdf"

[11] Robert W. Sebesta, "Concepts of Programming Languages," 7th Edition, Addison-Wesley Publishing Company, 2002.

[12] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, "The Java Language Specification," Third Edition, Sun Microsystems, Inc., 2005.

[13] C.S. Koong, J.S Tyan, S.F. Chuang, D.J. Chen, "A Component-Based Visual Scenario Construction Language for Electronic Book," IEEE COMPSAC 2000, the 24th Annual International Computer Software and Applications Conference, Taipei, Taiwan, October 25-27, 2000, pp. 255-260.C.S.

[14] Online Laborlawtalk Encyclopedia. Available: "http://encyclopedia.laborlawtalk.com/"

[15] WinBatch Macro Scripting Language. Available: "http://www.winbatch.com/"

[16] N. Manasse, "Speech Recognition"; *University of Nebraska, Lincoln, 1999.*

[17] Microsoft Speech SDK, Version 5.1 Documentation, *Microsoft Corporati. 2001.* Available: "http://download.microsoft.com/download/speechSDK/SDK/5.1/ WXP/ EN-US/speechsdk51.exe"

[18] Bruce Powel Douglass, "Real-time Design Patterns: Robust Scalable Architecture for Real-time Systems," Pearson Education, 2003.

[19] Design Patterns in Java. Available: http://www.fluffycat.com/java/patterns.html

[20] Chung-Chien Hwang, "Objected-Oriented Program Behavior Analysis Based on Control Patterns"; PhD. dissertation, Computer Science and Information Engineering, National Chiao-Tung University, Taiwan, 2002.

[21] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen; Object-Oriented Modeling and Design, 1991 Prentice-Hall.

[22] Grady Booch; Object-Oriented Analysis and Design with Applications, the Benjamin/Cummings Publishing Company, Inc., 1994.

[23] Robot Battle Scripting Language Functions.

Available: “http://www.duke.edu/~cc27/RobotBattleCommandManual.html”

[24] Speech-Actuated Manipulator.

Available: “http://www.research.att.com/history/89robot.html”

[25] W. C. Chen, “A Reuse-based Software Construction Paradigm for Visualized Reusable Components and Frameworks”; PhD. dissertation, Computer Science and Information Engineering, National Chiao-Tung University, Taiwan, 1998.

[26] Microsoft’s Windows API Reference Web-Site. Available: “http://www.mentalis.org”

[27] Programming Techniques Reference Forum. Available: “http://www.xtremevbtalk.com/”

[28] S.J. Gibbs, D.C. Tsichritzis; Multimedia Programming, Objects, Environments, and Frameworks, Addison-Wesley Publishing Company, 1995.

[29] OMG’s CORBA Specification, Object Management Group’s Standard.

Available: “http://www.corba.org”

[30] H. Okada, K. Kato, T. Ikegamai, Y. Tatusmi, and T. Asahi, “Proposal of PC Remote Control System by Mobile Devices,” IPSJ Sig Notes (2001-HI-93), 2001(38): 1 6, 2001.

[31] Norman Makoto Su, Yutaka Sakane, Masahiko Tsukamoto, Shojiro, Nishio, “Systems Issues: Rajicon Remote PC GUI Operations via Constricted Mobile Interfaces,” Proceedings of the 8th annual international conference on Mobile computing and networking, September 2002.

[32] Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joseph Hughes, Thomas K. Harris, Roni Rosenfeld, Mathilde Pignol, “Infrastructure for Ubicomp: Generating Remote Control Interfaces for Complex Appliances,” Proceedings of the 15th annual ACM symposium on User interface software and technology, October 2002.

[33] Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S.M., and Shuster, J.E., “ An Appliance-Independent XML User Interface Language,” in the Eighth International World Wide Web Conference, 1999, Toronto, Canada.

[34] De Baar, D.J.M.J., Foley, J.D., Mullet, K.E., “Coupling Application Design and User

Interface Design," in Conference on Human Factors and Computing Systems, 1992 Monterey, California ACM Press, pp. 259-266.

[35] Nichols, J., Myers, B.A., Higgins, M., Hughes, J., Harris, T.K., Rosenfeld, R., Shriver, S., "Requirements for Automatically Generating Multi-Modal Interfaces for Complex Appliances," in ICMI, 2002.

[36] Olufisayo Omojokun, S. Pierce, L. Isbell, Prasun Dewan , "Comparing end-user and Intelligent Remote Control Interface Generation," Personal and Ubiquitous Computing, Volume 10, Issue 2, January 2006.

[37] Neil R. N. Enns, I. Scott MacKenzie, "Touchpad-based Remote Control Devices," Conference on Human Factors in Computing Systems, April 1998.

[38] Azam Khan, George Fitzmaurice, Don Almeida, Nicolas Burtnyk, Gordon Kurtenbach, "A Remote Control Interface for Large Displays," Proceedings of the 17th annual ACM symposium on User interface software and technology, October 2004.

[39] Eddie Schwalb, "Synopsis - Books and Software iTV Handbook: Technologies & Standards," Computers in Entertainment (CIE), Volume 2, Issue 2, April 2004.

[40] Anind K. Dey, Gregory D. Abowd, "Towards a Better Understanding of Context and Context-Awareness," 1999.

[41] Shih-Jung Peng, Jan Karel Ruzicka and Deng-Jyi Chen, "A Generic and Visual Interfacing Framework for Bridging the Interface between Application Systems and Recognizers," Journal of Information Science and Engineering, Vol. 22, No.5, September 2006, pp.1077-1091.

[42] Shih-Jung Peng and Deng-Jyi Chen, "A Generic Interface Methodology for Bridging Application Systems and Speech Recognizers," 2007 International Conference on Information, Communications and Signal Processing, 10-13 December, 2007, in Singapore.

[43] Deng-Jyi Chen, Shih-Jung Peng and Chin-Eng Ong, "Generate Remote Control Interface

Automatically into Cellular Phone for Controlling Applications Running on PC," Journal of Information Science and Engineering, 2008.09.16. Accepted.

[44] Shih-Kun Huang, "Objected-Oriented Program Behavior Analysis Based on Control Patterns," PhD. dissertation, Computer Science and Information Engineering, National Chiao-Tung University, Taiwan, 2002.

[45] Jones J., "DVB/MHP JavaTM Data Transport Mechanisms", Proceedings of the 40th International Conference on Tools Pacific, Objects for internet and embedded applications, Volume 10, 2002, pp. 115-121.

[46] Microsoft Corporation. Universal plug and play forum. Available: "http://www.upnp.org/ "

[47] API specification for the Java2 Platform, Standard Edition, version 1.4.2., Available: "http://java.sun.com/j2se/1.4.2/docs/api/"

[48] Design Patterns in Java. Available: "http://www.fluffycat.com/java/patterns.html"

[49] Java TV API 1.1 (JSR-927): Available: "http://java.sun.com/javame/reference/apis/jsr927/"

[50] Jeffrey Nichols, Brad A. Myers, Brandon Rothrock, "UNIFORM: Automatically Generating Consistent Remote Control User Interfaces," Proceedings of the SIGCHI conference on Human Factors in computing systems, ACM, April 2006, pp. 611-620.

[51] T. Ha, J. Jung, and S. Oh., "Method to Analyze User Behavior in Home Environment," Personal and Ubiquitous Computing, 10(2--3):110--121, 2006.

[52] Jan Hess, Guy Küstermann, Volkmar Pipek, "Premote: a User Customizable Remote Control," CHI '08 extended abstracts on Human factors in computing systems, ACM, April 2008, pp. 3279-3284.

[53] Seong Joon Lee, Yong Hwan Kim, Sung Soo Kim, Kwang Seon Ahn, "A Remote Monitoring and Control of Home Appliances on Ubiquitous Smart Homes," Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications, February 2008.

Appendix

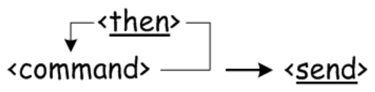
A. BNF of Control Commands

```
<execution_command> ::= <command_string> send | <clear-console> | <undo-phrase> |
                <store-cursor> | <recover-cursor> | <capture-it>
<command_string> ::= <statement> | <statement> then <command_string>
<clear-console> ::= clearConsole
<undo-phrase> ::= undoPhrase
<store-cursor> ::= storeCursor
<recover-cursor> ::= recoverCursor
<capture-it> ::= captureIt
<statement> ::= <set-distance> | <set-dragspeed> | <set-drop> | <show-grid> | <drag> |
                <drag-square> | <drag-coordinate> | <click> | <click-square> | <click-coordinate> |
                <move> | <clear-text> | <send-key> | <loop> | <capture-it> | <select-stage> |
                <select-grid> | <select-application>
<set-distance> ::= setDistance to <distance>
<set-drag-speed> ::= setDragspeed <speed>
<set-drop> ::= setDrop <boolean>
<show-grid> ::= showGrid <boolean>
<drag> ::= drag (<direction> | <pattern>) by <distance>
<drag-square> ::= dragSquare <square> (to (<square> | <coordinate>) | (by <pattern> | <direction>)
                by <distance>
<drag-coordinate> ::= dragCoordinate <coordinate> (to (<square> | <coordinate>) | (by <pattern> |
                <direction>) by <distance>
<click> ::= click by <clicktype>
<click-square> ::= clickSquare <square> by <clicktype>
```

```
<click-coordinate> ::= clickCoordinate <coordinate> by <clicktype>

<move> ::= move (to (<square> | <coordinate>) | <direction> by <distance>)

<clea-Text> ::= clearText

<send-key> ::= sendKey <string>

<loop> ::= loop <number> times

<select-stage> ::= selectStage <string>

<select-grid> ::= selectGrid <string>

<select-application> ::= selectApplication <string>

<square> ::= <string>

<coordinate> ::= <number> , <number>

<number> ::= <digital> | <digital> <number>

<string> ::= <letter> | <digital> | <letter> <string> | <digital> <string>

<click-type> ::= leftclick | rightclick | doubleclick

<distance> ::= veryshort | short | normal | long | verylong

<speed> ::= veryslow | slow | normal | fast | veryfast

<boolean> ::= true | false

<direction> ::= east | west | south | north | northeast | northwest | southeast | southwest

<pattern> ::= triangle | square | pentagon | hexagon | octagon | curves | zigzag | spiral

<digital> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<letter> ::= a | b | c | … | y | z | A | B | C | … | Y | Z
```

B. Partial Syntax Tree of Control Commands

Send:



Then:

statement x — then — statement y ··· send

setDistance:

setDistance — to — ***distance***

setDragSpeed:

setDragSpeed — ***speed***

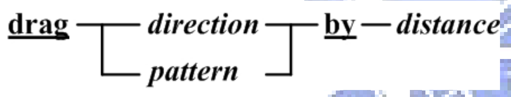
setDrop:

setDrop — ***boolean***

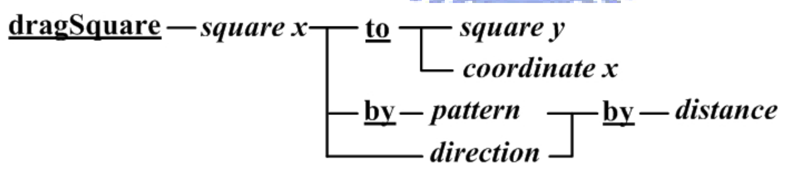
showGrid:

showGrid — ***boolean***

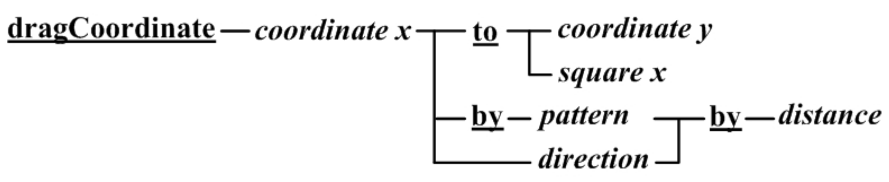
Drag:



dragSquare:



dragCoordinate:



click:

click — by — ***clicktype***

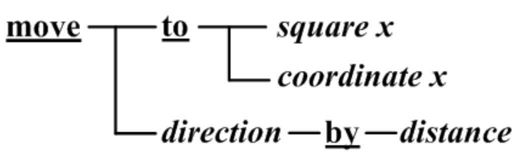
clickSquare:

clickSquare — ***square x*** **— by —** ***clicktype***

clickCoordinate:

clickCoordinate — ***coordinate x*** — **by** — ***clicktype***

Move:



clearText:

clearText – **then** – **loop** – ***number*** – **times**

sendKey:

sendKey — ***text x***

loop:

loop 3 times

selectStage:

selectStage — ***stage x***

selectGrid:

selectGrid — ***grid x***

selectApplication:

selectApplication — ***application x***

Vita

	Shih-Jung Peng
Jan 20 1965	Born in Hsin-Chu, Taiwan, R.O.C.
1988-1990	Bachelor of Engineering
	Department of Electronic Engineering
	National Taiwan University of Science and Technology
1992-1994	Master of Engineering
	Department of Computer Science and Electrical Engineering
	National Central University
2000-2009	Ph.D. of Computer and Information Science
	Department of Computer Science and Engineering
	National Chiao-Tung University
Field Of Study	Multimedia Authoring Tools, E-learning, System Application

Publications

[1] Referred Journal Paper

1. **Shih-Jung Peng**, Pi-Feng Liang and Deng-Jyi Chen, "Effective Learning Model and Activate Learning Algorithm for Improving Learning Efficiency," Journal of Information Science and Engineering, Vol.23, No.6, November, 2007, pp.1849-1863. (SCI)
2. **Shih-Jung Peng**, Jan Karel Ruzicka and Deng-Jyi Chen, "A Generic and Visual Interfacing Framework for Bridging the Interface between Application Systems and Recognizers," Journal of Information Science and Engineering, Vol. 22, No.5, September 2006, pp.1077-1091 .(SCI)
3. Deng-Jyi Chen, **Shih-Jung Peng** and Chin-Eng Ong, "Generate Remote Control Interface Automatically into Cellular Phone for Controlling Applications Running on PC," Journal of Information Science and Engineering, (2008.09.16. accepted.)

[2] Referred Conference Paper

1. **Shih-Jung Peng** and Deng-Jyi Chen, "A Generic Interface Methodology for Bridging Application Systems and Speech Recognizers," 2007 International Conference on Information, Communications and Signal Processing (IEEE ICICS2007), 10-13 December, 2007, in Singapore.
2. **Shih-Jung Peng**, Pi-Feng Liang and Deng-Jyi Chen, "Effect Learning Curve Model and Active Media Learning Algorithm for Improving Learning Efficiency," Taipei ICS 2004 (International Computer Symposium), Dec. 15, 2004, pp. 1097-1102.
3. Pi-Feng Liang, **Shih-Jung Peng**, and Deng-Jyi Chen, "Probability Model and replica Allocation Methods in a Multimedia Mobile Learning System," Taipei ICS 2004 (International Computer Symposium), Dec. 15, 2004, pp. 613-618.
4. Deng-Jyi Chen, Pi-Feng Liang, **Shih-Jung Peng** and Li-Chieh Yu, "An Efficient Learning Model for Mobile Environments using Graph and Probability Analysis," Taipei ICS 2004 (International Computer Symposium), Dec. 15, 2004, pp. 220-225.
5. **Shih-Jung Peng**, Pi-Feng Liang and Deng-Jyi Chen, "Heuristic Media Allocation

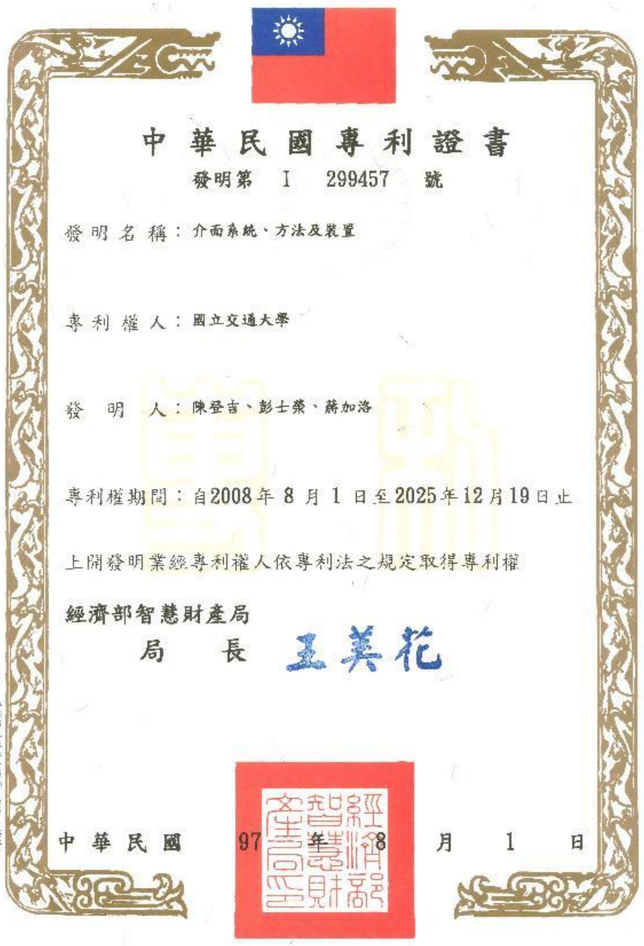
Methods Based on User's Mobility Moving Pattern for Multimedia Mobile System," 2003 International Conference on Informatics, Cybernetics and Systems (ICICS2003), December 15-16, 2003, pp. 535-544.

6. Pi-Feng Liang, **Shih-Jung Peng** and Deng-Jyi Chen, "Media Access Probability Model in a Multimedia Mobile learning System," 2003 International Conference on Informatics, Cybernetics and Systems (ICICS2003), December 15-16, 2003.

[3] Patent

1. 陳登吉，彭士榮，蔣加洛，"介面系統、方法與裝置", patent No I 299457, 2008.08.01.~2025.12.19. For Taiwan.

2. Deng-Jyi Chen, **Shih-Jung Peng**, Jan Karel Ruzicka, "Interface System, Method and Apparatus," patent application No 20070150280A1, 2007.06.28. For USA. (pending)



中 華 民 國 專 利 證 書

發明第 I 299457 號

發 明 名 稱：介面系統、方法及裝置

專 利 權 人：國立交通大學

發　明　人：陳登吉、彭士榮、蔣加洛

專利權期間：自2008年 8 月 1 日至2025年12月19日止

上開發明業經專利權人依專利法之規定取得專利權

經濟部智慧財產局

局　　長　王美花

經濟部智慧財產局

中 華 民 國　97 年 8 月 1 日