設計及製作介面系統產生器用於整合連結前端認知辨識系統及後端應用軟體系統(1/3)

The design and Implementation of Interface Interfacing generator for integrating and bridging front-end recognizers and back-end application software systems (1/3)

計畫編號：NSC97-2221-E-009-062-MY3
執行期限：97 年 08 月 01 日至 100 年 07 月 31 日
主持人：陳登吉　交通大學資訊工程系教授

一、摘要

中文摘要

在[17]的文獻中指出大部份的應用軟體系統裡幾乎有 80%的軟體程式碼是和介面系統有相關的。介面系統的花費是開發應用軟體時必需面對的議題，應用軟體的使用者常以人機介面的好壞來評定該應用軟體的好壞。如何快速並簡化介面系統的開發或修改是一重要的研究主題。一般而言，介面系統可分為人機介面系統和應用軟體及應用軟體之間的整合介面系統。前者是目前較常見的議題，我們已在上期國科會三年期研究計畫有開發一使用者人機介面產生器，而後者是強調在兩種應用軟體之間的整合時所設計的介面系統。本研究針對後者提出一個為期三年的研究計畫。期待能達到有彈性且易延伸的介面之介面系統架構 Interface Interfacing System (如圖 1 所示)，期待能給軟體開發者在製作應用軟體的介面系統時能有較彈性且易維護的解決方式。此計畫的完成我們將予提供一簡易的橋樑，來協助系統開發者將兩種應用軟體(前端的認知器與後端 GUI 的應用軟體)整合成以前端認知器為人機介面的應用系統，將可和前期的國科會計畫整體連成一無縫式的軟體介面系統的整體解決方案。對軟體介面將有重要的貢獻。

下圖為整合後端 window 環境內的傷心小棧應用軟體及前端語音辨識應用軟體之間的介面整合的 interface interfacing system。透過此介面系統 軟體開發者可較容易整合一使用語音為輸入的傷心小棧(solitary)的應用軟體。換句話說是將現有的傷心小棧(solitary)的軟體可透過語音的方式來做人機操作介面的軟體。其他應用軟體若要使用相同語音方式來操控應用軟體亦可，此 interface interfacing system 將比傳統的方法在製作介面系統時達快速且不易出錯的好處，同時可減輕介面軟體程式設計製作者的維護。

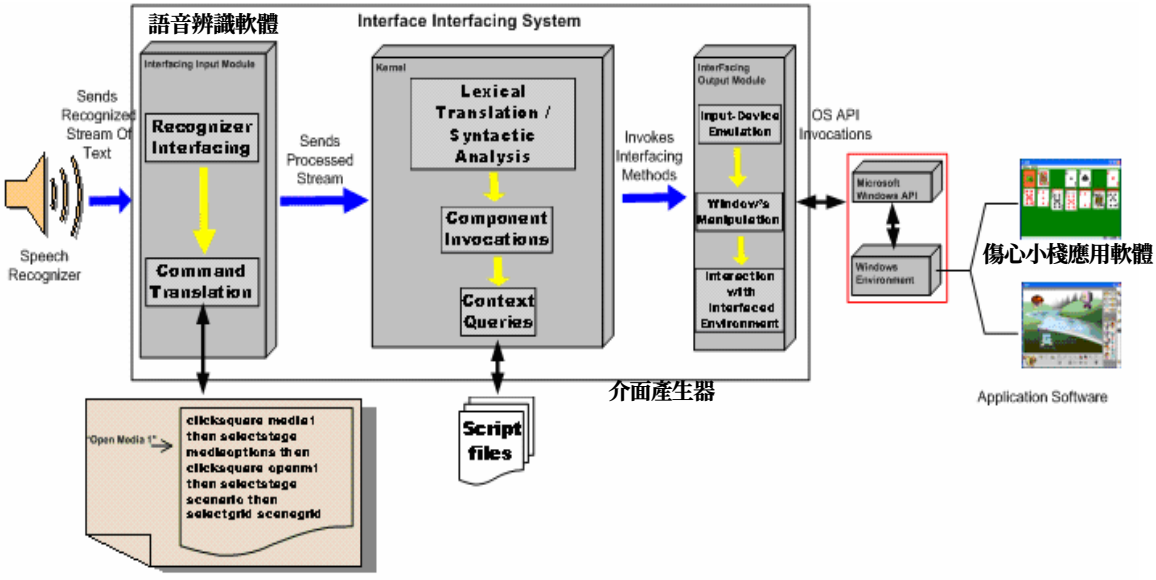


圖 1. The proposed Interface Interfacing System

此 Interface Interfacing System，我們已做了先期研究。部份內容已發表在 Journal of Information Science and Engineering 期刊上。本計畫基於此研究成果上，重新作一較完整的規畫並實作完成此理想的架構及系統。

在第一年的計畫裡我們將先期研究成果重新規畫並定義其系統架構，包括前端的語音命令語言(command language)、parser，以及後端的應用程式的介面整合模組。建立基本架構與定義所需的 Script 語言及 GUI 模組，先以滑鼠進行測試，接著企圖提供一個有系統的方法利用語音辨識操控應用軟體來和滑鼠模組結合以達使用語音來控制滑鼠的互動工作。

在第二年裡我們將把第一年在 PC 上開發的觀念架構修整使其能使用到手持裝置上(如 PDA、smart phone 等環境)，專注於設計及製作一個應用程式介面載入器用以載入 PC 上 Java AP 與處理來自手機操作介面程式的控制命令。設計及製作一個手機內 Java 程式之介面產生器用以產生手機內的 Java 應用軟體之介面設定使其遙控 PC 上相同之應用軟體。前端的認知器改為遙控方式而後端的應用程式則在 PC 環境模擬成功後再移植到手持系統的環境 本研究以 Java Midlet AP 為例子，透過 Smart phone 的手持系統實際遙控應用軟體，圖 2 為一各模組間的 Remote Interfacing System 架構。

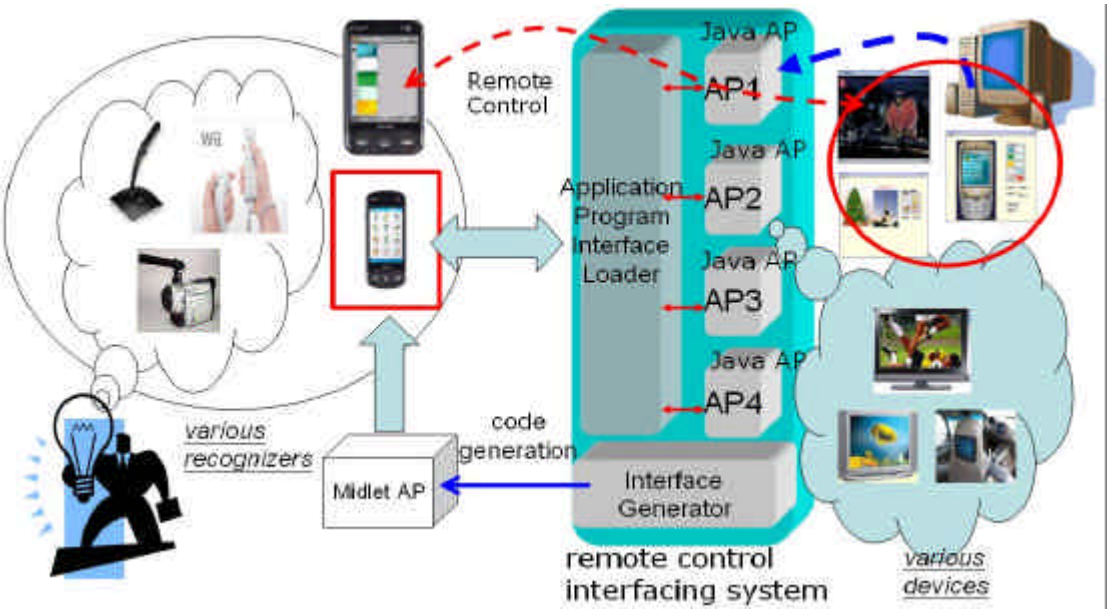


圖 2、Remote Control Interfacing System Overview

第三年我們將前兩年的研究成果加以應用到多媒體內容的製作及不同的多媒體播放平台上，例如行動多媒體名片樣板編輯器及播放器、電子賀卡樣板編輯器及播放器，最後考量這些已編輯完成的多媒體軟體內容在大型 LCD 及 LED 平台上可以撥放的多媒體協調技術，並實際以這些應用軟體為例來驗證所提出的介面整合系統的效益。

關鍵詞：See-through interface、程式碼的剖析器(Parser Generator)、介面產生器、認知辨識、語音辨識(Speech Recognizers)、軟體工程

英文摘要

It has been shown that the major effort spent on the design and implementation of the application system software is the user interfaces (UI) [17] (or human-machine interface (HMI)). If UI can be developed in a short time, it will be a great help to reduce development time for application software systems. Therefore, many researchers have been seeking better solutions to aid UI designers to create UI systems.

In general, there are two kinds of interface system: human-machine interface and interface for bridging application software as one. The former concerns the GUI design and implementation for the application software. The later concerns with the integration of recognizer and application software together to form a new application

software that uses the recognizer as the front-end system. In this proposal research, we layout a three-year integration project that focus on the later interface technology, called generic Interface Interfacing system, Figure 1 depicts the detailed components.

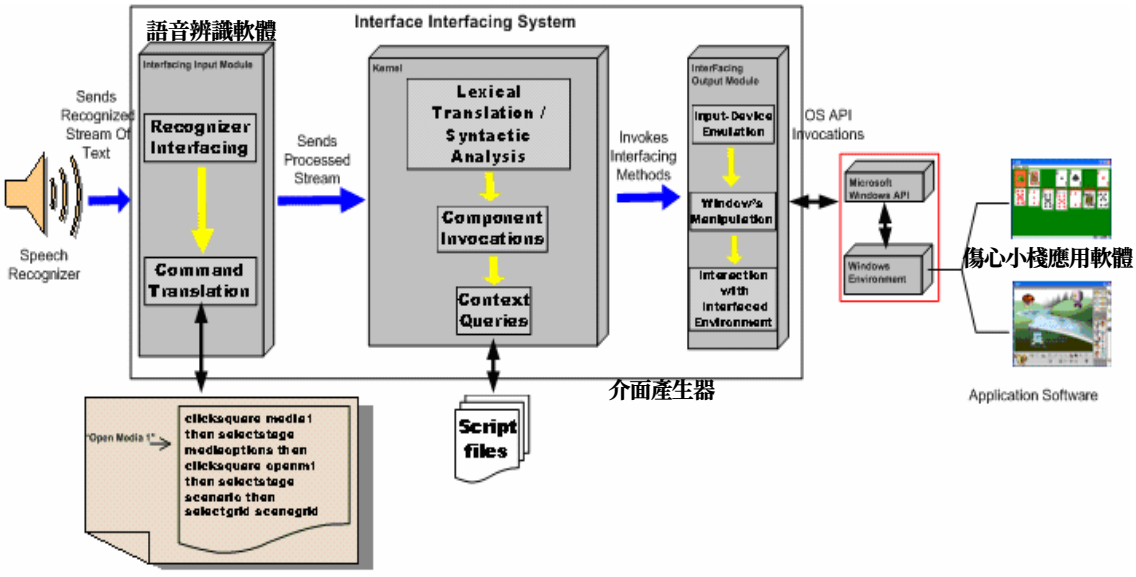


Figure 1、The proposed Interface Interfacing System

Basically, application systems that utilize recognition technologies such as speech, gesture, and color recognition provide human-machine interfacing to those users that are physically unable to interact with computers through traditional input devices such as the mouse and keyboard. Current solutions, however, use an ad hoc approach and lack of a generic and systematic way of interfacing application systems with recognizers. The common approach used is to interface with recognizers through low-level programmed wrappers that are application dependent and require the details of system design and programming knowledge to perform the interfacing and to make any modifications to it. Thus, a generic and systematic approach to bridge the interface between recognizers and application systems must be quested.

In the first year of this integration research work, we propose a generic and visual interfacing framework for bridging the interface between application systems and recognizers through the application system's front end, applying a visual level interfacing without requiring the detailed system design and programming knowledge, allowing for modifications to an interfacing environment to be made on the fly and more importantly allowing the interfacing with the 3rd party applications without requiring access to the application's source code. Specifically, an interfacing script

language for building the interfacing framework is designed and implemented. The interfacing framework uses a see-through grid layout mechanism to position the graphic user interface icons defined in the interfaced application system. The proposed interfacing framework is then used to bridge the visual interface commands defined in application systems to the voice commands trained in speech recognizers. The proposed system achieved the vision of interface interfacing by providing a see-through grid layout with a visual interfacing script language for users to perform the interfacing process. Moreover such method can be applied to commercial applications without the need of accessing their internal code, and also allowing the composition of macros to release interaction overhead to users through the automation of tasks. Figure 1 also indicates an example that a solitary game or an authoring system in window system can be played using the speech recognizer in window system after the integration using the proposed approach.

The main contributions of such interface interfacing system include 1) Productivity is reasonable good: system developers no need to trace the low level code (without requiring the detailed system design and programming knowledge) while integration the recognizer with the application software, 2) Maintenances effort is low: allowing for modifications to an interfacing environment to be made on the fly, and 3) Flexibility is good: allowing the interfacing with the 3rd party applications without requiring access to the application's source code.

In the 2nd year project, we continue the concept used in the first year to investigate the handheld device environment such as PDA or Smart phone. In this case, we use the remote control capability in the smart phone as the front-end recognizer and java program as the back-end application software. The choice of the java as the implementation language is rested on its heterogeneous platform adaptation features. Specifically, we will propose an interface generator, similar to the concept of the parser generator, to automatically generate remote control programs for a specific multimedia application in the smart phone. With this generator, designer does not need to write the textual remote control programs in the smart phone. This will simplify the development process and make the control system development and modification more flexible. Figure 2 depicts the detailed components. In Figure 2, it indicates that a interface generator (the interface interfacing system) can proceed to perform a code generation (Java Midlet AP) after the back-end application software in the PC environment has been integrated with the remote control module using the

proposed approach. Of course, the remote control module can be replaced by Wii-like recognizer if it is needed.

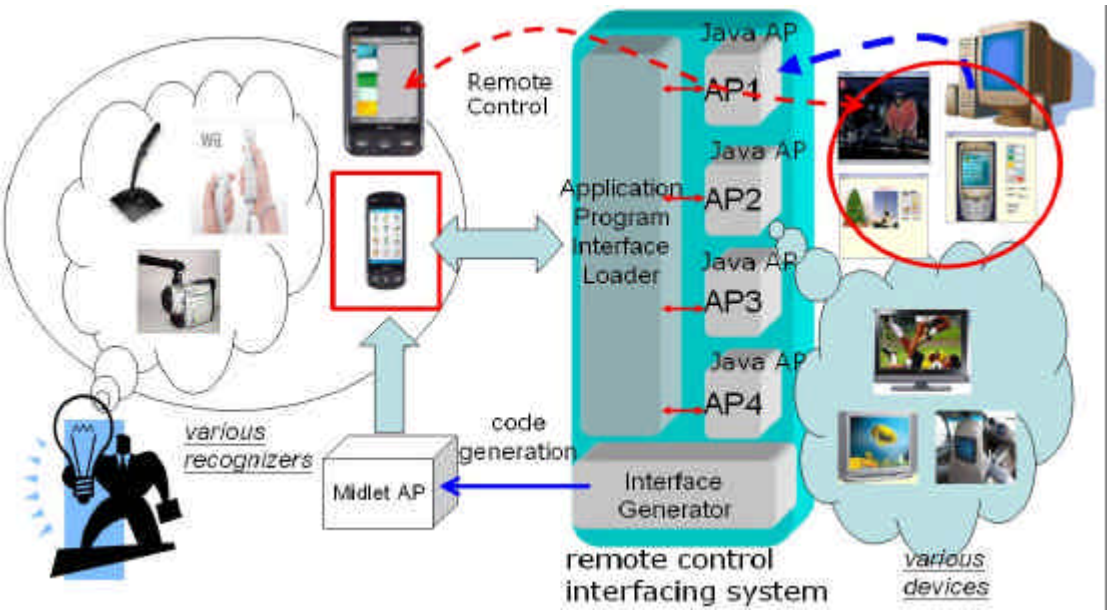


Figure 2、Remote Control Interfacing System Overview

With the quick advance of technology, screen display of digital TV and mobile system becomes more and more elegant and is able to present fine and vivid multimedia contents. Most of the multimedia contents, such as advertisement, motion pictures, messages, etc., can be displayed on different kinds of platforms. If user can use some simple instruments (such as smart phone, PDA, etc.) to remotely communicate with the multimedia application module in the display device (such as PC monitor, digital TV, etc.), then the control becomes live and interesting. But there are various control instruments and display devices, and different kinds of control methods. If one wants to write the control program or partially modify the control features for the multimedia application module in the display device, then he needs to know the software source code in the multimedia application module that will be remotely controlled, so that he can custom-design a set of remote control programs for each multimedia application. However, there is a lot of multimedia application; a custom design for each of these applications becomes time consuming and less efficient.

Once we have built the interface interfacing system for both in the PC environment (the first-year project) and smart phone environment (the 2nd year project), we are ready to author various kinds of multimedia presentation such as

mobile name card template system or e-card presentation and use smart phone to remotely synchronize the presentation on top the big LCD and LED displayers。This is the major effort spent in the 3rd year.

Keywords：See-through interface、Software Engineering、Parser Generator 、Interface generator、Recognizers、Speech Recognizers

二、計畫緣由與目的

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, as shown in figure 1.

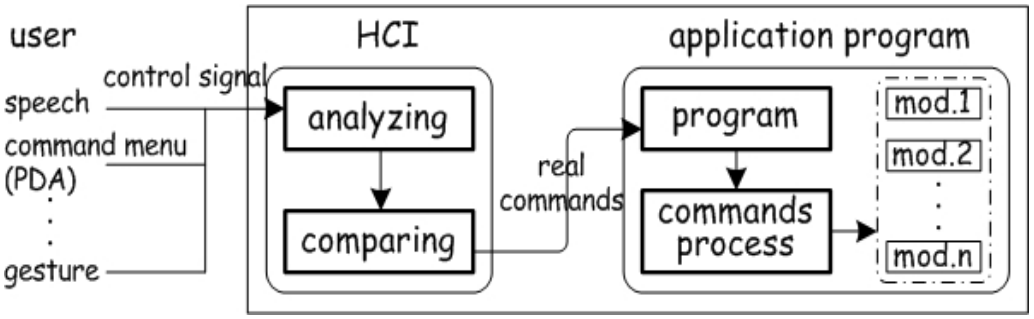


Figure 1 General developing method of HCI control

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.

To overcome these issues, 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 PDA device is taken as an example. 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, which did not have HCI control ability previously, with speech or wireless remote HCI control functions in an easier and more efficient manner and do not need to write any program code, as shown in figure 2.

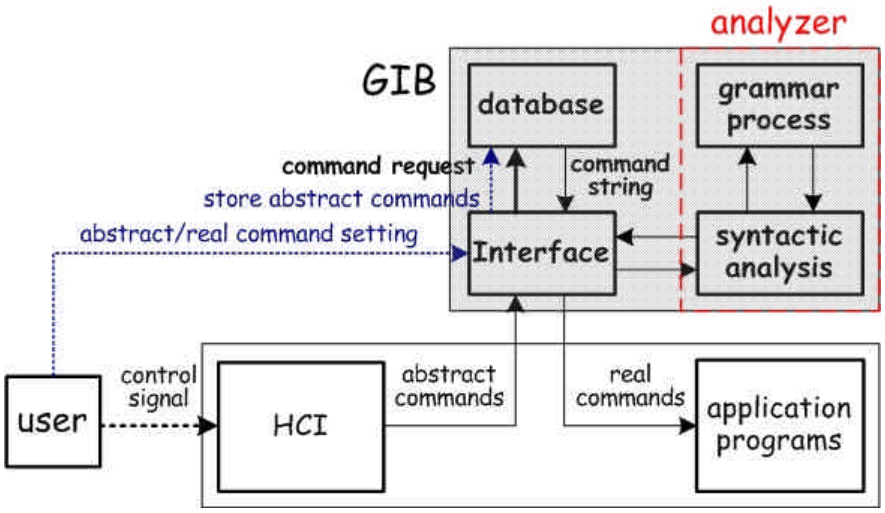


Figure 2 Architecture of GIB control system

三、結果與討論 (第一年)

In the following, we present the 1st year research results of this three years project.

3.1. Introduction (outline of the scope of this three-year project)

Interfacing applications with various recognition technologies (such as speech,

gesture, and color recognition) will impact current methods of interaction in the area of human-machine interfacing technology. Interfacing application systems with these different recognition technologies have opened wide possibilities to these types of users; however current ways of interfacing applications with recognizers are lacking of a generic and systematic way, time consuming, and highly application systems coupled and dependent. Particularly, current solutions that aim at bridging the interface between speech recognizers and application systems usually lead to tightly coupled systems where one application is wrapped by a specific recognizer through a low-level programming implementation that makes the future modifications very difficult. Also, without supporting mechanisms to abstract group of actions into single reusable macro-level commands to simplify user interaction tasks creates intense and time-consuming overheads for end users. Applications systems, especially multimedia oriented ones deal with highly dynamic content, interfacing of this kind of content is not yet addressed. A generic application-independent speech-driven interface framework that allows the generation of a modifiable visual interfacing environment without the need of dealing with low-level details must be quested.

In this research, we attempt to provide a generic and visual interfacing framework for bridging the interface between application systems and recognizers through a generic and systematic approach. Specifically, an interfacing script language is designed and implemented that allows users to define the interfacing commands between a speech recognizer and application software.

3.2. Related Work and the Proposed Solution

Current approaches to interface the interface of speech recognizers and the interface of application software uses a wrapping integration approach that focuses on the integration of the recognizer's API and the application's components through a direct and tightly coupled way (Fig. 1). The application is in charge of setting up the recognizer's environment, grammar domain, receiving recognition results and interpreting these results to perform the respective internal invocations to execute interactions on its GUI [1]. As it can be foreseen, in Fig. 1, the integration results is one application interfaced with one speech recognizer through a interfacing layer that is in charge of directly mapping speech commands into actions on the application's components.

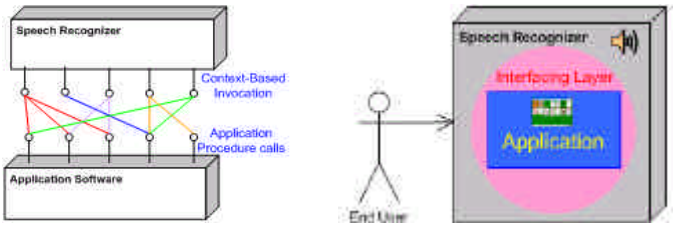
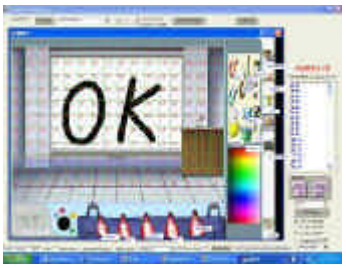


Figure 1、 Wrapping integration.

Most of speech-driven robots adopt such interfacing approach for its design and implementation, for instance, the AT&T's Speech-Actuated Manipulator (SAM) [2]. Under such a tightly coupled-system, it is not surprising that any modifications on the low level application software's commands will result in the recoding of the speech interface, leading to the recompilation of the whole system. Other related application systems such as Vspeech 1.0 [3] and Voxx 4.0 [4] provide interfacing by integrating a speech recognizer with the Window OS environment that is in charge of handling the windows of applications; however they still suffer from the limitations such as low-level interface and requiring detailed system design and programming knowledge.

The common interface approach used in these speech-recognition systems is to interface with recognizers through low-level programmed wrappers that are application dependent and require the details of system design and programming knowledge to perform the interfacing and to make any modifications to it. Thus, we proposed an application-independent visual interfacing generator to bridge the interface of a speech recognizer [5] and the interface of application systems. In the proposed approach, when incorporating a speech-recognizer to an application system, a user through a visual interfacing framework composes a visual interfacing environment by drawing reference zones on top of the GUI's interactive areas (buttons, menu items, links, and containers) of application system, without the need of programming low-level code for the integration. User-generated visual interfacing environments (Fig. 2) for applications are interacted with by the system as it processes user's requests to perform interaction on the environment's zones that are graphically positioned over interaction objects of applications.



(a) Application without reference zone. (b) Application with reference zone.

Figure 2、 The interfacing visual environment

The proposed system interacts with target applications by performing invocations to the Operating System's API and then controls and manipulates the original input-device (such as mouse) defined in the target application under the window environments to perform interactions directly on the visual interfacing environment that lays above target applications' GUI.

3.3. Involved technologies

Creating a successful generic and visual interfacing system for integrating applications with recognizers required the understanding on several technologies, including the "See-Through Interface" paradigm [6], the proposed interfacing script language, and the localized speech-recognizer interfacing mechanisms. These technologies individually be- long to different fields of study, however when implemented in a cooperative environment, these technologies merge to contribute towards the vision of interface interfacing.

3.3.1 See-Through Interface Paradigm

In our visual interfacing framework, the concept of "See-Through Interface" paradigm [6] is employed to construct a transparent grid layout that allows application front- end integration with recognizers through the drawing of reference zones.

In [7], the authors create an immersive environment that submerges users into a virtual space, effectively transcending the boundary between the real and the virtual world. Transparent interfacing allows this virtual 3D world to be manipulated by the user without the need of relaying on traditional input devices such as the mouse or keyboard for interaction.

In our visual interfacing framework, we use a transparent grid layout mechanism to position the GUI icons defined in the interfaced application system. In this way, any GUI based application systems can be interfaced using the proposed visual

framework with different recognizers.

3.3.2 An Interfacing Script Languages

The language specification of the designed script language for this study is simple enough to allow programmers to quickly achieve fluency in the language. Our language design is based on Just-In-Time compilation by compiling the code as necessary, running it in an interpreted framework [9]. In the following subsections, we describe the proposed interfacing script language.

3.3.2.1 Data types

Types limit the values that a variable can hold or that an expression can produce, limit the operations supported on those values and determine the meaning of operations. Strong typing helps detect errors at compile time [9, 10].

3.3.2.2 General static semantics

Commands in the Interfacing Script Language are separated into selection commands that take care of switching the different interfacing visual environment content. Assignment commands that take care of assigning values to system internal identifiers and lastly action commands that focus on interacting with application system's interfacing content, performing actions that directly affect the target application.

3.3.3 Localized Recognizer Interfacing

In our visual interfacing framework, a localized recognizer interfacing by integrating a speech recognizer [5] through its API is designed and implemented. The interface is done by a specialized component that allows the future integration of other recognizers without performing modifications to the rest of the system. The assigned tasks to this component are kept to a minimal in order to maintain the complexity of interfacing a new recognizer at the lowest. These tasks include the listening of recognition content, initialization, setup and handling of the target recognizer only. A more detailed treatment on the proposed interfacing script language can be found in [12].

3.4 the Details of the visual interfacing framework

The proposed visual interfacing framework system interacts with target applications by performing invocations to the Operating System's API to manipulate

its input-device and windows environments to perform interactions directly on the "Transparent Interface" that sits on top of the GUI of the target applications. In the proposed approach, the interfacing of recognition devices and applications is done through two different interfacing layers that interact directly with the system's kernel (Fig. 3).

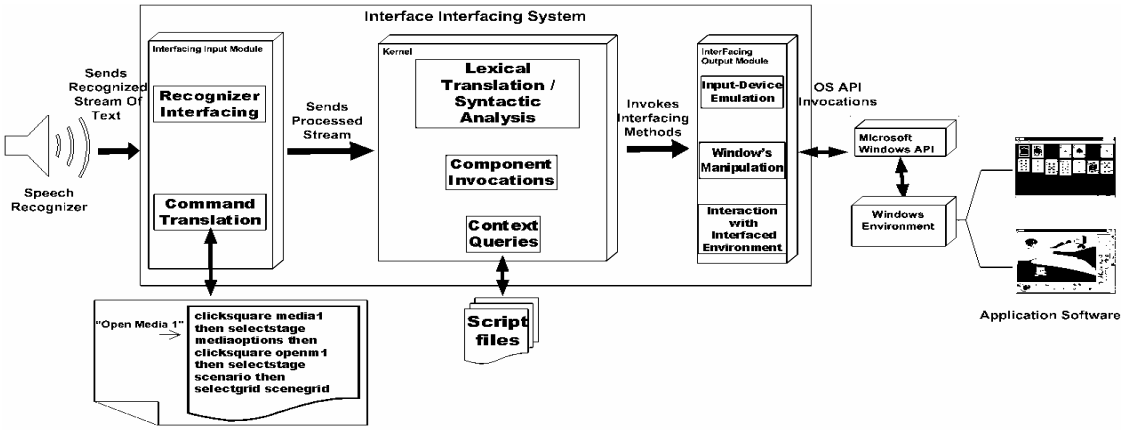


Figure 3、The proposed interface interfacing system.

3.4.1 Interface Input Module Processes

When the speech recognition engine recognizes spoken phrases, it outputs those phrases as text streams in the spoken language, according to how they are defined in the recognizer's XML Grammar Definition. The stream of text is then passed down to a component in charge of translating recognized text into the standard language of the system.

The Macro Interpreter then receives the stream of text and checks if it contains keywords that reference macros, it does so by querying the Macro Data Repository for matches. If a match is found, the keyword inside the stream of text gets replaced with the corresponding macro. Once a macro is loaded, it is passed down to the Wild Card Translator that checks for the presence of wildcards. Wildcards are part of the system's design strategy to allow the reutilization of a macro with different dynamic entities (Actors) by allowing the user to assign values to wildcards during runtime, in this way avoiding the redefinition of macros for every dynamic entity. When a wildcard is found, it is replaced with the current actor that has focus applying the macro to it. Fig. 4 depicts the above mentioned processes.

3.4.2 Kernel Module Processes

Translated commands that result from the Interfacing Input Module process are sent to the Kernel so that they can be interpreted into a target program (Fig. 5) that

provides the interaction behavior to be applied to the interfacing environment. As the stream of text enters the kernel, the Lexical Translator splits the stream of text into token sets. Each token set represents a single command that is fed down to the Syntactic Analyzer for interpretation. When the Syntactic Analyzer receives a token set, it analyses it token by token and traverses the parsing structure until a match of a valid command with a compatible format is found. Once the parsing is successful, the corresponding target program is executed at the Event Delegating Component that delegates the invocations to the respective system components involved in the interaction.

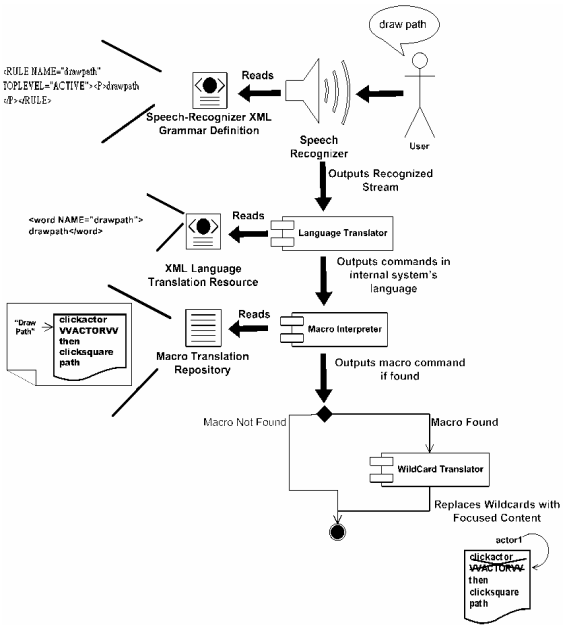


Figure 4、Command translation process.

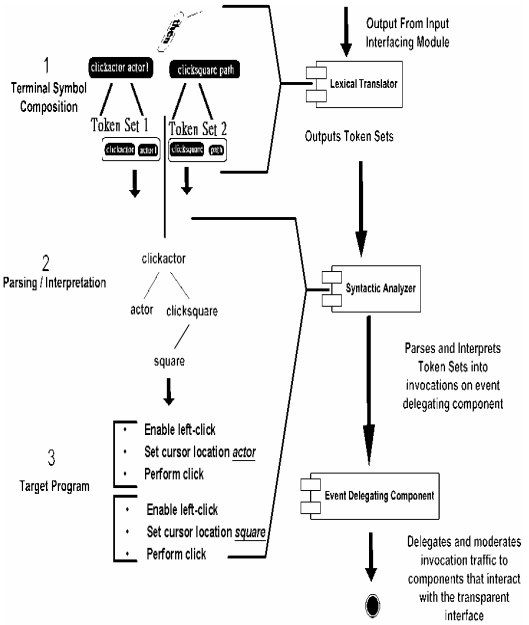


Figure 5、Command interpretation process.

The Lexical Analyzer distributes its chores to four sub-programs (Fig. 6), one in charge of getting the next stream input through an event handling function, other one in charge of building lexemes as described above, other tokenizing sub-program to take care of removing non-relevant characters and finally a subprogram that handles the recognition of reserved words, constants and identifier names. The later with the purpose of validating the content of the data types of the command in question by looking them up in their corresponding tables to make sure they exist in the system and that no reserved word are being used.

In our syntactic analysis we trace a leftmost derivation (Fig. 7), tracing the parse tree in preorder, beginning with the root and following branches in left-to-right order. Expanding non-terminal symbols to get the next sentential form in the leftmost

derivation, basing the expansion route on the type of the non-terminal symbol [9]. Due to the simplicity and recursive nature of the language's grammatical rules, our approach implements a recursive descent parser rather than utilizing parsing tables to accomplish the syntactic analysis, in this way assuring that the next token represents the left most token of input that has not been used in the parsing, this token is compared against the first portion of all existing right hand sides of the non-terminal symbol, selecting the right hand sides where a match is found.

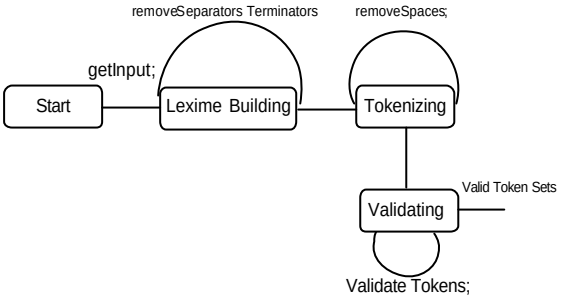


Figure 6、Tokenizing transitions

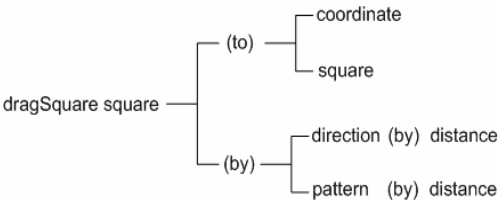


Figure 7、Parsing tree of 'dragSquare'

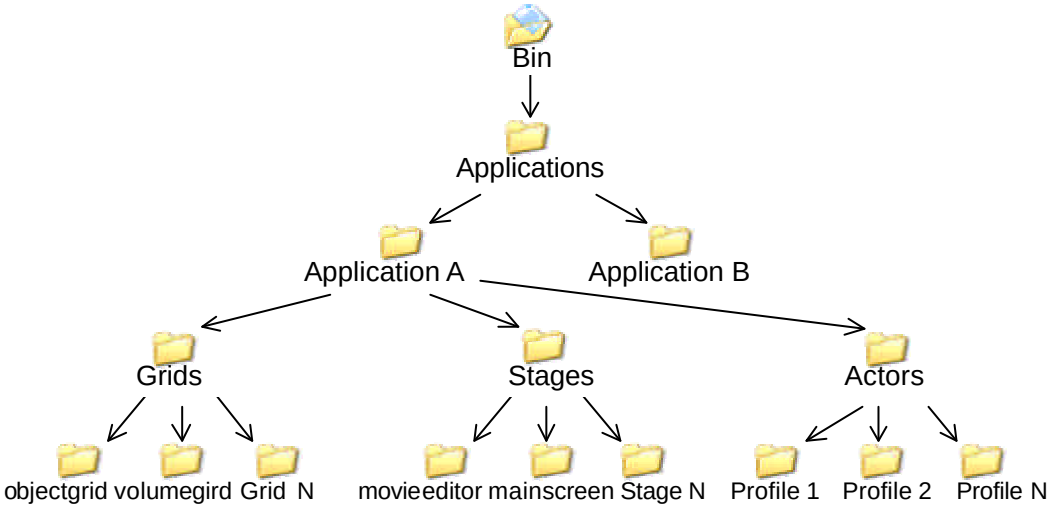


Figure 8、Interfacing objects hierarchical organization.

The Kernel module is also in charge of storing, retrieving, and performing the object activation on the different interfacing objects that are used for building a visual interfacing environment of an application. It also handles the dynamic interfacing content and provides the tracking mechanism to relocate dynamic interfacing object whenever a user interacts with such content. The interfacing script language supports scripting commands for the Kernel module to perform loading, storing, and removing

of objects of type application. These interfacing script commands include *square*, *actor*, *actor profile*, *stage*, *and grid*. The Kernel module also supports the querying mechanism used by other system internal components to retrieve specific information of objects as needed during the interaction process of interaction. Reference interfacing objects of the system are stored- retrieved and modified dynamically into and from a four level hierarchical directory structure, as in Fig. 8.

3.4.3 Interface Output Module Processes

The main function of the Interfacing Output Module is to provide the mechanisms to interact directly with the front-end of application system through the interfacing visual environment by performing input-device emulation and window's environment manipulation, taking care of manipulating input devices to perform mouse or keyboard related actions on the Interfacing Visual Environment through the Input Device Controller component. This component takes care of emulating the following mouse actions: -Left_ Mouse_Click, -Left_Mouse_Double_Click, -Right_Mouse_Click, -Right_Mouse_Double_ Click, -Drag_and_Drop, -Move.

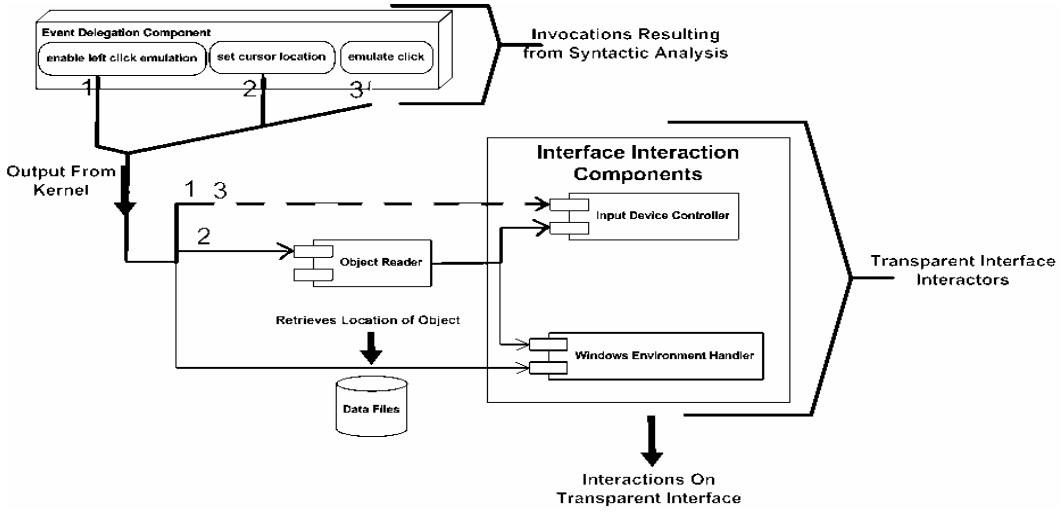


Figure 9、Visual interfacing environment interaction process.

Target programs that result from the syntactic analysis are executed through the Event Delegating Component. Depending on the command, the requests for each of the involved events is sent to corresponding component that interact directly with the interfacing environment through the mechanisms described above, accomplishing the completeness of a command's execution process (Fig. 9). A labeling system is also developed to visually label each of the registered reference zones at their graphic

location with their corresponding registered identification name

3.5 Interfacing procedures and Examples

The procedure involved in interfacing a target application with a speech recognizer through our proposed framework requires the fulfillment of multiple steps that are done to ensure a successful interfacing.

3.5.1 Interface Interfacing Procedures

The interfacing procedure is separated into multiple steps as depicted in Fig. 10:

Step 1: Interface the Target Application.

The first step involved in interfacing an application to a speech recognizer is to register a desired application into the proposed system. Once the target application is registered, we create the visual interfacing environment by drawing reference zones on the transparent interface that lays on top of the application's GUI, in this way referencing application's content such as buttons, containers and menus through the graphic registration of grids and squares, separating this content into stages that each represent the different GUIs of the application. Fig. 11 lists the detailed procedures of the target application software registration.

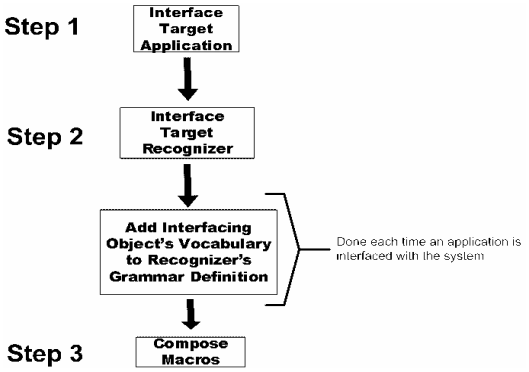


Figure 10、Interface interfacing procedure

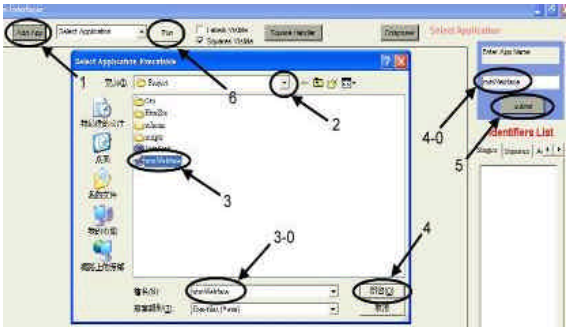
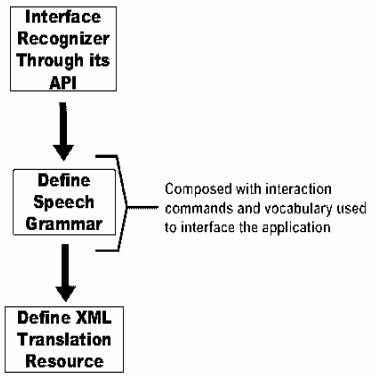


Figure 11、Registration target application



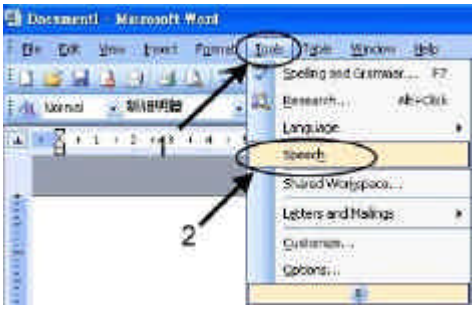
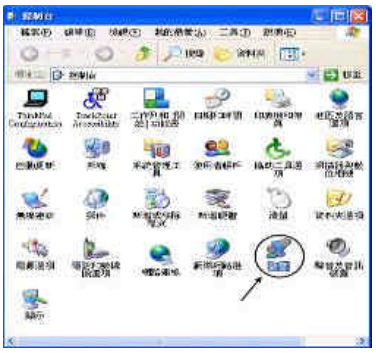
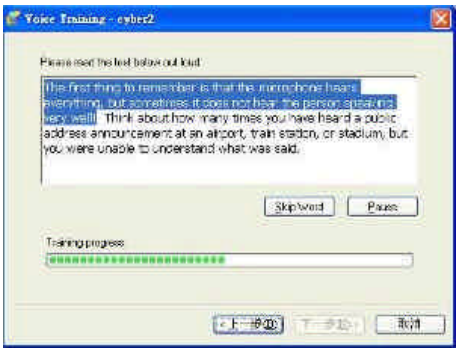


Figure 12、Recognizer interfacing steps

Figure 12a、Installation of speech-recognizer.



(b)



(c)

Figure 12b、12c、The Microsoft's speech-recognizer training.

Step 2: Interface the Target Recognizer.

The second step is to interface the chosen recognizer, that wanted to be integrated into the proposed interfacing framework system, by programming the recognizer's API calls that are used to start, setup and handle the recognizer and as well as the calls involved in retrieving recognition content in the system's specialized recognizer interfacing component. In the following, we provide an example by illustrating the interface of the Microsoft's Speech-Recognizer V.6.1 [5] with the proposed interfacing framework system. Fig. 12 shows the major steps in this Speech-Recognizer integration. Procedures to install and training the Microsoft's Speech-Recognizer V.6.1 are listed as shown in Figs. 12 (a-c).

```
<RULE NAME="sqrs ">
<l> <P>save</P> <P>player</P> <P>new</P> <P>normal</P>
<P>duplicate</P>
Continues …
```

Figure 12d、Recognition vocabulary preparation.

```
<RULE NAME="dragsquare" TOPLEVEL="ACTIVE">
<P>dragsquare</P> <o>
<RULEREF NAME = "sqrs" / > <o> <p>to</p>
<l> <P>save</P> <P>player</P> <P>new</P> <P>normal</P>
<P>duplicate</P>
Continues …
```

Figure 12e Composed rule definition that uses references to other lower-level rules.

```
<grammar>
<word NAME="Actor">Actor</word>
<word NAME="Profile">Profile</word>
Continues …
```

Figure 12f、 Translation repository.

Whenever an application is interfaced with the system, a copy of this generic grammar definition is customized by adding the corresponding vocabulary that was used to create the interfacing environment of the application in question (Fig. 12 (d)). The script program will be generated automatically.

The grammar definition consists of a set of rules that are defined through extensible markup language (Fig. 12 (e)). These set of rules are used by the speech-recognizer to validate recognized words, restricting the possible words or sentences chosen during the speech recognition process.

Not in all cases the grammar defined for the recognizer's will match the exact syntax of the system's language (perhaps a recognizer that does not support speech is integrated to the system, such as a motion recognizer), to tackle this problem the definition of a translation XML resource file is made (Fig. 12 (f)).

Step 3: Macro Composition.

Once an application is properly interfaced with a speech recognizer, we compose a set of macro commands to simplify user interaction with the interfaced environment by wrapping complex and repetitive tasks into short, reusable context free commands.

The registration of macro commands (Fig. 13) takes place through a macro composer where the user composes the macros by writing their execution content in the system's defined language and writing a "keyword" that is used to reference the macro during the invocation process.

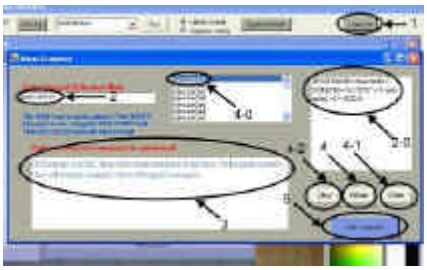


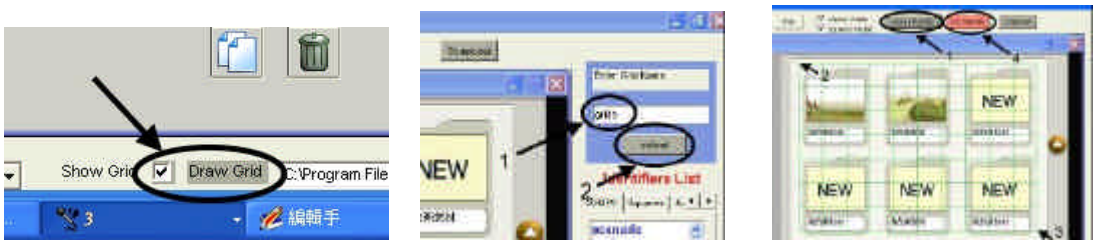
Figure 13、Registering a macro

Figure 14、Registering squares

3.5.2 Interface Interfacing Objects

When referencing a target application, an interfacing environment is created where different objects are used to reference interaction areas of the application. *Squares* are referencing objects used to interface buttons or zones of applications, each *square* has a name given by the user and they are registered by drawing them on top of the interaction zone to interface. To register a *square* one must first select the desired *stage* to associate the *square* with. Objects known as *stages* are created for organizing and separating the different *squares* that are registered, separating them based on the different GUIs that the application presents. Each *stage* has a name given by the user. Fig. 14 lists the detailed procedures of registering a square named ‘mountain’.

More complex referencing objects such as *grid*, are built and composed of auto-generated *square* objects and are used to reference *panes* and *containers* of the target application, allowing for a localized referencing through *coordinates*. Each *grid* has a name given by the user, and they are registered through drawing on the desired interaction zone. Figs. 15 (a-c) lists the detailed procedures of registering grids command named ‘grids’.



(a) Choose Draw Grid (b) Give grids a file name (c) Procedure of drawing grids

Figure 15、Registering grids.

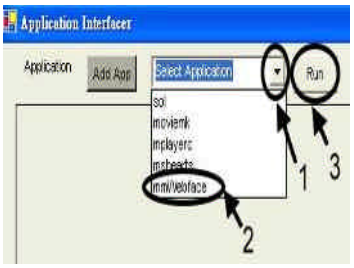
Fig. 16 lists the detailed procedure of registering an actor profile named ‘TVactor’. P1) Press Add Actor Profile (labeled as 1-0) in Fig. 16 and the system will generate a profile name automatically. P2) Select an actor (labeled as 2). P3) Choose an actor control function (labeled as 3). P4) Draw a moving path of actor (labeled as 4). The ‘TVactor’ will move around as specified by the created moving path when a voice command is given during the run time environment.

Figure 16、Registering dynamic content actors and actor profiles

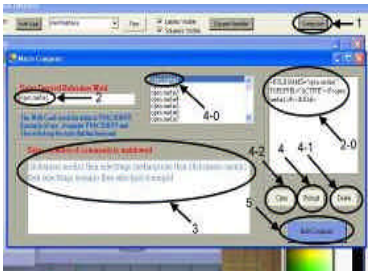
3.5.3 Examples with Interfacing Applications

The proposed interfacing framework has been used for interfacing several commercialized applications with the Microsoft Speech Recognizer. Figs. 17 (a-f) depicts some snapshots for the interface with Bestwise's Visual Authoring Tool (2004 version). A completed example can be found in [13].

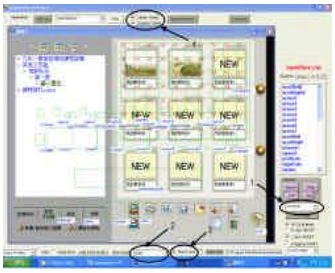
(a) Choose recognizer language



(b) Install interfacing environment



(c) Registering macro command



(d) Choose stage and grids

(e) Speech a macro to control system

(f) Speech a macro to control system

Figure 17、Snapshots for the interface with Bestwise's Visual Authoring Tool

3.6 Conclusion

This research overcomes some common problems suffered by developers when bridging an application system to the interface of a recognizer. The proposed approach presents a more flexible and efficient interfacing. To design and implement the

proposed interface interfacing framework, we addressed a number of challenges and limitations imposed by current approaches, by employing several techniques such as the “See-Through Interface”, object oriented design patterns, and incorporate a script language definition together with a parsing technique. As a result, the proposed interface interfacing framework enhances the interfacing of applications to recognizers by making it an easy, generic and flexible process.

The major contributions of this study include:

1) Offers a simplistic and personalized way to interface applications with recognizers through the front-end, without the need of dealing with low-level issues such as system design and programming.
2) Allows modifications to a recognition interfacing environment of an application without requiring the access to source code of applications and re-compilation of it.
3) Offers a generic and custom interface interfacing environment that allows the coexistence of multiple applications that hold different interfacing requirements.
4) Tackles the challenges and limitations imposed by current solutions that focus on wrapping a single application with a single recognizer in a highly coupled manner.

四. 計畫成果自評

In the 1st year project, we have completed a generic and visual interfacing framework for bridging the interface between application systems and recognizers through the application system’s front end, applying a visual level interfacing without requiring the detailed system design and programming knowledge, allowing for modifications to an interfacing environment to be made on the fly and more importantly allowing the interfacing with the 3rd party applications without requiring access to the application’s source code. Specifically, an interfacing script language for building the interfacing framework is designed and implemented. The interfacing framework uses a see-through grid layout mechanism to position the graphic user interface icons defined in the interfaced application system.

The research results from this project have been submitted to conferences and journals for publication. Also, part of the technology developed from this research

project has been filed patents application in the territory of Taiwan and the U.S.A. through the IP office of National Chiao Tung University. These related technology developed in this project has been technology transferred to industrial sectors.

Papers Publication:

1) **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)
2) **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
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.) **(SCI)**
4) Chung-Yueh Lien, Hsu-Chih Teng, **Deng-Ji Che**n, Woei-Chyn Chu, and Chia-Hung Hsiao, " A Web-Based Solution for Viewing Large-Sized Microscopic Images" Journal of Digital Imaging, **Published online:** 27 June 2008, 0897-1889 (Print) 1618-727X (Online). doi: 10.1007/s10278-008-9136-x http://www.springerlink.com/content/109379/.,

Patent:

1) **專利名稱：介面系統、方法及裝置 INTERFACE SYSTEM, METHOD AND APPARATUS．專利範圍：中華民國(台灣)．專利發明人：陳登吉 彭士榮 蔣加洛．發明專利號碼：(第 I 299457)．專利期限：From 2008/08/01 to 2025/11/10.**
2) **專利名稱：多媒體簡訊樣板套用系統及播放系統、多媒體簡訊樣板套用方法及播放方法，專利範圍：中華民國(台灣)．專利發明人:陳登吉 洪啟彰 楊博鈞；發明專利號碼：(第 I 292667)．專利期限：From 2008/01/11 to 2025/12/13.**

3) A Generic and Visual Interfacing Framework for Bridging the Interface between Application Systems and Speech Recognizers (USA) inventors: 陳登吉 彭士榮 蔣加洛(pending)

Technology transfer:

1) 學習部落格內的文件控管及保護機制技術(技轉給智勝國際科技公司),July 31, 2008.

五.參考文獻

1. B. Balentine, D. Morgan, and W. Meisel, How to Build a Speech Recognition Application, Enterprise Integration Group, 1999.
2. Speech-Actuated Manipulator, http://www.research.att.com/history/89robot
3. VSpeech 1.0, Team BK02 product, http://vspeech.sourceforge.net.
4. Voxx 4.0, Voxx Team product, http://voxxopensource.sourceforge.net.
5. Microsoft' s Speech Recognizer V.6.1, Microsoft product, http://www.microsoft.com.
6. E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose, "Toolglass and magic lenses: the see-through interface," Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304.
7. Y. Boussemart, F. Rioux, F. Rudzicz, M. Wozniewski, and J. 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.
8. S. K. Huang, "Objected-oriented program behavior analysis based on control patterns," a Ph.D. Dissertation, Department of Computer Science and Information Engineering, National Chiao Tung University, Taiwan, 2002.
9. R. W. Sebesta, Concepts of Programming Languages, 5th ed., Addison-Wesley Publishing Company, 2002.
10. J. Gosling, B. Joy, G. Steele, and G. Bracha, The Java Language Specification, 2nd ed., Sun Microsystems, Inc., 2000.
11. BestWise International Computing Company, http://www.caidiy.com.tw.
12. J. K. Ruzicka, "The design and implementation of an interfacing framework for bridging speech recognizer to application system," a Master Dissertation,

Department of Computer Science and Information Engineering, National Chiao Tung University, Taiwan, 2005.

13. S. J. Peng, "Bridging the interface between application systems and recognizers," Technical Report No. NCTU-CSIE-SE-TR-001, Department of Computer Science and Information Engineering, National Chiao Tung University, Taiwan, 2005.

14. WinBatch Macro Scripting Language, http://www.winbatch.com/.

15. B. P. Douglas, Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems, Addison-Wesley Publishing Company, 2003.

16. Microsoft Speech SDK, Version 5.1 Documentation, Microsoft Corporation, 2001.

17. E. Lee, "User-interface development tools," *IEEE Software*, Vol. 7, 1990, pp.31-36.