JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 22, 1077-1091 (2006)

A Generic and Visual Interfacing Framework for Bridging the Interface between Application Systems and Recognizers*

SHIH-JUNG PENG, JAN KAREL RUZICKA AND DENG-JYI CHEN
Department of Computer Science and Information Engineering
National Chiao Tung University
Hsinchu, 300 Taiwan

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 mouse or keyboard. Current solutions to interface application systems with recognizers, however, use an ad hoc approach and lack of a generic and systematic way. 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 this research work, we provide 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 transparent 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 can be applied to GUI based commercial software without the need of accessing their internal code, and also allowing the composition of macros to reduce interaction overhead to users. Examples are applied using the proposed interfacing mechanisms to demonstrate the applicability and feasibility of the proposed visual interfacing approach.

Keywords: interfacing environment, recognizer interfacing, see-through interface, generic interfacing framework, application interfacing, customized interfacing

1. MOTIVATION

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

Received August 16, 2005; accepted January 17, 2006.
Communicated by Jhing-Fa Wang, Pau-Choo Chung and Mark Billinghurst.
* This research was supported in part by the National Science Council of Taiwan, BestWise International computing Co., CAISER (National Chiao Tung University, Hsinchu, Taiwan), and Ta Hwa Institute of Technology (Hsinchu, Taiwan).

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.

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.

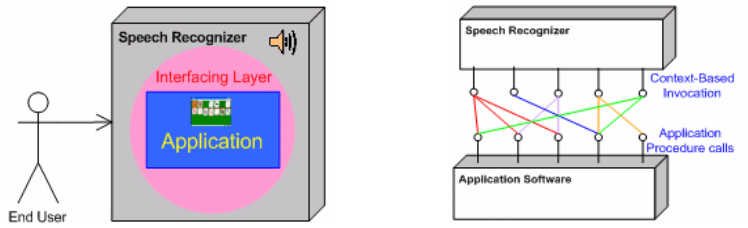
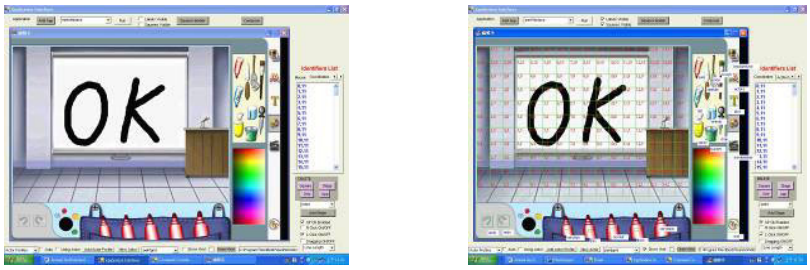


Fig. 1. Wrapping integration.

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, lead-

ing 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.

Fig. 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. 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 belong to different fields of study, however when implemented in a cooperative environment, these technologies merge to contribute towards the vision of interface interfacing.

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.

3.2 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.

3.2.1 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].

3.2.2 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 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].

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).

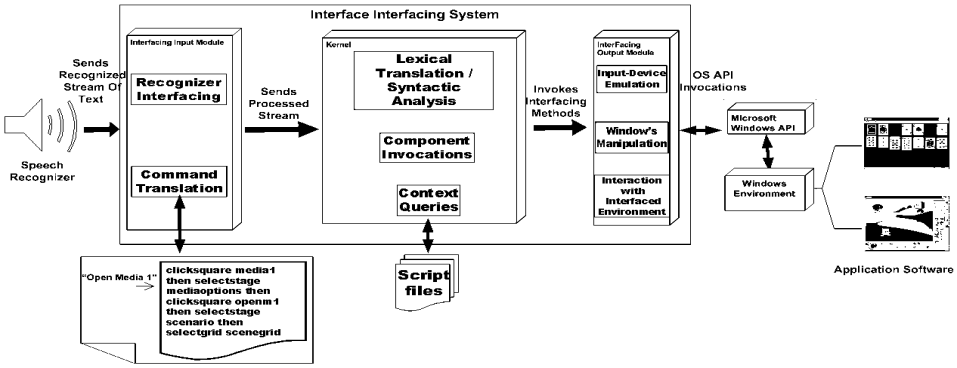


Fig. 3. The proposed interface interfacing system.

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.

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

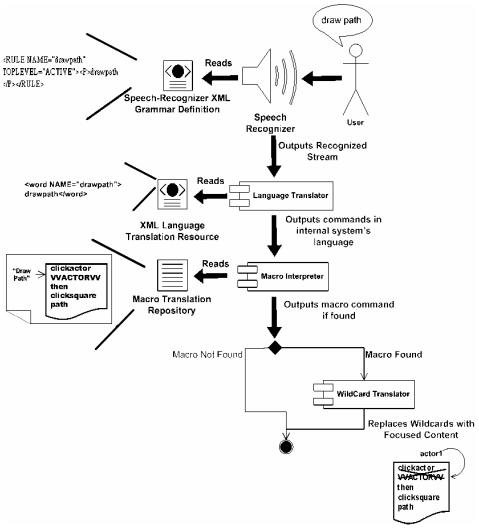


Fig. 4. Command translation process.

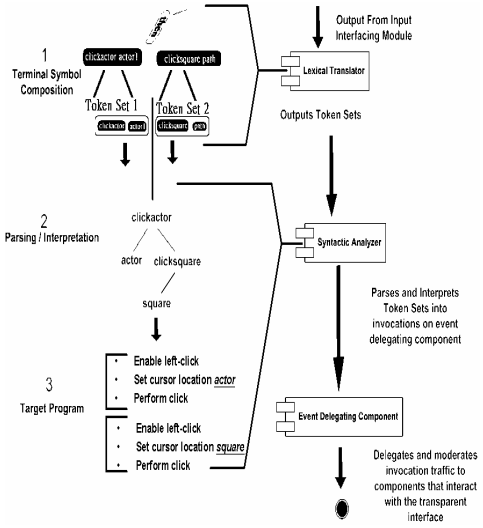


Fig. 5. Command interpretation process.

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.

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.

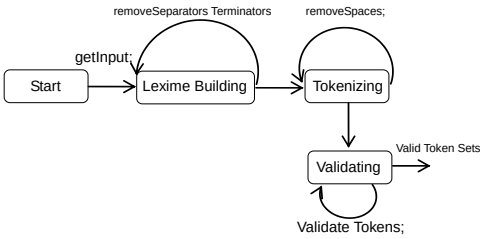


Fig. 6. Tokenizing transitions.

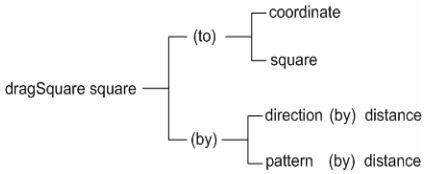


Fig. 7. Leftmost derivation parsing tree of the ‘dragSquare’ command.

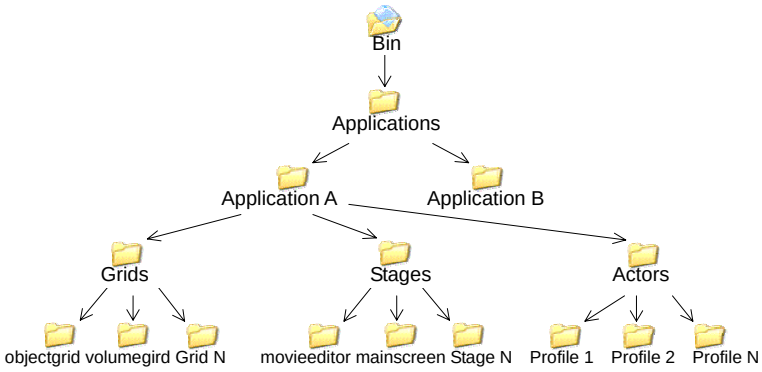


Fig. 8. Interfacing objects hierarchical organization.

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.

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.

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

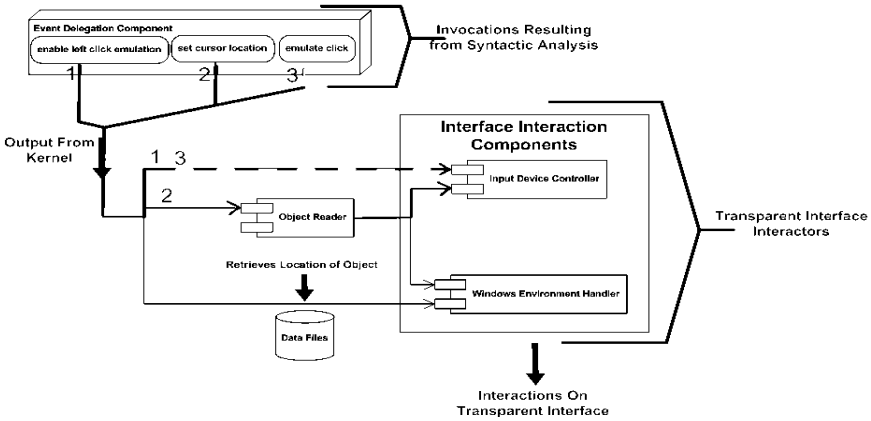


Fig. 9. Visual interfacing environment interaction process.

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.

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.

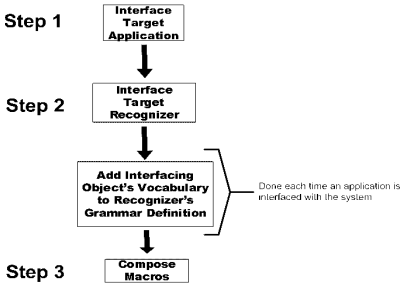


Fig. 10. Interface interfacing procedure.

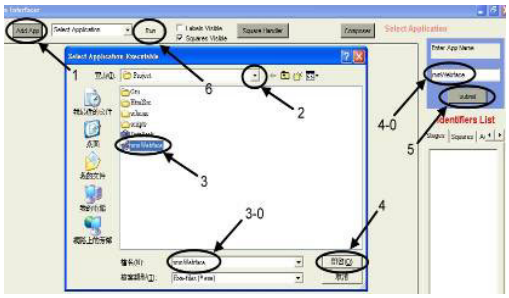


Fig. 11. Registration target application.

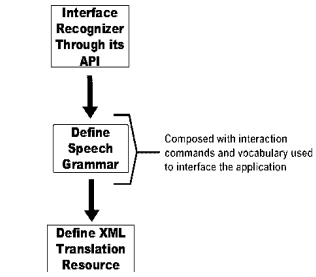
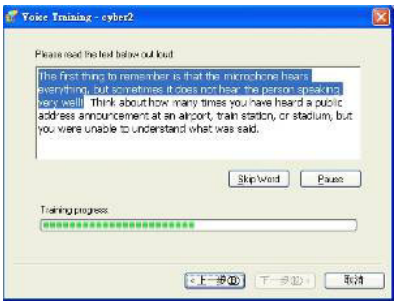


Fig. 12. Recognizer interfacing steps.

Fig. 12. (a) Installation of the Microsoft's speech-recognizer.

(b)



(c)

Fig. 12. (b, c) 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 …
```

Fig. 12. (d) 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 …
```

Fig. 12. (e) Composed rule definition that uses references to other lower-level rules.

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

Fig. 12. (f) 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.

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

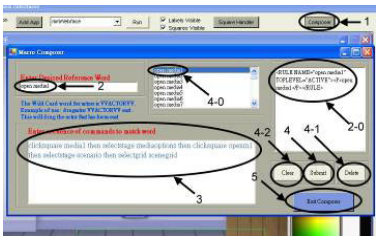


Fig. 13. Registering a macro.

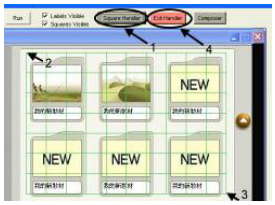
Fig. 14. Registering squares.

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.

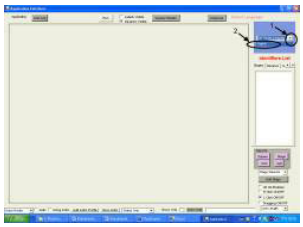
Fig. 15. Registering grids.

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.

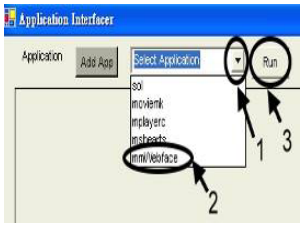
5.3 Examples with Interfacing Applications

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].

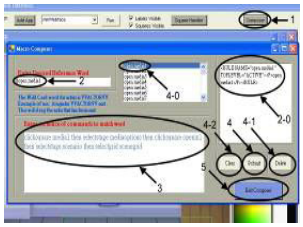
Fig. 16. Registering dynamic content actors and actor profiles.



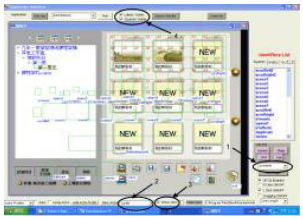
(a) Choose recognizer language.



(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.

Fig. 17. Snapshots for the interface with Bestwise's visual authoring tool.

6. ASSESEMENTS AND CONCLUSION

For the evaluation of the proposed approaches, we use both the BestWise's visual authoring software [11] and Solitary game to interface with the speech recognizer through the proposed visual interfacing framework system [5]. The interfacing of the target application was done by drawing reference zones of its interaction objects on a transparent interface to then create a grammar definition for the speech engine to recognize these interfacing environment's zones. Once a visual interfacing visual environment was made for the target application, several macro commands were composed, and finally some interaction scenarios were executed. The details of the interface procedures can be found in [13].

6.1 Evaluation Results

Based on the interfacing applied to the target application, we evaluate the process involved in integrating it with a speech recognizer through our framework and provide a

comparison on what advantages where experienced. Table 1 summarizes how current challenges and limitations of current interfacing methods are tackled through our proposed interfacing framework.

Table 1. A comparison on the proposed approach against current application challenges.

Challenge	Our proposed interfacing approach	Current interfacing approaches
Generic	It can be used to interface one recognizer with multiple applications currently and will be expanded to multiple recognizers with multiple applications.	They focus on only programming a direct interfacing of one application with one speech recognizer.
Complexity	Allows interfacing a recognizer with the 3rd party applications through a visual environment without the need of accessing their source code.	Recognizer integration through the back-end of applications and requiring low-level programming and system design knowledge.
Customizable	Allow modifications under a visual interfacing environment to be done at run-time, without the need of re-compilation of any source code.	Highly coupled system design. Any modifications on application's interfacing environment will require the re-compilation of source code.
Efficiency	Integrates a script language for users to interact directly with the system in real-time and allows the composition of context-free reusable macros to simplify user interaction.	Lack of a post-interfacing mechanism to abstract a group of actions into single composed commands to minimize and simplify user interaction tasks.

6.2 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.

REFERENCES

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," 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," 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.

Shih-Jung Peng (彭士榮) received the B.S. degree in Electronic Engineering from the National Taiwan University of Science and Technology, Taipei, Taiwan, and M.S. degree in Computer Science and Information Engineering from National Central University, Taoyuan, Taiwan, in 1990 and 1994, respectively. He is now a Ph.D. student at Computer Science and Information Engineering Department of National Chiao Tung University, Hsinchu, Taiwan, and he is also a Teacher at Information Management of Ta Hwa Institute of Technology, Hsinchu, Taiwan. His research interests include e-learning, window's application, performance and reliability modeling.

Jan Karel Ruzicka Gonzalez (蔣加洛) received the B.S. degree in Computer Science from the Institute of Information and Technology Formation, San Jose, Costa Rica, and M.S. degree in Computer Science and Information Engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2001 and 2005, respectively. He has been involved in the field of software engineering as a project leader in the various developments of reusable object oriented technologies utilized in web applications (automatization of customized context generation), console gamming (reusable videogame engines) and window's applications.

Deng-Jyi Chen (陳登吉) received the B.S. degree in Computer Science from Missouri State University (cape Girardeau), U.S.A., and M.S. and Ph.D. degrees in Computer Science from the University of Texas, Arlington, U.S.A. in 1983, 1985, 1988, respectively. He is now a professor at Computer Science and Information Engineering Department of National Chiao Tung University, Hsinchu, Taiwan. Prior to joining the faculty of National Chiao Tung University, he was with National Cheng Kung University, Tainan, Taiwan. So far, he has been publishing more than 130 referred papers in the area of software engineering (software reuse, object-oriented systems, visual requirement representation), multimedia application systems (visual authoring tools), e-learning and e-testing system, performance and reliability modeling and evaluation of distributed systems, computer networks. Some of his research results have been technology transferred to industrial sectors and used in product design. So far, he has been a chief project leader of more than 10 commercial products. Some of these products are widely used around the world. He has been received both research awards and teaching awards from various organizations in Taiwan and serves as a committee member in several academic and industrial organizations.