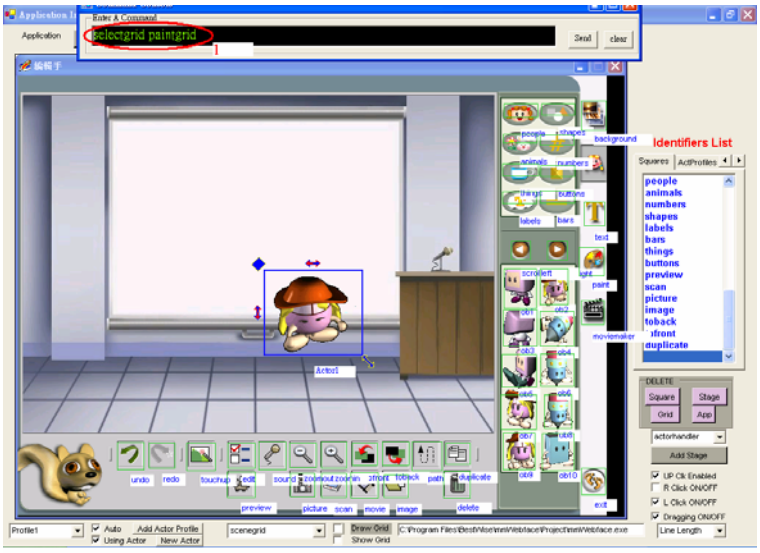

Figure 45(a). **Dragging an Actor**

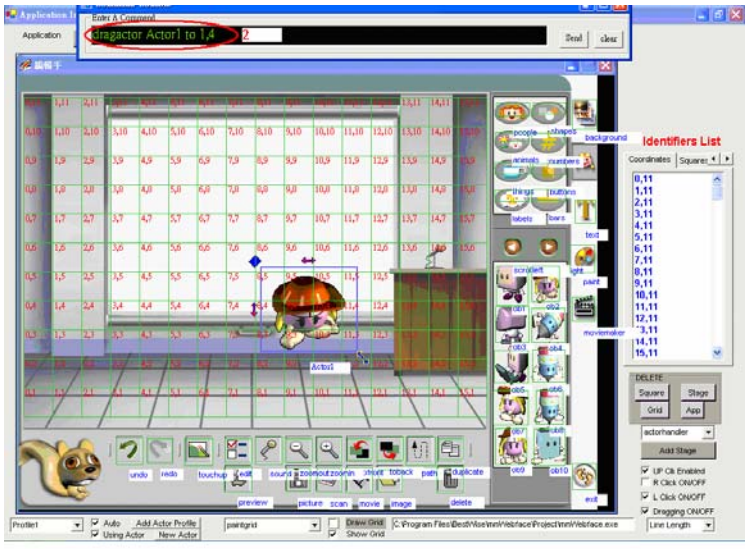

Figure 46(b). **Dragging an Actor**

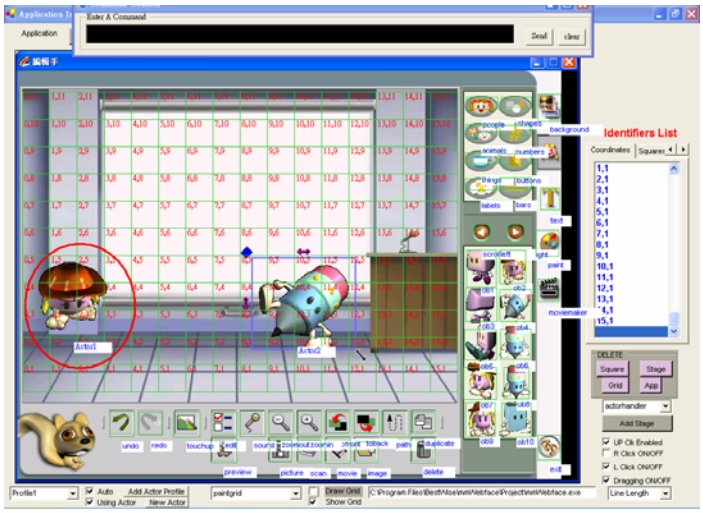

Figure 47(c). **Dragging an Actor**

Table 8 presents the description of the steps involved in dragging an actor.

Table 8. **Dragging Actor**

Steps	**High-Level View**	**Low-Level View**
Select Grid	End user speaks the “selectgrid” command to select a grid	Triggers an invocation through the *Event Driven GUI* component to the *Square Managing* class, this class invokes a factory method inside the *Square Mapping Mechanism* component that reads each square from file and returns a collection of square objects that belong to the grid. Back in the *Square Managing* class, each square from the collection get drawn on screen
Drag actor	End user speaks the “dragactor Actor1 to 1,4” command	System invokes the *MouseAI* component through the *Event Driven GUI* component, setting the mouse interaction mode to “movement”, then the *ActorHandler* component is invoked to retrieve the selected Actor’s location. The cursor’s location is then set to that of the Actor through the *MouseAI* component. The *ActorHandler* component then stores the new coordinates of the actor, by querying the *StoreHandler* component to submit the updated actor information as described in **5.6.1**. The *MouseAI* is finally invoked to set its mode to “dragging”, and the new coordinates are used to set the cursor to the new position

5.6.3 Utilizing Wildcards

Wildcards are part of the system’s design strategy to allow the reutilization of a macro with different

entities (Actors) by allowing the user to assign the wildcard value during runtime. To utilize wildcards one must first focus on an actor through the “focus command” and then invoke the macro interaction command, the system will take care of replacing wildcards with the focused actor.

Figure 41 depicts a snapshot of the events involved in utilizing a macro that contains wildcards.

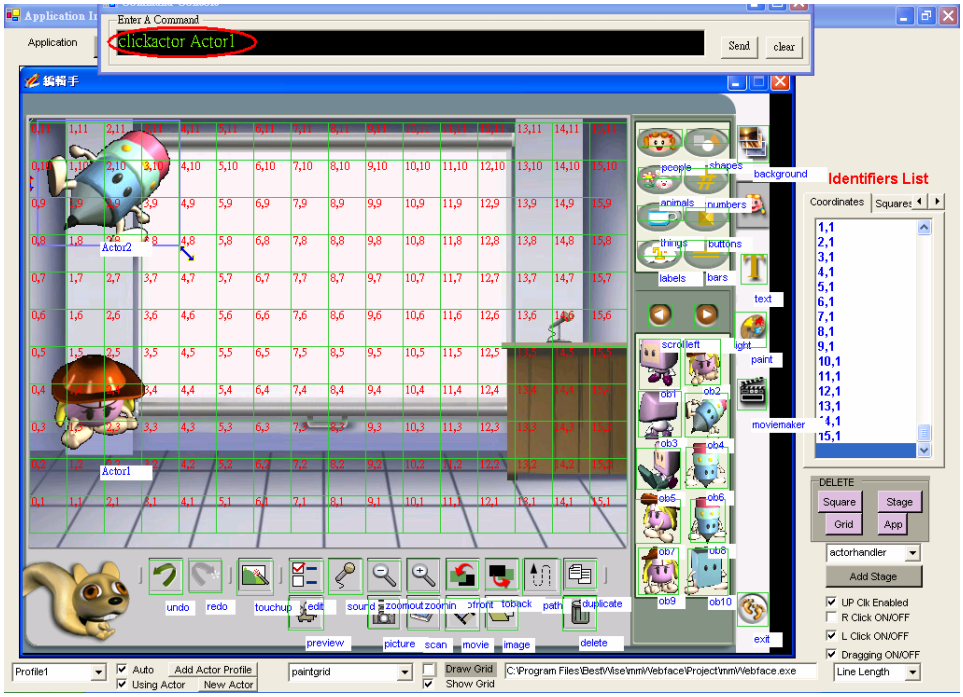

Figure 48. **Utilizing Wildcards**

Table 9 presents the description of the steps involved in loading a macro with wildcards and assigning them a specific actor.

Table 9. **Utilizing Wildcards**

Steps	**High-Level View**	**Low-Level View**
Focus Object	End User speaks the “Focus Actor1” Command	The system gains awareness of the chosen object.
Invoke Wildcard Command	End User speaks a Macro Command containing wildcards	The *Macro Interpreter* translates the keyword into valid commands defined in the system’s language. Wildcards found inside the commands are replaced by the value specified by the user (Actor1) by the *Wildcard Translator*

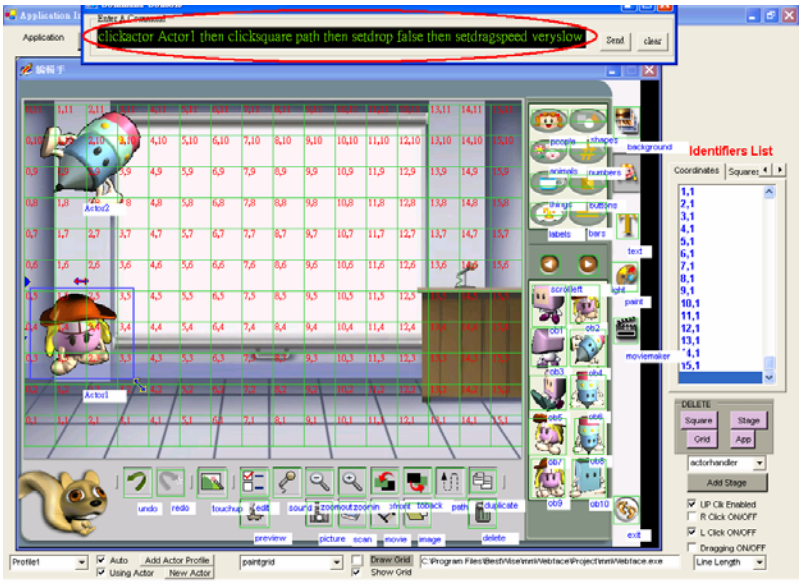

Figure 49. **Utilizing a Macro**

In this example the macro command Pattern1 is spoken, that translates to:
"*clickactor Actor1 then clicksquare path then setdrop false then setdragspeed veryslow then dragactor Actor1 north by verylong then setdragspeed fast then dragactor Actor1 northeast then setdragspeed slow then dragactor Actor1 east then loop 2 then setdrop true then setdragspeed veryfast then dragactor Actor1 by octagon by short*"

The macro command discussed for this example draws a path for the focused actor. It does so by setting the speed of dragging (by invoking the *mouseAI* in the same manner described in **5.6.2**). Then it uses the "dragactor" command to drag the actor in a specific direction and at a specific distance.

5.6.4 Utilizing the Capturing Method

To capture external application's windows into our system's main container the user must invoke the "captureit" command. Figures 43 and 44 depict snapshots of the usage of the capturing method that is incorporated into a macro command. The operation steps are labeled as 1, 2 and 3 in the figures.

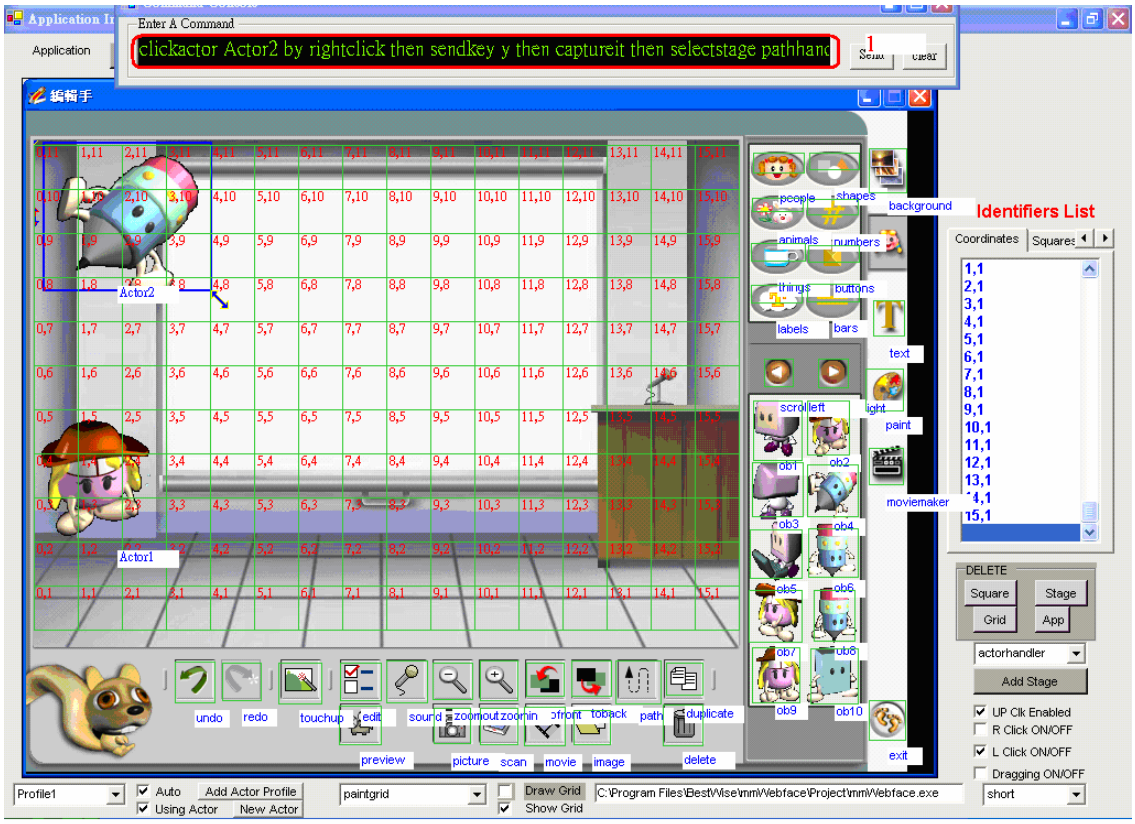

Figure 50. **Defining a Path through Capture**

Table 10 presents the description of the steps involved in defining a path of an actor through the utilization of the capture method.

Table 10. **Defining a Path through Capture**

Steps	**High-Level View**	**Low-Level View**
Focus Object	End User speaks the “Focus Actor2” Command	The system gains awareness of the chosen object.
Invoke Wildcard Command	End User speaks the macro Command “definepath”.	The *Macro Interpreter* translates the keyword into valid commands defined in the system language. Wildcards found inside the commands are replaced by the value specified by the user (Actor2) by the *Wildcard Translator*

“definepath” Macro Command translation:

“*clickactor Actor2 by rightclick then sendkey y then captureit then selectstage pathhandler then selectgrid patterngrid*”

We present the low level view of the execution of the translated command as below:

Triggers an event of type right-click on the focused actor and uses the API function calls of "sendkey" and "captureit" that are invoked through the "*Event-Driven GUI*" component.

The sendkey command emulates the stroke of the key Y, popping up an application menu, then using the "captureit" command to capture the external window inside our system's container. The "selectStage" command is then called to load the corresponding stage that corresponds to the current section of the target application. The *SquareHandler* mechanism loads all the corresponding squares from file(located inside the stage directory) for this stage. The squares are loaded and their respective label is instantiated.

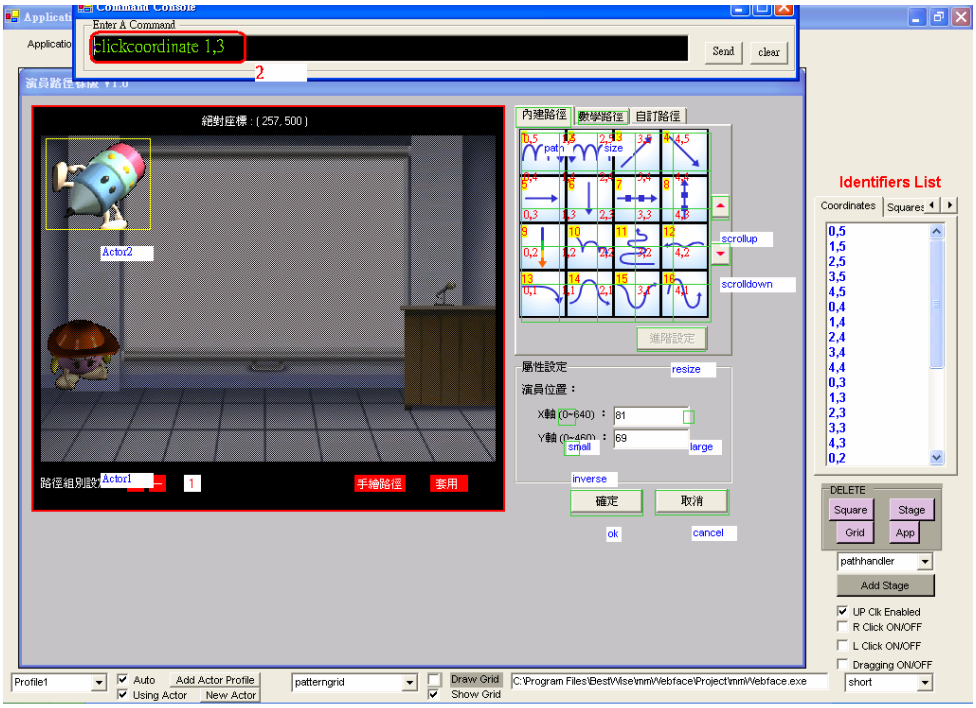

Figure 51. **Capturing Screen**

Table 11 presents the continuation of the description of the steps involved in defining a path of an actor through the utilization of the capture method.

Table 11. **Defining a Path, Continued**

Steps	**High-Level View**	**Low-Level View**
Click Coordinate	End User speaks the "clickcoordinate 1.3" command to select a moving pattern	The system performs a click on the coordinate specified in a similar way as described in **5.6.2**
Accept	End User speaks the "clicksquare ok" command to accept the previously selected pattern and exiting the stage.	The system performs a click on the square specified in a similar way as described in **5.6.2**

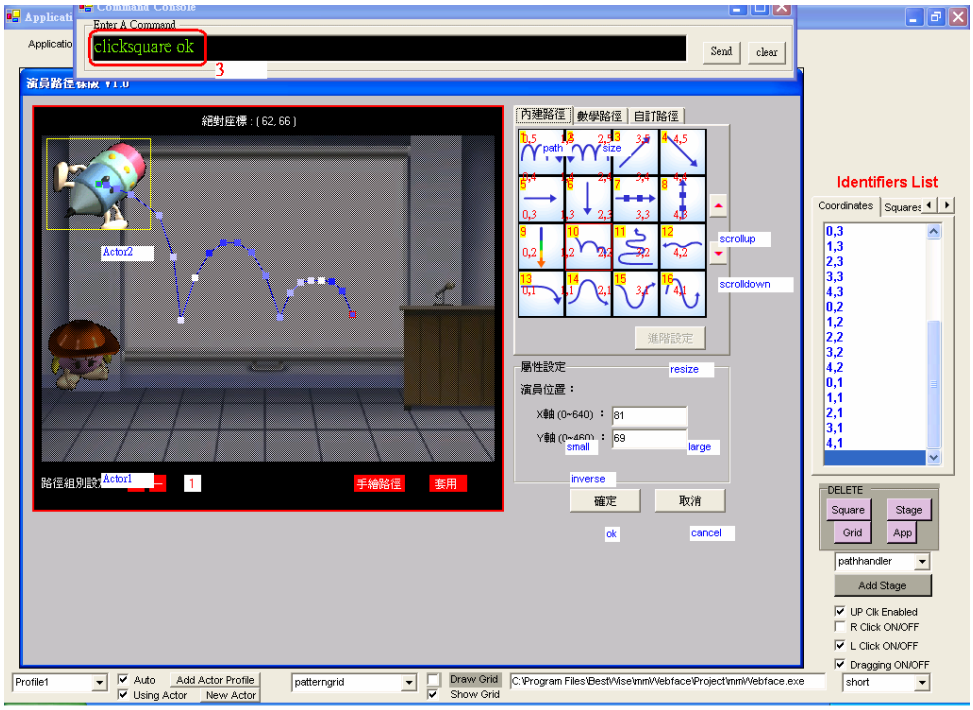

Figure 52. **Defining a Path, Continued**

5.6.5 Interacting With the Painting Mode

Figures 46(a) 46(b) 46(c) depict a snapshot of basic interaction through a grid and predefined patterns that can be used for painting panes. To enter the painting mode of the target interfaced application, we first invoke the commands to load the corresponding stage and grids of the painting GUI. We then draw a figure by invoking commands that perform moving patterns as the mouse is dragged.

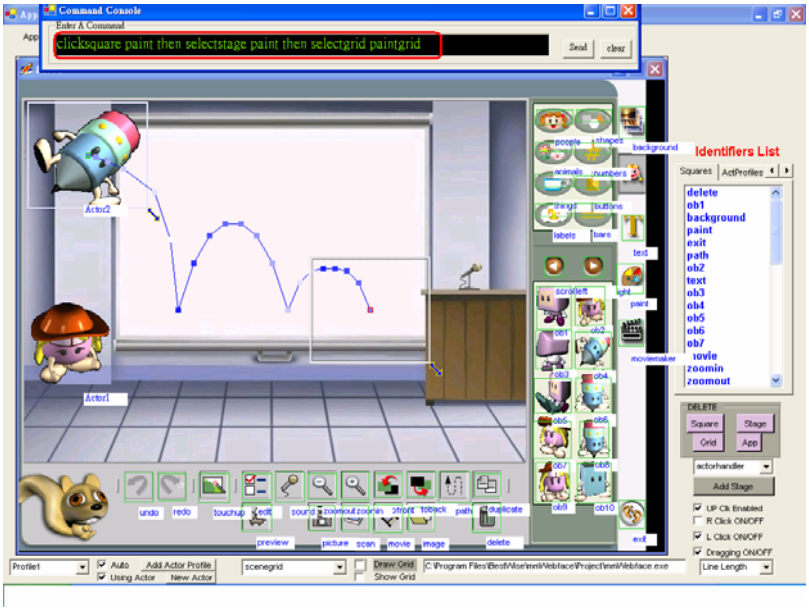

Figure 53(a). **Interacting with the Paint Mode**

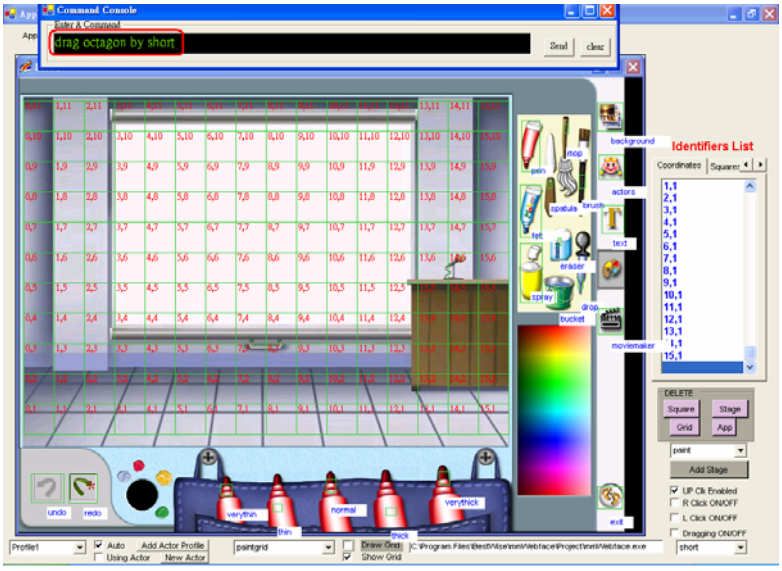

Figure 54(b). **Interacting with the Paint Mode**

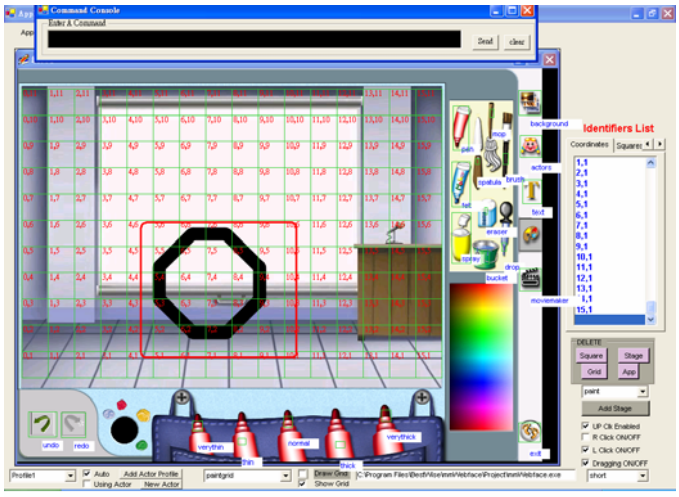

Figure 55(c). **Interacting with the Paint Mode**

Table 12 presents the description of the steps involved in interacting with the paint mode of the target application

Table 12. **Paint Mode Interaction**

Steps	High-Level View	Low-Level View
Select Painting Stage	End user speaks the “selectstage paint” macro command	The command gets translated into the systems internal language. Then it triggers an event of type click on the square labeled “paint”, selects the “paint” stage and the “paintgrid” grid.
Draw an image	The “drag octagon by short” command is spoken by the end user, triggering the drag command to take place in the directions that are involved in tracing an octagon	Invocations take place at the “*MouseAI*” component in a similar way as in **5.6.2**.

5.6.6 Dragging Objects Referenced by Squares and Coordinates

Figures 47(a) 47(b) depict a snapshot of the steps involved in dragging an object that is referenced by a square to a zone that is also referenced by a square, consisting of the user invoking a “dragsquare” command.

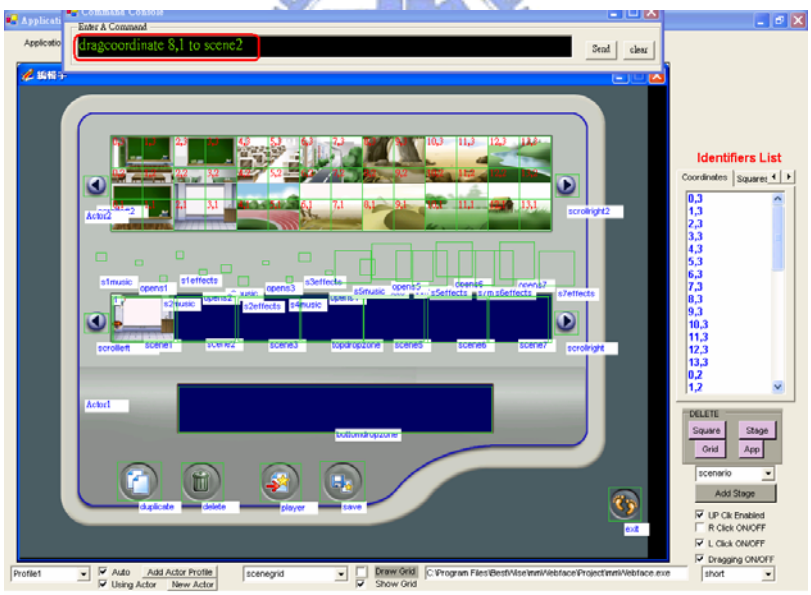

Figure 56(a). **Dragging Objects Referenced by Squares and Coordinates**

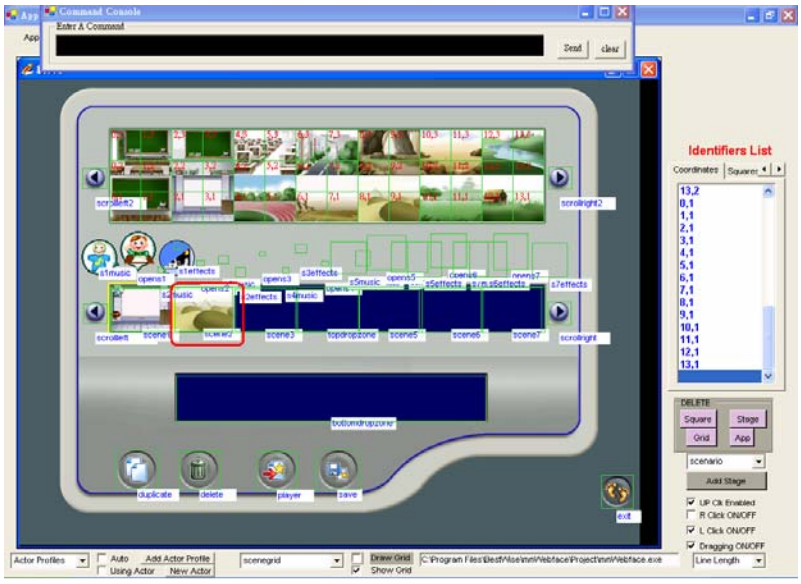

Figure 57(b). **Dragging Objects Referenced by Squares and Coordinates**

Table 13 presents the description of the steps involved in square to square dragging

Table 13. **Square to Square Dragging**

Steps	**High-Level View**	**Low-Level View**
Select a source and sink	End user speaks the command "dragcoordinate 8,1 to scene2"	The cursor is moved to the respective coordinate, dragging mode is activated, dragging is performed to the destination("Scene2"). All this is done by making invocations to the "*Square Mapping Mechanism"* to get the coordinates of both the squares and invocations to the "*Mouse AI"* component to perform the involved mouse actions in a similar way of what is done in **5.6.2**

5.7 Evaluation

We attempt to evaluate the proposed system from two general aspects: 1) "what to be evaluated" and "how to evaluate it". In "what to be evaluated", we focus on evaluating the common application challenges and limitations of current approaches identified in Chapter 1. In "how to evaluate it", we use a qualitative analysis against those metrics identified in the "what" part, by marking the features supported in our approach that are used to tackle these challenges. In the following subsections, we elaborate these analyses.

5.7.1 What to be evaluated

We evaluate how our approach enhances the process involved in interfacing an application with a recognizer by taking into account the characteristics of current interfacing methods with their challenges and limitations by working a way around them.

We evaluate the following key issues:

- **Generic Interfacing:**
  - We evaluate our approach in terms of its flexibility to interface with more than one application environment and also how its interfacing GUI is designed to achieve this purpose.
- **Complexity:**
  - We evaluate our approach in terms of the level of difficulty involved in interfacing application software with a speech recognizer.
- **Customization:**
  - We evaluate the overhead involved in performing future modifications to an interfacing environment of an application.
- **Efficiency:**
  - We evaluate our approach in terms of its capabilities to abstract a group of actions to simplify user interaction with target applications.

5.7.2 How to evaluate it

To evaluate our approach, we interfaced an application software[23] with a speech recognizer[17] by referring the application's interaction zones, then creating a grammar definition for the speech engine to recognize theses interfacing environment's zones, also by composing macros to abstract a set of actions, and finally by interacting with the interfaced application through our framework, all of the above by following the interfacing steps described by the scenarios that are covered earlier in this chapter. Based on this we present and evaluate the features supported in our approach by contrasting them with the challenges and limitations that are suffered by current interfacing approaches. These features are summarized bellow:

- **Generic Interfacing.**

We address non-generic recognizer interfacing by creating a generic, application-independent, recognizer-driven interface generator framework that allows the creation of interfacing visual environments that fit multiple applications. As portrayed in the interfacing composition scenarios discussed in this chapter, our framework allows for custom making of interfaces for applications by

drawing referencing objects such as squares, grids and moreover provides dynamic interaction capabilities through the registration of actors.

- **Complexity.**

We address complexity by interfacing applications through their front-end by adopting the "See-Through Interface" paradigm, allowing a visual interfacing mechanism that does not deal with low level system design and implementation issues.

- **Customization.**

We address customization issues by adopting the "See-Through Interface" paradigm that allows front-end visual customization of the interfacing context during run-time without affecting other application's recognition interfaces and without the need of dealing with source code re-compilation each time changes are made.

- **Efficiency.**

We address interaction inefficiency by defining and integrating a script language into the system and providing a post-interfacing mechanism to abstract sets of actions into single, context-free reusable macros as depicted in the interaction scenarios of this chapter. Macros are defined with the use of simple reference words, allowing the user to have a less-intense and time consuming interaction with the system by minimizing the amount of commands that he/she must speak, providing a more natural way of speaking and enhancing speech-recognition efficiency.

5.7.3 Evaluation Results

In this section we summarize our evaluation results and provide a contrast with the challenges and limitations of current approaches through a comparison table bellow:

Table 14 summarizes the system evaluation in terms of the application challenges.

Table 14. **Comparison of our System against Application Challenges**

Challenge	Interfacing Interface Framework	Current Interfacing Approaches
Non-Generic Interfacing	Offers a generic interfacing environment that can be used to interface a recognizer simultaneously with multiple applications that require different visual interfacing requirements.	Current approaches focus on programming a direct interfacing of one application with one speech recognizer. Current approaches lack a graphic interfacing environment that interfaces application's interaction zones through drawing.
Complexity	Allows interfacing a recognizer with 3rd party applications through a visual environment that interfaces applications through the front end, without the need of accessing their code.	Current approaches focus on recognizer integration through the back-end of applications requiring low-level programming and system design knowledge.
Non-Customizable	Allows modifications to a visual interfacing environment to be done at run-time, without the need of compilation of any source code.	Current approaches' tightly coupled system design does not allow the customization of the interaction environment by the user. Modification of application's interfacing environment requires re-compilation of source code.
Inefficiency	Our approach offers a script language for users to interact directly with the system in real-time and that allows the composition of context-free reusable macros to simplify user interaction and increase speech recognition accuracy.	Lack of a post-interfacing mechanism to abstract a group of actions into single composed commands to minimize and simplify user interaction tasks.

The proposed system achieved the vision of Interface Interfacing by providing a new method to interface applications with a recognizer without requiring programming knowledge by the users that perform the interfacing that can be applied to commercial applications without the need of accessing their internal code, and also allows the composition of macros to facilitate interaction task, in this way overcoming common limitations and challenges of current approaches.

5.8 Conclusion

After an extensive qualitative evaluation, it was established that the proposed system generally achieves its goals well, in terms of supporting the vision of Interface Interfacing and enhancing application interfacing to recognizers by addressing current application challenges. However, some limitations were identified, such as the lack of interfacing context switching agents in charge of dynamically setting up the interfacing environment for the user as this navigates the target application as well as the requirement of interfacing recognizers into the framework through API coding instead of applying a further abstracted method. In overall, this chapter provided the visualization on how our system tackles the challenges and enhances the limitations that are suffered by current interfacing approaches.

CHAPTER SIX
Conclusions and Future Work

6.1 Conclusion and Major Contributions of this Research

This research alleviates some common problems suffered by developers when bridging an application to the interface of a recognizer. The proposed approach presents a more flexible and efficient interfacing compared with the current approaches. 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,” design patterns, and a script language definition plus a parsing technique to carry out the system design and implementation. The proposed Interface Interfacing Framework enhances the interfacing of applications to recognizers by making it an easy, generic and flexible process.

We also investigated and identified several challenges for developing and deploying successful applications in speech-recognition interfacing environments. Some specific criteria that it’s addressed from application-centric viewpoints are established. We studied current technologies that have a potential to overcome individual challenges and limitations suffered by current interfacing approaches. We designed and implemented a system that incorporates these technologies in a cooperative way to tackle the limitations and challenges of current solutions. We evaluated the system qualitatively in terms of its functionality against the criteria concluded from the challenges and limitations of current approaches, and how it supports the vision of Interface Interfacing and how it enhances the interfacing process. The major contributions of this research include:

- 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.
- Allows modifications to a recognition interfacing environment of an application without requiring the access to source code of applications and re-compilation of it.
- Offers a generic and custom interface interfacing environment that allows the coexistence of multiple applications that hold different interfacing requirements.
- 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.

6.2 Future Work

Many challenges that were not addressed completely in this study still exist in the implementation of Interface Interfacing systems. The following are the major limitations of our proposed system:

- **Non-dynamic Switching of Interfacing Content for Applications**
  - User must invoke the loading of interfacing content as he/she navigates through the different sections of an application through vocal commands

- **No plug-in Recognizer Interfacing**
  - Do to the current design of recognizers, one must program a recognizer's Integration through calls on the recognizer's API

- **Limited Tracking of Dynamic Application Content**
  - Since we take a Front-End integration approach, tracking with dynamic content of applications is limited to user specified movement of actors. Whenever an application auto-positions its actors (through animations), our system is unable to efficiently provide actor tracking but however allows the user to re-locate actors that are misplaced

The complexity of most applications, demand a highly organized and smart interfacing framework in order to interact with them. Work on the later is ongoing and further investigations need to be carried out in both the research and development areas, specifically in developing smart agents in charge of switching interfacing content for applications dynamically and transparently to the user as their GUI are navigated. Development of agents that compose grammar definitions for recognition devices dynamically as the user register interfacing zones, eliminating the need of the administrator user to perform such low level tasks. Interaction with dynamic content of applications is still needed to be enhanced as this study only attempt to provide a solution that will facilitate this kind of interaction but however leaving the doors open for enhancing opportunities, especially in the areas of smart tracking mechanisms and dynamic boundary delimiters.

The Interface Interfacing field of study is still premature but it has a very promising future that will change the way of integrating recognition capabilities to 3rd party applications, allowing fast interfacing with recognition capabilities and detaching users from intense and non-natural interaction. More importantly eliminating the complicated and time-consuming processes of recognition integration to applications.

DEFINITION OF TERMS

GUI A graphical user interface (or GUI, pronounced "gooey") is a method of interacting with a computer through a metaphor of direct manipulation of graphical images and widgets in addition to text.

CFG Context Free Grammar is used for definition of languages

BNF Backus-Naur Form, is the most common notation used to express context-free grammars.

API Application Program Interface. The interface (calling conventions) by which an application program accesses operating system and other services. An API is defined at source code level and provides a level of abstraction between the application and the kernel (or other privileged utilities) to ensure the portability of the code.

SDK Software Development Kit is software provided by a software vendor to allow their products to be used with those of other software vendors.

OS An Operating System is The low-level software which handles the interface to peripheral hardware, schedules tasks, allocates storage, and presents a default interface to the user when no application program is running.

IE Internet Explorer is a popular web-browser developed by Microsoft

ASCII American Standard Code for Information Interchange is the basis of character sets used in almost all present-day computers. US-ASCII uses only the lower seven bits (character points 0 to 127) to convey some control codes, space, numbers, most basic punctuation, and unaccented letters a-z and A-Z.

URL Uniform Resource Locator is a standard way of specifying the location of an object, typically a web page, on the Internet.

XML XML is a markup language for documents containing structured information. Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption or content in a database table, etc.). Almost all documents have some structure. A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents.

REFERENCE APPENDIX

[1] *B. Balentine, D. Morgan, and W. Meisel.* How to Build a Speech Recognition Application, *Enterprise Integration Group*, 1999.

[2] Speech-Actuated Manipulator. Available: http://www.research.att.com/history/89robot.html

[3] VSpeech 1.0, Team BK02 product. Available: http://vspeech.sourceforge.net

[4] Voxx 4.0, Voxx Team product. Available: http://voxxopensource.sourceforge.net/

[5] IVOS 2.0.1, ComunX product. Available: http://ivos.comunx.com/

[6] *Eric A. Bier, Maureen C. Stone, K. Pier, W. Buxton Tony 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, 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.*

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

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

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

[11] *C.S. Koong,* "A Component-based Visual Scenario Construction Environment for Non-Programming Users to Create Interactive Electronic Books"; *A Masters Deegre Thesis, Computer Science and Information Engineering, National Chiao-Tung University, Taiwan, 2002.*

[12] *James Gosling,Bill Joy,Guy Steele,Gilad Bracha* ; The Java Language Specification, *Second Edition, Sun Microsystems, Inc., 2000.*

[13] *Robert W. Sebesta* ; Concepts of programming languages, *Fifth Edition, Addison-Wesley Publishing Company , 2002.*

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

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

[16] Microsoft Speech SDK, Version 5.1 Documentation, *Microsoft Corporation, 2001.*

[17] Microsoft's Speech Recognizer V.6.1, Microsoft product.

Available: http://www.microsoft.com

[18] *Bruce Powel Douglas*; Real-time design patterns: robust scalable architecture for Real-time systems, *Addison-Wesley Publishing Company, 2003.*

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

[20] *Shih-Kun Huang,* "Objected-Oriented Program Behavior Analysis Based on Control Patterns";*a Ph. D. 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] BestWise International Computing Company. Available: http://www.caidiy.com.tw

[25] *W. C. Chen,* "A Reuse-based Software Construction Paradigm for Visualized Reusable Components and Frameworks"; *a Ph. D. 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://visualbasicforum.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] Voice Recognition Systems. Available: http://talktoyourcomputer.com

APPENDIX I

SPEECH ENGINE GRAMMAR DEFINITION

```
<GRAMMAR LANGID="409">

<DEFINE>
<ID NAME="NUM" VAL="1"/>
<ID NAME="VISLIST" VAL="3"/>
<ID NAME="BOOLEAN" VAL="4"/>
<ID NAME="GRIDS" VAL="5"/>
<ID NAME="STAGES" VAL="6"/>
</DEFINE>

<!--Runtime Defined variables-->
<RULE NAME="Actor">
        <P>Actor<RULEREF NAME="num"/></P>
    </RULE>

<RULE NAME="Profile">
        <P>Profile</P>
<RULEREF NAME="num"/>
    </RULE>

<!--Other Variables-->
<RULE NAME="boolean">
<L PROPNAME="boolean" PROPID="BOOLEAN">
  <P>true</P>
<P>false</P>
</l>
    </RULE>

<RULE NAME="spd">
<l>
   <P>veryslow</p>
   <P>slow</p>
   <P>fast</p>
   <P>veryfast</p>
</l>
</RULE>

<!--visuallist-->
<RULE NAME="vislist">
          <L PROPNAME="vislist" PROPID="VISLIST">
                <P>actorlist</p>
  <P>stagelist</p>
  <P>gridlist</p>
  <P>applicationlist</p>
```

```
  <P>actorprofilelist</p>
  <P>squarelist</p>
</L>
</Rule>


<!--Commands-->
<RULE NAME="showgrid" TOPLEVEL="ACTIVE">
            <P>showgrid</p>
<o>
        <RULEREF NAME="boolean"/>
</o>
<o>
<RULEREF NAME="then"/>
</o>
    </RULE>

<RULE NAME="setdistance" TOPLEVEL="ACTIVE">
            <P>setdistance</P>
<o>
              <P>to</P>
        <RULEREF NAME="mse"/>
</o>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="addactor" TOPLEVEL="ACTIVE">
            <P>addactor</P>
<o>
              <P>to</P>
<o>
<RULEREF NAME="coord"/>
</o>
</o>
<o><RULEREF NAME="then"/>
        </o>

<o>
<RULEREF NAME="sqrs"/>
</o>
    </RULE>

<RULE NAME="setactorbutton" TOPLEVEL="ACTIVE">
            <P>setactorbutton</P>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>
```

```
<RULE NAME="setdrop" TOPLEVEL="ACTIVE">
            <P>setdrop</P>
        <RULEREF NAME="boolean"/>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="setdragspeed" TOPLEVEL="ACTIVE">
            <P>setdragspeed</P>
<o>
        <RULEREF NAME="spd"/>
</o>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="setvisualaide" TOPLEVEL="ACTIVE">
            <P>setvisualaide</P>
<o>
        <RULEREF NAME="boolean"/>
</o>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="addactorprofile" TOPLEVEL="ACTIVE">
            <P>addactorprofile</P>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="pastetext" TOPLEVEL="ACTIVE">
            <P>pastetext</P>
<o>
        <l>
              <RULEREF NAME="abc"/>
        <RULEREF NAME="num"/>
          <p>*+</p>
</l>
</o>
<o><RULEREF NAME="then"/>
        </o>
</RULE>

<RULE NAME="cleartext" TOPLEVEL="ACTIVE">

            <P>cleartext</P>
        <o>
        <RULEREF NAME="then"/>
        </o>
```

```
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="captureit" TOPLEVEL="ACTIVE">
            <P>captureit</P>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="sendkey" TOPLEVEL="ACTIVE">
            <P>sendkey</P>
<o>
<RULEREF NAME="abc"/>
<RULEREF NAME="num"/>
</o>
<o><RULEREF NAME="then"/>
        </o>
</RULE>

<RULE NAME="usingactor" TOPLEVEL="ACTIVE">
            <P>usingactor</P>
        <RULEREF NAME="boolean"/>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="usedelay" TOPLEVEL="ACTIVE">
            <P>usedelay</P>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="setactor" TOPLEVEL="ACTIVE">
            <P>setactor</P>
<o>
<RULEREF NAME="Actor"/>
<P>to</P>
<l>

        <RULEREF NAME="coord"/>
        </l>
</o>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="resetactor" TOPLEVEL="ACTIVE">
            <P>resetactor</P>
<o><RULEREF NAME="then"/>
```

```
        </o>
    </RULE>

<RULE NAME="setlist" TOPLEVEL="ACTIVE">
            <P>setlist</P>
        <RULEREF NAME="vislist"/>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="eraseactor" TOPLEVEL="ACTIVE">
            <P>eraseactor</P>
        <RULEREF NAME="Actor"/>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="eraseprofile" TOPLEVEL="ACTIVE">
            <P>eraseprofile</P>
<o>
<RULEREF NAME="Profile"/>
</o>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="opendeveloper" TOPLEVEL="ACTIVE">
            <P>opendeveloper</P>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="drag" TOPLEVEL="ACTIVE">
            <P>drag</P>
<o>
<RULEREF NAME="mse"/>
<o>
<p>by</p>
<RULEREF NAME="mse"/>
</o>
</o>
<o><RULEREF NAME="then"/>
        </o>
</RULE>

<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>
        <o>
        <o>
        by
        </o>
        <RULEREF NAME="mse"/>
        <o>
        <p>by</p>
        <RULEREF NAME="mse"/>
        </o>
        </o>
        </o>
<o><RULEREF NAME="then"/>
        </o>
</RULE>

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

<RULE NAME="dragactor" TOPLEVEL="ACTIVE">
            <P>dragactor</P>
        <o>
```

```
        <RULEREF NAME="Actor"/>
        <o>
        <p>to</p>
        <l>
        <RULEREF NAME="sqrs"/>
        <RULEREF NAME="coord"/>
        </l>
        </o>
        <o>
        <o>
        by
        </o>
        <RULEREF NAME="mse"/>
        <o>
        <p>by</p>
        <RULEREF NAME="mse"/>
        </o>
        </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>

<RULE NAME="clicksquare" TOPLEVEL="ACTIVE">
            <P>clicksquare</P>
<RULEREF NAME="sqrs"/>
<o>
<p>by</p>
<RULEREF NAME="mse"/>
</o>
<o><RULEREF NAME="then"/>
        </o>
</RULE>

<RULE NAME="clickactor" TOPLEVEL="ACTIVE">
            <P>clickactor</P>
<RULEREF NAME="Actor"/>
<o>
<p>by</p>
<RULEREF NAME="mse"/>
```

```
</o>
<o><RULEREF NAME="then"/>
        </o>
</RULE>

<RULE NAME="clickcoordinate" TOPLEVEL="ACTIVE">
              <P>clickcoordinate</P>
<RULEREF NAME="coord"/>
<o>
<p>by</p>
<RULEREF NAME="mse"/>
</o>
<o><RULEREF NAME="then"/>
        </o>
</RULE>

<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="stop" TOPLEVEL="ACTIVE">
              <P>stop</P>
    <o><RULEREF NAME="then"/>
        </o>
</RULE>

<RULE NAME="selectstage" TOPLEVEL="ACTIVE">
              <P>selectstage</P>
        <RULEREF NAME="stages"/>
  <o><RULEREF NAME="then"/>
        </o>
 </RULE>
```

```
<RULE NAME="selectgrid" TOPLEVEL="ACTIVE">
            <P>selectgrid</P>
        <RULEREF NAME="grids"/>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="selectactorprofile" TOPLEVEL="ACTIVE">
            <P>selectactorprofile</P>
        <RULEREF NAME="Profile"/>
<o><RULEREF NAME="then"/>
        </o>
    </RULE>

<RULE NAME="selectapplication" TOPLEVEL="ACTIVE">
            <P>selectapplication</P>
<RULEREF NAME="apps"/>
<o><RULEREF NAME="then"/>
        </o>
</RULE>

<!--Operators-->
  <RULE NAME="from" TOPLEVEL="ACTIVE">
            <P>from</P>
    <o><RULEREF NAME="then"/>
        </o>
</RULE>

<RULE NAME="loop" TOPLEVEL="ACTIVE">
            <P>loop</P>
        <o><RULEREF NAME="num"/>
        <o><RULEREF NAME="times"/>
        </o>
</o>
<o><RULEREF NAME="then"/>
        </o>
</RULE>

<!--Terminators-->
<RULE NAME="times">
            <P>times</P>
    </RULE>

<RULE NAME="then" TOPLEVEL="ACTIVE">
            <P>then</P>
    </RULE>

<!--SOUND RECOGNITION COMMANDS-->
<RULE NAME="send" TOPLEVEL="ACTIVE">
            <P>send</P>
```

```
    </RULE>

<RULE NAME="undofrase" TOPLEVEL="ACTIVE">
            <P>undofrase</P>
    </RULE>

<RULE NAME="undoall" TOPLEVEL="ACTIVE">
            <P>undoall</P>
    </RULE>

<!--Coordinates-->
<RULE NAME="coord">
<RULEREF NAME="num"/>
<P>,</P>
<RULEREF NAME="num"/>
</Rule>

<!--numbers-->
<RULE NAME="num">
        <L PROPNAME="num" PROPID="NUM">
<P>0</P>
<P>1</P>
<P>2</P>
<P>3</P>
<P>4</P>
<P>5</P>
….<P>100</P>
</L>
</RULE>

<!--abc-->
<RULE NAME="abc" TOPLEVEL="ACTIVE">
        <L>
<P>a</P>
<P>b</P>
<P>c</P>
<P>d</p>
….<P>x</p>
<P>y</p>
<P>z</p>
</L>
</RULE>

<!-- applications -->
<RULE NAME="apps">
<l>
  <p>mmWebface</p>
</l>
</rule>
```

```
<!-- mouse constants-->

<RULE NAME="mse">
        <L>
            <P>leftclick</P>
            <P>rightclick</P>
            <P>doubleclick</P>
            <P>veryshort</P>
            <P>short</P>
            <P>normal</P>
            <P>long</P>
            <P>verylong</P>
            <P>north</P>
            <P>south</P>
            <P>west</P>
            <P>east</P>
            <P>northeast</P>
            <P>northwest</P>
            <P>southeast</P>
            <P>southwest</P>
            <P>octagon</P>
            <P>hexagon</P>
            <P>pentagon</P>
            <P>square</P>
            <P>triangle</P>
            <P>zigzag</P>
            <P>curves</P>
            <P>spiral</P>
</L>
</RULE>

<!--BELLOW IS ALL APPLICATION DEPENDANT, SO ITS THE ONLY
PART OF THIS DEFFENITION THAT SHOULD BE MODIFIED-->

<!--variables for use with outsorting tool-->
<RULE NAME="sqrs">
<L>
</L>
</RULE>

<!-- stages -->
<RULE NAME="stages">
        <L PROPNAME="stages" PROPID="STAGES">
</L>
</RULE>

<!-- grids -->
<RULE NAME="grids">
        <L PROPNAME="grids" PROPID="GRIDS">
```

</L>
</RULE>

<!--Composed Commands
Following are key words that when recognized will trigger the execution of 'macros' based on the defined language-->

<!--DYNAMICINSERTIONFLAG-->

<!--DYNAMICINSERTIONZONE ENDS-->

</GRAMMAR>

APPENDIX II

SYSTEM'S LANGUAGE BNF DEFINITION

<Statement_Sequence> ::= <Statement> | <Statement> <then> <Statement> | <Statement_Sequence>
< Statement > ::= <setDistance> | <setDragSpeed> | <addActorProfile> | <eraseProfile> |
< addActor> | <eraseActor> | <setActor> | <resetActor> | <setDrop> | <setList> |
<setVisualAide> | <showGrid> | <usingActor> | <drag> | <dragSquare>| <dragCoordinate> |
<dragActor> | <click> | <clickSquare> | <clickActor> | <clickCoordinate> | <move> |
<pasteText> | <clearText> | <sendKey> | <loop> | <openDeveloper>| < captureIt> | <selectStage> |
<selectGrid> | <selectActorProfile> | <selectApplication>

<setDistance> ::= SETDISTANCE <to> <Distance>
<setDragSpeed> ::= SETDRAGSPEED <Speed>
<addActorProfile> ::= ADDACTORPROFILE
<eraseProfile> ::= ERASEPROFILE <ActorProfile>
<addActor> ::= ADDACTOR [<to> <Coordinate>]
<eraseActor> ::= ERASEACTOR <Actor>
<setActor> ::= SETACTOR <Actor> <to> <Coordinate>
<resetActor> ::= RESETACTOR
<setDrop> ::= SETDROP <Boolean>
<setList> ::= SETLIST VisList
<setVisualAide> ::= SETVISUALAIDE <Boolean>
<showGrid> ::= SHOWGRID <Boolean>
<usingActor> ::= USINGACTOR <Boolean>
<drag> ::= DRAG <Direction>|<MovingPattern> <by> <Distance>
<dragSquare> ::= DRAGSQUARE <Square> (<Direction>|<by>|<to>)
<Square> | <Coordinate> | <Distance> | <MovingPattern> [<by> <Distance>]
<dragActor> ::= DRAGACTOR <Actor> (<Direction>|<by>|<to>)
<Square> | <Coordinate> | <Distance> | <MovingPattern> [<by> <Distance>]
<dragCoordinate> ::= DRAGCOORDINATE <Coordinate> (<Direction>|<by>|<to>)
<Square> | <Coordinate> | <Distance> | <MovingPattern> [<by> <Distance>]
<click> ::= CLICK [<by> <Clicktype>]
<clickSquare> ::= CLICKSQUARE <Square> [<by> <Clicktype>]
<clickCoordinate> ::= CLICKCOORDINATE <Coordinate> [<by> <Clicktype>]
<clickActor> ::= CLICKACTOR <Actor> [<by> <Clicktype>]
<move> ::= MOVE (<Direction>|<to>) <Square>|<Coordinate>|Dystance
<pasteText> ::= PASTETEXT <String>
<clearText> ::= CLEARTEXT
<sendKey> ::= SENDKEY <letter>
<loop> ::= LOOP [<times>]
<openDeveloper> ::= OPENDEVELOPER
<captureIt> ::= CAPTUREIT
<selectStage> ::= SELECTSTAGE <Stage>
<selectGrid> ::= SELECTGRID <Grid>
<selectActorProfile> ::= SELECTACTORPROFILE <ActorProfile>
<selectApplication> ::= SELECTAPPLICATION <App>

<Identifier>::= <Zone>|<ConstantVal>|<Operators>|<Terminators>
<Zone> ::= <Stage>|<App>|<Square>|<Grid>|<Coordinate>|<Actor>|<ActorProfile>
<Actor> ::= ACTOR {<number>} <ActorProfile> ::= ACTORPROFILE {<number>}
<Stage> ::= {<string>|<number>} <App> ::= {<string>|<number>}
<Square> ::= {<string>|<number>} <Grid> ::= {<string>|<number>}
<Coordinate> ::= <number> ”,” <number>

<ConstantVal> ::= <VisList>|<Clicktype>|<Direction>|<Distance>|<Speed>|<MovingPattern>
<VisList> ::= <Constant> <Clicktype> ::= <Constant> <Direction> ::= <Constant>
<Distance> ::= <Constant> <Speed> ::= <Constant> <MovingPattern> ::= <Constant>
<Constant> ::= {<string> | <number>}

<Operators> ::= <to>|<by>
<to> ::=<Actor>|<Square>|<Coordinate> to <Square>|<Coordinate>|<Distance>
<by> ::= <Direction>|<Coordinate>|<Square>|<Actor> by <Distance>|<Clicktype>

<Terminators> ::= <then>|<times>
<then> ::= <Statement> then <Statement>
<times> ::= <number> times

<Boolean> ::= true|false
<number> ::= <digital> | <digital> <number>
<string> ::= <letter> | <digital> | <letter> <string> | <digital> <string>
<digital> ::= 0|1|2|3|4|5|6|7|8|9
<letter> ::= a|b|c|.....|y|z|A|B|C|.....|Y|Z