

HYPERSIGN: AN INTERACTIVE  
SIGN LANGUAGE DICTIONARY

by

HSING CHUNG LEE

M.S. Kansas State University, 1986

-----

A REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY  
Manhattan, Kansas

1989

Approved by:

  
Major Professor

LD  
2608  
R41  
CMS  
1989  
L44  
C.2

A11208 317664

## CONTENTS

1.	Introduction .....	1
2.	Background .....	4
2.1	Introduction .....	4
2.2	Features .....	6
2.2.1	Basic Units .....	6
2.2.2	Menus .....	7
2.3	HyperTalk .....	9
3.	Documentation .....	13
3.1	Environment .....	13
3.2	How To Use HyperSign .....	13
3.2.1	Searching .....	13
3.2.2	Adding .....	14
3.3	Overall Organization .....	15
3.4	Menu Stack .....	15
3.5	Sign Stack .....	20
3.6	Button Functions .....	21
3.7	Programming .....	22
3.7.1	Button scripts .....	23
3.7.2	Menu Scripts .....	25
4.	Discussion .....	29
4.1	HyperSign .....	29
4.2	Experience With HyperCard .....	31
4.3	HyperCard Limitations .....	33
4.3.1	Note .....	35
4.4	Conclusion .....	36
	References .....	37

## List Of Figures

Figure 1 HyperCard Hierarchy .....	10
Figure 2 Menu Card .....	16
Figure 3 Searching Card .....	17
Figure 4 Adding Card .....	18
Figure 5 Key Word Card .....	19
Figure 6 Sign Card .....	20

## INTRODUCTION

This report presents the implementation of an application called HyperSign. HyperSign is an interactive sign language dictionary in which one can find a specific sign corresponding to a key word. It is implemented in HyperCard, a user information management system. The objectives of building HyperSign are: (1) to have an interactive sign language dictionary with faster access time than traditional paper sign language dictionary. (2) to have an experience with HyperCard.

HyperCard is an user information management system for Macintosh computers. It is different from traditional programming environments or database management systems. It is a combination of graphic tools, a graphical and a text database management system, and a programming environment. The programming language is called HyperTalk. HyperCard can also connect to applications written in other programming languages.

This report consists of 4 chapters, labelled Introduction, Background, Documentation and Discussion. Some of the more important aspects of HyperCard will be introduced in the Background. The detailed description of HyperSign implementation is covered in the

Documentation. In the Discussion, I evaluate the implementation experience using HyperCard.

HyperSign was designed to be easy to use. The user can navigate almost entirely with a mouse. When searching for a particular sign, a user can click an Alphabet button



then choose a Key word button **BASEBALL**

and the sign image will be brought up.

**BASEBALL**



New key words and corresponding signs can be easily added without any knowledge of the HyperTalk programming language.

All of the sign images in HyperSign were digitized using a Macintosh compatible scanner and then pasted into HyperSign Cards.

The Initial implementation consists of 700 words, organized into 27 stacks; composed of one Menu stack, which is management center with about 200 lines of code, and 26 Alphabet stacks, which store the sign images. Each sign is about 2 by 2 inches, with a resolution of 72 dpi and a file size of 4 K with PICT format.

## BACKGROUND

A short introduction of HyperCard is presented in this chapter. The programming language HyperTalk is also discussed.

### 2.1 Introduction:

According to Bill Atkinson, the creator of HyperCard, HyperCard "is an authoring tool and an information organizer. You can use it to create stacks of information to share with other people or to read stacks of information made by other people. So it's both an authoring tool and sort of a cassette player for information." [2]. Apple calls it, "a personal toolkit that gives users the power to use , customize, and create new information using text, graphics, video, music, voice and animation" [1]. HyperCard is viewed by many people as a media, the so-called authoring tool, which combines text, graphics sound and interaction to express ones idea.

HyperCard is not just a database, wordprocessor or programming environment. HyperCard provides an interactive environment for text, graphics, sound, programming, and interactive buttons. The basic unit of HyperCard is a Card which is a scornful of information. A collection of Cards is a Stack, the file unit of HyperCard. HyperCard offers a powerful

programming language, HyperTalk. HyperTalk is interpreted. The syntax is similar to Pascal with a nice user interface. The programming system has some features found in object oriented programming languages. HyperTalk has objects, message passing, and event handlers but HyperTalk lacks the ability to create new class.

The concept of HyperCard is derived from Hypertext. Jeff Conklin [7] defined Hypertext as "Windows on the screen are associated with objects in a database, and links are provided between these objects, both graphically (as labelled tokens) and in the database (as pointers)". The implementation of HyperCard is very similar to the NoteCards system [8] developed at Xerox PARC in Xerox Lisp environment. NoteCards has been used for authoring, programming, personal information management, project management, and engineering design.

HyperCard was written by Bill Atkinson with support from Dan Winkler who helped to write HyperTalk. To make HyperCard practical in terms of size and speed, Atkinson developed proprietary algorithms for fast searching and for the compression and decompression of graphics images. HyperCard can run on Macintosh computers with 1 mega byte Ram and



two 800K-byte floppy disk drives. A hard disk and extra memory are most essential for smooth operation.

## 2.2 Features:

The user interface of HyperCard is consistent with the Macintosh style of pull-down menus. To let inexperienced users feel comfortable and to protect the application from unwarranted alteration, HyperCard allows 5 different user levels:

<u>Level</u>	<u>Powers And Abilities</u>
Browsing	just browsing through cards; no alteration of contents
Typing	adds text entry and editing on cards.
Painting	adds access to the Painting Tools.
Authoring	adds access to Button and Field tools.
Scripting	adds access to HyperTalk scripts (programming).

### 2.2.1 Basic Units:

The basic units of HyperCard are stacks, background, cards, fields and buttons. Each of these has properties which can be manipulated either directly by users or by HyperTalk. A HyperCard application as a collection of stacks. A stack is a ordered collection of cards. A card can contain a combination of text, graphics, fields, and buttons. A field is just like a rectangular window in which text can be displayed and

entered. The style of a field can be controlled by user and there are facilities for searching text. Fields give HyperCard some data base features. The major function of a button is to carry out actions according to the content of the HyperTalk script assigned to that button. A script is a piece of HyperTalk code. Buttons are activated by clicking the mouse. Each unit has many properties attached to it, including font, name, location and script.

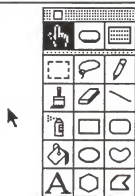
The resolution of HyperCard is limited to 342 high by 512 wide pixels. Only one card can be open (displayed) at any time. A background can contain graphics, text, buttons and fields. It can be shared by many cards.

There are standard Macintosh menu bars for accessing all the HyperCard tools, including a complete script editor for the scripts in the basic units.

### 2.2.2 Menus:

The menu bars are context sensitive. When a certain subject is chosen, the functions in the menu bars change accordingly that relate to this particular subject . For example, when a pencil tool is chosen from the menu:

🍏 File Edit Go Tools Objects



the menu will be changed into

🍏 File Edit Go Tools Paint Options Patterns



Similarly, when the Button Tool is chosen, the active object in the menu becomes buttons.

The functions in the menu bar can be manipulated through programming. A script can mimic the action of a user. For example:

```

on mouseUp
  doMenu "New Card"
  doMenu "Print Stack"
  click at 100, 200    ==> click at coordinate
  choose oval tool
  set linesize to 2
  set centered to false
  drag from 200, 50 to 140, 300
end mouseUp

```

### 2.3 HyperTalk:

Each of the five basic units can have its own HyperTalk script. HyperTalk can modify the existing properties of the basic objects, but one can not create new properties for objects. The syntax of HyperTalk is similar to that of a traditional programming language like Pascal, with assign statement, control, loop and operators. There are many powerful functions for arithmetic, sound, file and screen manipulation. Messages or events can be passed between objects, and the handlers will then carry out action accordingly. For example:

```

on openCard
  send "doMenu Delete Card" to stack "Home"
  send "mouseUp" to bkgnd button "Next" of stack "Home"
  PrintWelcomeMessages
end openCard

```

If an object does not have a message handler for a message, the object will pass the message to another object according to the hierarchy of HyperCard objects. The hierarchy diagram is redrawn from [4].

## Sources of messages

## Receivers of messages

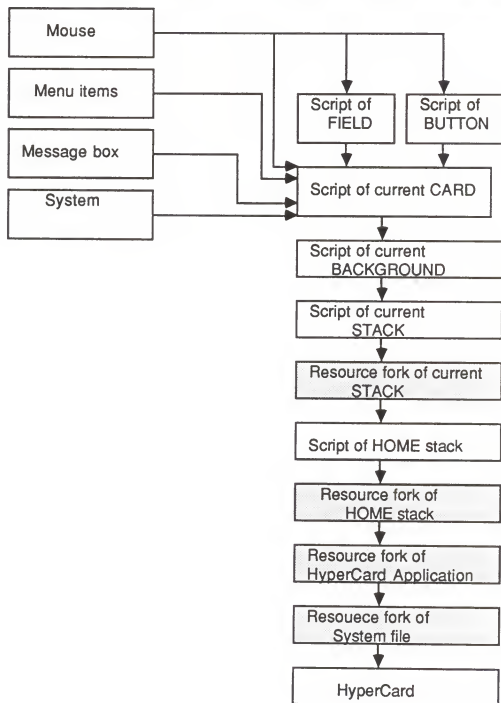


Fig. 1 Inheritance Hierarchy of HyperCard

One of the major features of HyperTalk is the ability to generate or modify a script. For example:

```
on openCard
    put "on mouseUp" & return into newScript
    put "go next" & return after newScript
    put "end mouseUp" & return after newScript
    set the script of button 1 to newScript
end openCard
```

In HyperCard programming, there is no main program as in traditional programming languages like Pascal. The occurring event dictates the flow of the program. The following script is contained in the Stack script:

```
On openStack
    DoSomething
end openStack
```

When the stack is opened, HyperCard looks into the Stack's script to find out what to do or what messages to send. HyperCard can start from any stack and activate other objects which in turn can activate other objects.

To communicate with other programming environments like Pascal, C or Assembly, HyperCard supports two types of external routines, XCMDs work like HyperTalk message handlers that can be called without or with arguments. They

can not return values. The XFCNs, work like user defined functions which can pass arguments and return values [3].

## DOCUMENTATION

The design, structure, and implementation of HyperSign are described in this chapter.

### 3.1 Environment:

The basic requirement needed to run HyperSign is a Macintosh computer with 1 mega bytes of RAM, 2 mega bytes of hard disk, HyperCard 1.0, System 5.2 and Finder 5.2. A Macintosh compatible flat bed scanner with a resolution up to 300 dpi was used to digitize the sign images from [9]. The format used was PICT. Each file contains 6-8 sign images.


The format of the file containing the original sign image, needed to add new sign, is PICT.

A desk accessory such as Artisto, is used to cut images scanned image files and put them into the clipboard for later pasting to the new sign card. The Multifinder must be turned off in order to run Artisto.

### 3.2 How to use HyperSign:

#### 3.2.1 Searching:



To open HyperSign, double click the Menu file  under Finder. The first card of Menu stack will show up on screen. To



search the sign image with a key word, click the Sign Dic



button **SIGN DIC** in the Menu Home card of Menu stack. This will bring up the Searching card. Then click the appropriate Letter



button **B**, corresponding to the first letter of a key word. The Key Word card will then be brought up. Clicking the Key Word button will cause the desired sign image card to be brought up.

### 3.3.2 Adding:

When adding a new key word and the corresponding sign



image one need to click the Add Word button **ADD WORD** in the Menu card. This will bring up the Adding card. Then click the



appropriate Letter button **B**. After clicking the Letter button, a window will prompt for the key word. A new Key Word button will be put into the Key Word card. Another window will prompt the user to click the Import Paint

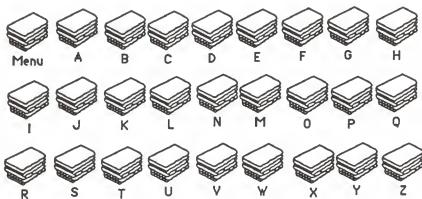


button **Import Paint**. A blank new card with the name of the new key word will be created in the Sign stack. The desk accessory, Aristo, can then be used to cut and paste the corresponding

sign image into the card. The time required to add a new key word and the corresponding sign image is about 40-60 seconds.

### 3.3 Overall Organization:

HyperSign consists of 27 stacks.



The reason to use 26 separate Letter stacks is for distribution. The combined size of all 27 stacks is much more than a floppy disk can hold. So each Letter stack has all the sign images corresponding to the key words which start with the same letter.

### 3.4 Menu stack:

The Menu stack is a management center containing the facilities of searching and adding new key words. There are 29 cards in Menu stack.

The purpose of card 1, the Menu Home card, is to let user go to the Adding or Searching card.

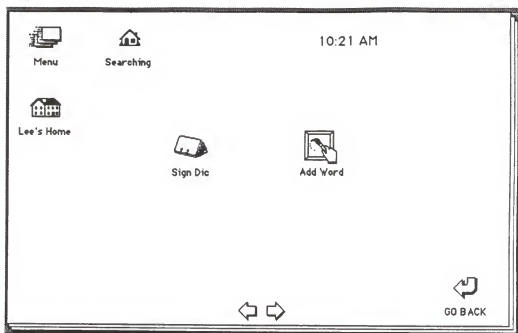


Fig. 2

The purpose of card 2, the Searching card, is to present all the 26 Letter buttons. When a Letter button is clicked, the card containing all the key words starting with this letter will be brought up.

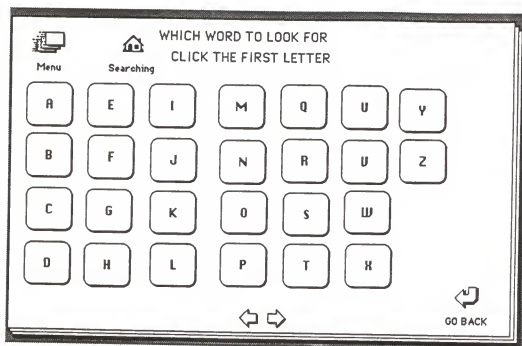


Fig. 3

The lay out of card 3, the Adding card, is the same as Searching card, but the alphabet buttons have different scripts that lead users into Key Word cards in adding mode.

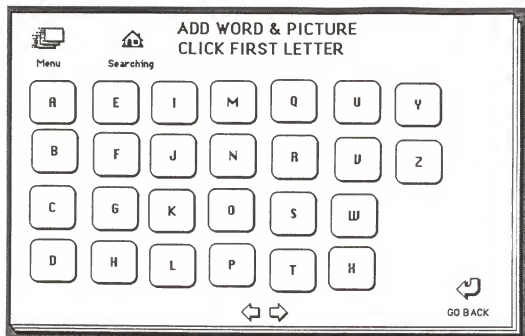


Fig. 4

Cards 4 through 29 are Key Word cards which contain all the key words with the same first letter. These 26 cards are in alphabetical order.

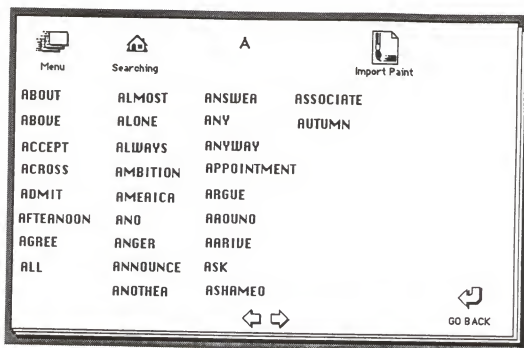


Fig. 5

### 3.5 Sign stack:

Each stack is named by a letter of alphabet. Each card is named with the key word of the sign image. All the cards in Sign stacks share the same format except for the sign image.

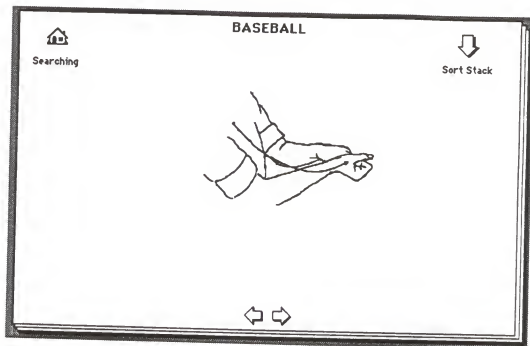


Fig. 6

### 3.6 Button functions:

All of the cards shared 5 common buttons:



**Menu** Menu Home button: goes to the first card of Menu Stack.



**Searching** Search Menu button: goes to the Searching card in the Menu Stack.



Direction buttons: go to previous or next card in the same stack.



**GO BACK** Go Back button: goes back to the previous card, the last closed card which could be in different stack.



**SIGN DIC** Sign Dictionary button: goes to the Searching card.



**ADD WORD** Add Word button: goes to the Adding card.





Letter button: goes to the Key Word card containing all the key words starting with this letter.

**BASEBALL** Key Word button: goes to the card containing the corresponding sign image.



**Import Paint** Import Paint button: If HyperSign is in adding mode, this will prompt the user to import a sign image. Otherwise, it will show a warning message.



**Sort Stack** Sort Stack button: The purpose of this button, only in Sign stack cards, is to sort the stack by the name of the cards.

### 3.7 Programming:

Because each object can have its own scripts and usually in an application there are many objects, it is ideal to be able to do most of the programming in one place. Also, according to the scoping rules of HyperCard that a shared event handler must be in a higher hierarchy. So In HyperSign, Menu stack served as a management center. Most of the event handlers is in stack script, Fig. 1, so that these event handlers can be shared by other objects with lower hierarchy.

### 3.7.1 Button scripts:

All the button scripts are shown bellow:



Home

```
on mouseUp
  go to stack "Menu"
end mouseUp
```



Sign Menu

```
on mouseUp
  go to card id 3314
end mouseUp
```



```
on mouseUp
  go to prev card
end mouseUp
```



```
on mouseUp
  go to next card
end mouseUp
```



GO BACK

```
on mouseUp
  go back
end mouseUp
```



SIGN DIC

```
on mouseUp
  go to card id 3314
end mouseUp
```



ADD WORD

```
on mouseUp
  go to card id 5854
end mouseUp
```



```
on mouseUp
  gotargetcard -- in searching card, go to the Key Word card
                -- starting with "B"
end mouseUp
```



```
on mouseUp
  addbutton -- in adding card, go to the Key Word card
            -- starting with "B" and begin adding procedure
end mouseUp
```

**BASEBALL**

```
on mouseUp
  gotarget -- go to the target card
end mouseUp
```



```

Import Paint
on mouseUp
    importpaint      -- start import paint procedure
end mouseUp

```



```

Sort Stack
on mouseUp
    sort by field "name"
end mouseUp

```

### 3.7.2 Menu scripts:

Following is the script of the Menu stack, the event handler for all events used in HyperSign. Note that comment statements start with "--"

```

-----
-- set the adding mode to false      --
-- preventing from accidentally activate --
-- the import button                  --
-----
on OpenStack
    global adding
    --> declaring a global variable
    put false into adding
    --> initial mode to be false
end OpenStack

-----
-- go to the card by the name of the  --
-- button being clicked and the stack  --
-- by the name of the present card     --
-----

```

```

on gotarget
  get the short name of this card
  --> only the card name
  put it into targetstack
  get the short name of target
  --> only the target <-> button name
  put it into targetcard
  go card targetcard of stack targetstack
  -->go to the sign card being searched
end gotarget

-----
-- go to the alphabet card, in same stack --
-- go to the card by the name of the button--
-- being clicked --
-----

on gotargetcard
  get the short name of target
  --> get the name of the button clicked
  put it into targetcard
  go card targetcard
  --> go to that card
end gotargetcard

-----
-- add the key word button, set the name --
-- of the button according to user's input --
-----

on addbutton
  --> ** preparations **
  global adding
  put true into adding
  --> adding mood is in effect
  global buttonname
  --> key word for user input,
  --> will be used globally, for important
  --> for the new key word button
  --> for the name of the new sign card
  --> for the field name of the new sign card
  --> script for new button

```

```

put "on mouseUp" & return & "gotarget" & return & ¬
"end mouseUp" into ¬
buttonscript
--> press option and "L" for ¬,
--> continuation of a new line
--> ** end of preparation **
--> ** action starts here **
gotargetcard
-- go to the alphabet card, in same stack
-- go to the card by the name of the button
-- being clicked
--> things to do with new button
get the number of buttons
--> get the number of buttons in present card
put it into bnumber
choose button tool
doMenu "New Button"
--> add new new button
set style of button (bnumber +1) to transparent
--> no outline
set showName of button (bnumber +1) to true
--> show button name => key word
ask "what is the name of the button"
--> ask user for the key word
put it into buttonname
--> buttonname to be used in many places
set the name of button ( bnumber + 1 ) to buttonname
set the script of button buttonname to buttonscript
--> self-generating code, script for new button
answer "Drag button to desired position, -> add new card"
choose button tool
--> do it for user
get the location of button buttonname
click at it
--> highlight the new button
put false into adding
--> return to none-adding mode
end addbutton

```

```

-----
-- for importing new sign image
-----
on importpaint
  global adding
  -->if in adding mode
  if adding is true then
    global buttonname
    get the short name of this card
    put it into targetstack
    go stack targetstack
    doMenu "New Card"
    set the name of this card to buttonname
    put buttonname into field "name"
    --> sen the name of this card and show it in
    --> a field of the card according to
    --> the key word which is user input in "addbutton"
    answer "Now import paint, ->Artisto "
    --> remind user to user DA, Artisto,
    --> to import paint
  else answer "Need to start from Adding Menu"
  --> not adding mood
end importpaint

```

## DISCUSSION

In this chapter the issues of designing and implementation of HyperSign is discussed. The experience of using HyperCard is also discussed.

### 4.1 HyperSign:

The approach of designing HyperSign was to mimic the way a dictionary is used. Using HyperSign should be natural for anyone who has used a dictionary. I tried to minimize the effort required to learn and use HyperSign. The difference between HyperSign and a traditional sign language dictionary is that HyperSign is more flexible, in that the user easily can add new words and signs.

The initial implementation of HyperSign is over 700 signs and key words. The average size of the 26 Letter Stacks is about 50 K bytes, and Menu Stack is less than 100 K bytes. The total size of HyperSign is about 1,500 K bytes before any data compression. The size of HyperSign is a very reasonable for distribution. A medium level sign language dictionary is about 1,200 different words and a advanced-level one is about 2000 different words. Only five mega bytes will be needed to implement a advanced level dictionary in HyperSign. Thus, making a marketable HyperSign is very feasible.



One of the concerns about HyperSign performance is the speed of searching a sign. It should be much faster than using a traditional sign language dictionary to satisfy our objective. I did a rough measurement, using a Macintosh II with a 40 mega byte hard disk which had an average access time of 29 ms. To start up a Macintosh II takes about 12 seconds. The speed of searching depends on 2 major factors: human and machine. Opening the HyperSign stack took about 4 seconds. I tested myself how much time, on the average, it would take me to locate a sign. The result was about 5-8 seconds, which I considered pretty fast. Adding a new key word and the corresponding image, on average, took me about 40-60 seconds which I think it is very reasonable time.

There are a lot of possibilities of further development of HyperSign. First, one can add multiple links to a Sign card to link with words with close relations. For example, one can link the key words with similar meanings or with similar signs. Animation is another possibility. Flashing few cards in a roll can allow a user to better understand what a signing sequence should be. However the size and complexity of a animated HyperSign will be greatly increased.

#### 4.2 Experience with HyperCard:

Learning to program a Macintosh computer is not a very easy job. The learning curve of programming a Macintosh user interface is steep. Before I started this project, I only had very limited experience with Macintosh computers. The experience of learning HyperCard and building HyperSign was exciting to me. The friendly interface of HyperCard helped to minimize the frustration, common in using new environments. The HyperCard interface is consistent with the Macintosh system. It only took me a few hours to understand almost all the features about HyperCard. During the learning period, I worked with HyperCard and consulted Danny Goodman's book [2] only when I encountered something I could not understand. It is a very nice reference book about HyperCard. The official Apple HyperCard User's Guide [1] that came with the HyperCard is not adequate; it only mentioned HyperTalk once in the entire manual.

Writing HyperTalk script is very natural to me. HyperTalk is very comprehensive. Whenever I needed some functions, those functions were there and I did not need to write complex procedures or XCOMDs just for some simple operation.

At the beginning, I manually imported the sign images but soon I found out that all the mechanical steps could be programmed. That saved me a lot of time when I was

importing hundreds of pictures. The built-in routines which can manipulate the functions in the menu bar by using HyperTalk scripts were most useful to me.

Before I automated all the steps in creating New Key word buttons and the corresponding Sign cards, I had to manually write the same short script for each new Key Word button. I soon discovered a great feature of HyperTalk, its dynamic programming environment. Since a script is a property of an object, the script can be assigned, accessed and modified. A newly assigned or modified script immediately carries out its function. In this way, I could assign a script to each newly created button through programming. This dynamic programming environment is a very nice feature; it is a key to making HyperSign user friendly.

During the period of 5 months using HyperCard, I encountered few bugs. At first I tried to run on a Macintosh with less than 1 mega byte of RAM and it gave me a message about not enough RAM. Printing gave me some difficulty, but I solved the problem by changing the printer driver to newer version. Some times Artisto, the desk accessory, clashed with HyperCard; and that could have been the result of insufficient memory.

#### 4.3 HyperCard Limitations:

Unlike most Macintosh applications HyperCard has modes, which are certain states in which that only limited functions can be performed. For example, one has to be in button mode to modify buttons, but in button mode other objects can not be modified. Switching back and forth is a hassle when creating or modifying objects.

HyperCard automatically saves the whole stack to disk erasing the original file after each step, like moving a button or finishing typing script for a object. Only one step of undo is provided by HyperCard which makes going back to the original file impossible unless there is a back up copy.

Modifying the icons of a button is not an easy task as a resource editor is needed. The resolution of the HyperCard is only 72 dpi, fixed at 342 by 512 pixels. That is low compared to 300 dpi of most Macintosh publication applications. The size of the card is fixed, only about 3 by 5 inches, and can not be re-sized as is possible for windowing in most Macintosh applications. Only one card can be opened at any time that works against the window metaphor. The five built in objects are the only object available. There is no facility to create new object classes and that also limits the power of HyperCard.

I think, that in order to make scripts look somewhat like plain English, the HyperTalk syntax was made rather complex. The scripts may look like English at first sight, but scripts looked unnecessarily complicate to me when I actually started programming. The reserved word "to" is sometime optional and sometime required. I think that is unnecessary. There are 4 types of "if" and 6 types of "repeat". For example, the format of "if" statement:

- (1) if <true/false> then <command>
- (2) if <true/false> then <statement>  
    else <statement>
- (3) if <true/false> then  
    <statement 1>  
    <statement 2>

.....  
end if

- (4) if <true/false> then  
    <statement 1>  
    <statement 2>
- .....  
else  
    <statement 1>  
    <statement 2>

.....  
end if

The example above is more complicated than Pascal or C. I think the syntax of HyperCard is a little bit wordy and noisy. The limitation of one statement per line unless there is a "—" at the end of a line, makes the syntax complicated.

The interpretive programming environment makes prototyping easy and the debugging straight forward. The debugging environment is primitive in that it only gives an error message when an error occurs in the HyperTalk script.

#### 4.3.1 Note:

Silicon Beach Co. just announced SuperCard, a second generation version of HyperCard. SuperCard support colors, an object-oriented drawing environment, multiple windows and it can import drawing formats like PICT, PICT2, TIFF and Encapsulated Postscript Format (EPSF). The shapes, icons and cursor for pushing buttons can all be customized. It has a compiler to produce a stand-alone applications.

#### 4.4 Conclusion:

With HyperCard, I was able to build a pratical application, HyperSign, in a short period of time. Overall, I like the concept of HyperCard and its nice implementation. There are similar programs, but HyperCard is the first one avaiable to many people in a personal computer environment.

## REFERENCES

- (1) HyperCard User's Guide, Apple Computer Inc. Cupertino, CA., 1988.
- (2) Danny Goodman, "The Complete HyperCard Handbook," Bantam Books, New York, 1987
- (3) Keith Weiskamp and Namir Shamma, "Mastering HyperTalk," John Wiley & Sons, Inc., New York, 1988.
- (4) Gregg Williams, "HyperCard, First Impression," Byte, vol. 12, Number 14, pp. 109-117, Dec. 1987.
- (5) Salvatore Parascandolo, "A wild Card," Macuser, pp. 205-211, Feb. 1989.
- (6) Jeff Conklin, "Hypertext: An Introduction and Survey," IEEE. COMPUTER, pp. 17-41, SEP. 1987.
- (7) F.G. Halasz, T.P. Moran, and T.H. Trig, "NoteCards in a Nutshell" Proc. of the ACM Conf. on Human Factors in Computing System, Toronto, Canada, April 1987.
- (8) Communicative Skill Program, Terrence J. O'Rourke,

Director, "A Basic Course in Manual Communication" The  
National Association of the Deaf, Silver Spring, Maryland  
20910, 1979.



HYPERSIGN: AN INTERACTIVE  
SIGN LANGUAGE DICTIONARY

by

HSING CHUNG LEE

M.S. Kansas State University, 1986

AN ABSTRACT OF A REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY  
Manhattan, Kansas

1989

## ABSTRACT

This report presents the implementation of an application called HyperSign. HyperCard is an interactive sign language dictionary in which one can find a specific sign according to the key word.

One purpose in building HyperSign was to have an experience with HyperCard. HyperCard is a user information management system that is different from traditional programming environments or database management system. It is a combination of graphic tools, graphical and text database management system, and programming environment. HyperCard can also connect to applications written in other programming languages.

HyperSign was designed to be easy to use. The user can navigate almost entirely with a mouse. When searching for a particular sign, the user clicks a Letter button, chooses the Key Word button, and then the sign image will be brought up.

Initial implementation of HyperSign consists of 700 words. New key words and corresponding signs can also be easily added without any knowledge of HyperTalk, the programming language of HyperCard.