Talking to Smalltalk

Jeffrey Lo

University of California, Berkeley
College of Engineering
Department of Electrical Engineering
and Computer Science
Computer Science Division

ABSTRACT

This paper describes my research at UC Berkeley into an interfacing a speaker dependent voice recognition system to the Berkeley Smalltalk system to provide a better user interface for the Smalltalk system to increase programmer productivity.

May 23, 1985

This work was sponsored by Defense Advance Research Projects Agency (DoD) ARPA Order No. 3803 Monitored by Naval Electronic System Command under Contract No. N00034-K-0251.
Publication of this report made possible by grant number 25076.

Talking to Smalltalk

Jeffrey Lo

University of California, Berkeley
College of Engineering
Department of Electrical Engineering
and Computer Science
Computer Science Division

1. Overview

The goal of this project was to tie together two other projects, one being the porting of Xerox's Smalltalk-80[1] system to the SUN Microsystems workstation by David Ungar, one of Professor Patterson's graduate students, the other being the Professor Broderson and Robert Kavaler's Mara speech recognition effort. The result of this was a new user interface to the Smalltalk-80 system using speech. Once this was done, we tried to discover how speech could best be put to use to increase programmer productivity in the Smalltalk environment.

2. Using Smalltalk-80 with Speech

It is simple to use Berkeley Smalltalk with speech input. Each person that is to use it must have his own account and home directory. The templates for that user's voice are kept in a subdirectory in his or her account. The Mara daemon can then be started. This will download software to the recognizer and open the VU meter window. There must also be a copy of Smalltalk-80.sources and a Smalltalk image in the current directory. Bs is then started up. Once the Smalltalk system is running the user must execute the method 'train' to load the existing templates into the Mara recognizer or prompt the user to say the words if templates do not yet exist. At this point the user can make full use of the speech capabilities integrated into Berkeley Smalltalk.

The user can now scroll any of the windows by moving the mouse pointer into a window, selecting it and saying a scroll command into the headset microphone. The current commands for scrolling are 'scrollup' to scroll the text up, scrolling down is accomplished by saying 'scrolldown'. It is also possible to jump to the top or bottom of the text in the window by saying 'top' or 'bottom'. If the mouse is pointing to an editor window, by selecting text with the mouse and saying an editor command, the desired editor command will be performed. . The current commands are 'accept', 'again', 'cancel', 'copy', 'cut', 'paste', and 'undo'.

3. Hardware Description

3.1. SUN Microsystems Workstation

Smalltalk requires a few pieces of special hardware other than those specifically required for the speech recognizer. These are a bitmapped graphics display for it to put its windows on, and a mouse to use for pointing. To satisfy these requirements, Berkeley Smalltalk runs on a SUN Microsystems workstation. The SUN workstation employs a Motorola 68010 for its processor and uses the multibus that the recognizer plugs into. The standard operating system is 4.2 BSD UNIX[2]. All of this fits into a

1 Smalltalk-80 is a trademark of Xerox Corporation.

2 UNIX is a trademark of Bell Laboratories.

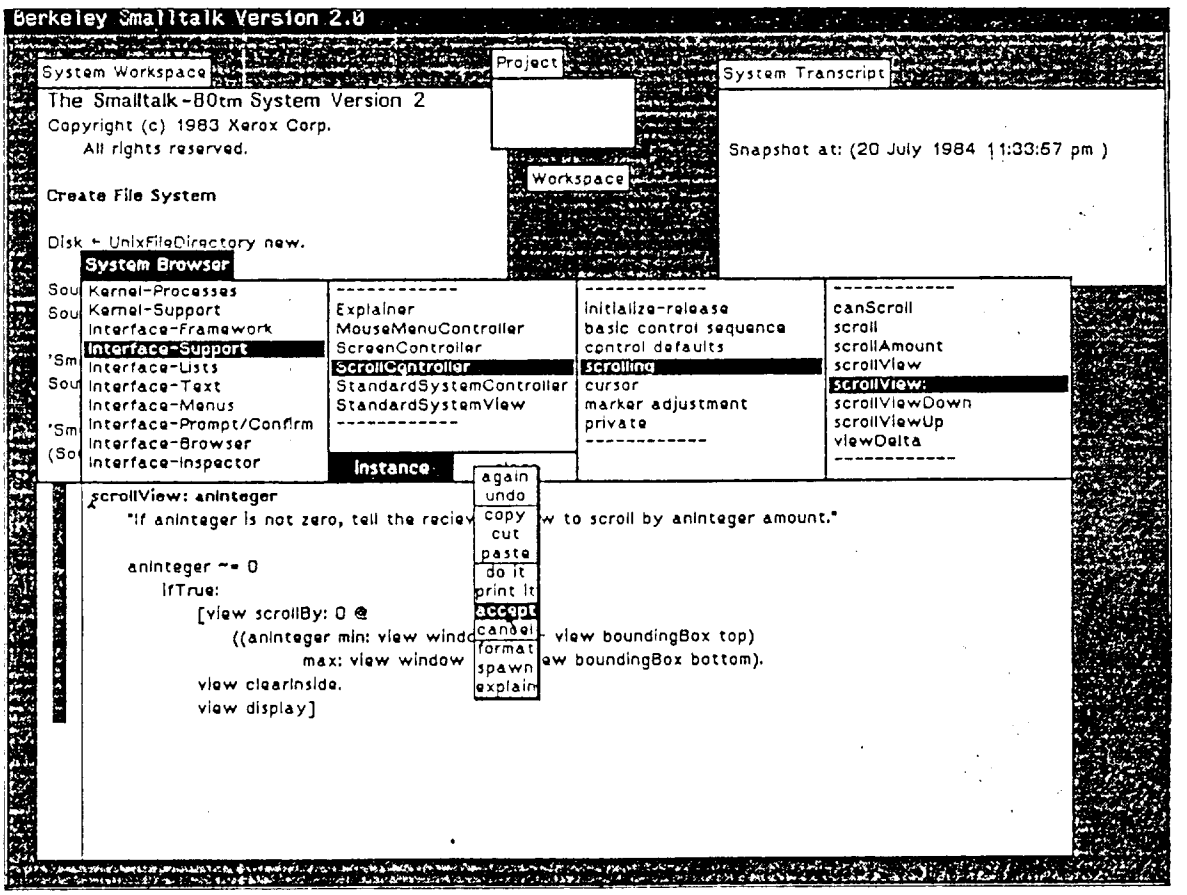
package that fits on a desktop.

3.2. Mara Speech Recognizer

The Mara system has three main hardware components, the recognizer board itself, a headset microphone for speech input, and an preamplifier for the microphone. The recognizer contains two special purpose integrated circuits developed at UC Berkeley. The purposes of these chips are to do spectral analysis and compute the Dynamic Time Warp Algorithm. There is also an Intel 80186 processor for the higher levels of the recognition algorithm and to manage templates. In addition to these basic components there is memory for the 80186, the time warp algorithm, and storing templates. There are also the ports to connect the board to the multibus and two serial channels for terminal communications.

4. Smalltalk Virtual Machine

Most of the Smalltalk-80 system, the part actually written in Smalltalk, is portable from machine to machine. The only part that is different is the so called "virtual machine". This is the lowest level part of Smalltalk, and handles all Smalltalk primitives. When the Smalltalk code is run, methods are executed that call other methods, and so on, but this all has to bottom out somewhere, and where it bottoms out are the Smalltalk primitives. Primitives are used for all machine dependent functions such as file access and running the graphics, for low level operations that cannot be reduced any further by Smalltalk such as arithmetic operations, and for speeding up operation by making compiled code for commonly used functions. Once there is a virtual machine for whatever hardware is available, Smalltalk can be run on that



Berkeley Smalltalk running on the SUN workstation

machine.

4.1. Berkeley Smalltalk

Smalltalk-80 was originally designed over a period of ten years at Xerox's Palo Alto Research Center to run on the Dorado computer. To make Smalltalk-80 run on a SUN workstation, a new virtual machine designed for the SUN had to be written. This was the project of David Ungar, a graduate student also here at Berkeley. He wrote a virtual machine in C for the SUN workstation. The result of his efforts is Berkeley Smalltalk, also known as 'bs'.

4.2. My Changes

For Smalltalk-80 to be able to use the Mara recognizer, it's virtual machine needed some new primitives to access the recognizer. Because of this, I had to modify the virtual machine. Two new primitives had to be added, one to train the recognizer, and one to get the output of it.

The training primitive is passed a string that is the word to be trained. It then calls the Mara daemon with the word to be trained. If a template does not already exist for the word, a window is popped up asking the user to say the word. The template is then stored for the recognizer. The daemon then returns a value, the index number for that particular word. The training primitive then passes this back to the calling routine.

The recognizer output routine is called with no arguments. It calls the daemon asking for the currently recognized word. If nothing has been said or nothing was correctly recognized, a negative value will be returned. Otherwise the index of the recognized word is returned.

5. Smalltalk System

The portable Smalltalk-80 system was developed at the Xerox Palo Alto Research Center. It is an entire programming environment that is based on the object oriented Smalltalk language. The environment is centered around the ideas of windows and using a mouse for input. These ideas have recently used to commercial success in the Apple Macintosh system.

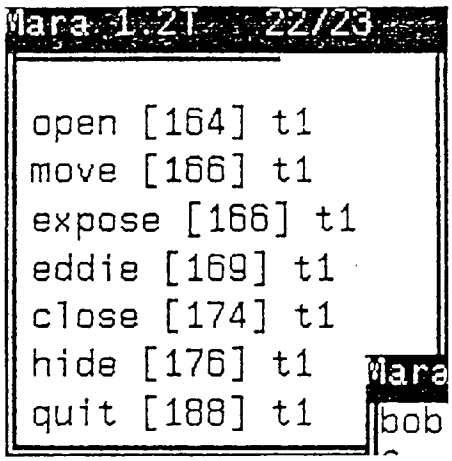
5.1. My Changes

In addition to the changes that I made to the Smalltalk virtual machine, I had to make some changes in the Smalltalk system code. There were a couple of new routines that were used in calling the new speech primitives. One of these called the training primitive and one called the output primitive. There was also a routine that when called would train each word and store the values returned from the recognizer in global variables for use later when trying to use speech commands. The *speechInput* method will return the value trained for the word that was said, or a negative value if the word was not recognized or nothing was said.

In each place that speech input was to be used, the speechInput was called to get the number for the word that was just said. This value is then checked against the stored values for any of the possible commands for that possible routine, i.e. for the scrolling routine, the values for scrolling the screen up and down and jumping to the top or bottom are compared. If one of these matches, that command is performed just as if it had been done with the mouse.

6. Mara Software

The software that operates the Mara recognizer is integrated with the suntools environment of the SUN workstation. The central part of this software is a daemon that talks directly to the Mara recognizer board and communicates with all processes that want to use the recognizer. The Mara system uses some windows of its own in the suntools environment. One of these contains a VU meter that shows the strength of the spoken word and shows the background noise. Another part of this window shows what words could have been spoken and gives a score on how well each template matched. A new window is popped up when a user is asked to train a word.



The Mara window with VU meter and scores

7. Conclusion

Overall, the combination of speech and Smalltalk worked well together. I found myself using it as I was developing it so that I would not always have to move my hands from the keyboard to the mouse.

As the system exists now, speech can be used to scroll any window on the screen up or down or jump to the top or bottom, and the editor menu commands can be used with speech. This could easily be expanded so that all system menus or a menu in a user program could use speech. Menus seem to make an ideal candidate for something that could be converted to use speech since these commands are short and can therefore be spoken more quickly than they could be typed or pointed to with a mouse. However, some things do not work as well. Anything graphical does not seem to work well since it is much easier and faster to use a mouse to point than to keep saying to the computer 'left, left, left, down, down', and so on.

Speaker dependence is another problem. With current technology, the only way speech can be recognized in real time with desktop machines is to use speaker dependent systems that require that the system be trained for each word by each person who will use it. This was not too bad as it only took about a minute to train 11 words, and is done only once per user. Eventually it is hoped that speaker independence can be achieved where any person can say anything in the computer's dictionary and have it know what he said. Once this problem is solved, speech will probably become a more common user interface since it will be easy for everyone to use.

8. Code for Speech

8.1. Virtual Machine Code

```
static char rcsId[] = "$Header: speechPrims.c,v 1.1 85/04/24 12:08:58 jlo Exp $";

/*
 *      Copyright (C) 1983 by the Regents of University of California
 *      All Rights Reserved.
 */

#include        <stdio.h>
#include        <math.h>
#include        <sgtty.h>
#include        <signal.h>
#include        <ctype.h>

#include        <sys/types.h>
#include        <sys/vtimes.h>
#include        <sys/timeb.h>

#include        "utils.h"
#include        "mem.h"
#include        "io.h"
#include        "prims.h"

/*
 * #includes for speech input.
 */

#include <stdio.h>
#include <sys/ioctl.h>
#include "../mara/mara.h"
#include <sys/socket.h>
#include <signal.h>
#include <fcntl.h>
```

```
#define SmallIntegerRATest\
        register oop_t      reg1, reg2;\
        \
        reg1 = *rp;\
        reg2 = rp[1]

#define SmallIntegerRAConvert\
        convertToInteger(reg1);\
        convertToInteger(reg2)

#define SmallIntegerOperationFinishWithOverflow(label) \
        asm("     jvs       label");\
        if (!isIntegerValue(reg1)) {\
                asm("label:");\
                return (NullRp);\
        }\
        *rp++ = integerObjectOf(reg1);\
        return (ac)

#define SmallIntegerOperationFinish \
        if (!isIntegerValue(reg1))\
                return (NullRp);\
        *rp = integerObjectOf(reg1);\
        return (ac)
```

convertToInteger

if

return

return

```
/*
 * Get the input from the mara descriptor.
 */
rp_t
SpeechPrm(rp, ac)                                        SpeechPrm
register oop_t *rp;
rp_t ac;
{
        SmallIntegerRATest;
        SmallIntegerRAConvert;
        /*
         * Return speech input.
         */
        reg1 = speechInput();
        SmallIntegerOperationFinish;
}

char *names[MAXUNAMES];
static int initialized = 0;

speechInput()                                            speechInput
{
        HEARING *hr;
        int i;
        int retval;
        int numbytes;

        if (initialized == 0)
                if (initializeSpeech() < 0)
                        return(-1);

        /*
         * Find out how many bytes are ready to be read from
         * the connection to the mara daemon.
         */
        numbytes = speechReady();
        /*
         * If there is not any data ready to be read, then just return -1.
         * By doing this we don't lose the data that we would if the
         * descriptor is non-blocking.
         */
        if (numbytes <= 0)
                return(-1);

        hr = GetHearing(NULL);
        if (hr == NULL)
                return(-1);
        /* If new data, then store in retval and return. */
        retval = hr->hr_data[0].hr_uname;
        FreeHearing(hr);
        return(retval);
}
```

```
/*
 * Return number of bytes available to be read from speech input.
 * On error return -1.
 */
speechReady()                                                    speechReady
{
        int x;

        if (initialized == 0)
                if (initializeSpeech() < 0)
                        return(-1);

        if (ioctl(mara_fd, FIONREAD, &x) < 0)
                return(-1);
        return(x);
}

initializeSpeech()                                               initializeSpeech
{
        WORD *w;
        int x;
        int window;

        initialized = 1;
        ConnectRecognizer(TRUE);
        SetMaraFlags(FLG_EVALMODE, FLG_EVALMODE);
        OnRecognizer(TRUE);
        if(getenv("WINDOW_ME")) {
                sscanf(getenv("WINDOW_ME"), "/dev/win%d", &window);
                OnRecognizer(FALSE);
                if(!AssocWindow(window)) {
                        fprintf(stderr,
                             "Cannot associate with window %d.\n", window);
                        return(-1);
                }
        } else {
                OnRecognizer(TRUE);
        }
}

/*
 * Train the word passed as a string.
 */

rp_t
TrainPrm(rp, ac)                                                 TrainPrm
oop_t *rp;
rp_t ac;
{
        int x;

        if ((x = TrainWord(rp)) < 0)
                return(NullOop);
        *rp = integerObjectOf(x);
```

```
        return (ac);
}


/*
 * Train it.
 */
TrainWord(rp)
register oop_t *rp;
{
        char word[BUFSIZ];
        int x;
        char *s;
        char path[128];

        OffRecognizer();
        getString(rp[1], word, FALSE);
        fprintf(stderr, "TrainWord(%s)\n", word);
        fflush(stderr);
        UnloadaSpelling(word);
        if ((x = LoadaSpelling(word, 0)) == MERROR)
                return(-1);
        fprintf(stderr, "Loaded\n");
        fflush(stderr);
        names[x] = (char *) malloc(strlen(word)+1);
        sprintf(names[x], "%s", word);
        TpVerify();
        OnRecognizer(TRUE);

        return (x);
}
```

TrainWord

8.2. Smalltalk-80 System Code

```
speechInput
    <primitive: 255>
    ^self primitiveFailed

        " nil speechInput "

train
    " Train the words for scrolling the screens and running editor menus. "
    Scrollup ← self trainWord: 'scrollup'.
    Scrolldown ← self trainWord: 'scrolldown'.
    Topscreen ← self trainWord: 'top'.
    Bottomscreen ← self trainWord: 'bottom'.
    SpAccept ← self trainWord: 'accept'.
    SpAgain ← self trainWord: 'again'.
    SpCancel ← self trainWord: 'cancel'.
    SpCopy ← self trainWord: 'copy'.
    SpCut ← self trainWord: 'cut'.
    SpPaste ← self trainWord: 'paste'.
    SpUndo ← self trainWord: 'undo'.

        " self train "

trainWord: t1
    " Train the string passed as the argument "
    <primitive: 254>
    ^self primitiveFailed

        " self trainWord: 'test' "

speechMenu
    " Use speech input to run the menu of the editor "
    SpVal = SpAccept ifTrue: [self accept. ^self].
    SpVal = SpAgain ifTrue: [self again. ^self].
    SpVal = SpCancel ifTrue: [self cancel. ^self].
    SpVal = SpCopy ifTrue: [self copySelection. ^self].
    SpVal = SpCut ifTrue: [self cut. ^self].
    SpVal = SpPaste ifTrue: [self paste. ^self].
    SpVal = SpUndo ifTrue: [self undo. ^self]

speechScroll
    SpVal ← nil speechInput.
    SpVal = -1 ifTrue:[^self].
    self canScroll ifFalse:[^self].
    SpVal = Scrollup ifTrue: [self scrollViewDown. self moveMarker. ^self].
    SpVal = Scrolldown ifTrue: [self scrollViewUp. self moveMarker. ^self].
    SpVal = Topscreen ifTrue: [self scrollView: 2000. self moveMarker. ^self].
    SpVal = Bottomscreen ifTrue: [self scrollView: -2000. self moveMarker. ^self]

scroll
    "Check to see whether the user wishes to jump, scroll up, or scroll down."

    | savedCursor regionPercent |
    savedCursor ← sensor currentCursor.
```

```
[self scrollBarContainsCursor]
      whileTrue:
            [Processor yield.
            self speechScroll.
            regionPercent ← 100 * (sensor cursorPoint x - scrollBar left) / scrollBar width.
            regionPercent <= 40
                  ifTrue: [self scrollDown]
                  ifFalse: [regionPercent >= 60
                        ifTrue: [self scrollUp]
                        ifFalse: [self scrollAbsolute]]].
savedCursor show
```