Effective Computer Access with Plan-Based Intelligent Screen Reader

by

Lin Ma

Advisor

Richard Simpson, Ph.D.

Submitted to the Graduate Faculty of

School of Health and Rehabilitation Science in partial fulfillment

of the requirements for the degree of

Master of Science in Rehabilitation Science and Technology

University of Pittsburgh

December, 2003

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

<u>Significance</u>

According to data collected from the 1996 National Health Interview Survey on Disability, some degree of vision impairment, defined as blindness in one or both eyes or any other reported trouble seeing, affects 8.3 million (3.1%) Americans of all ages [28]. Only 46% of individuals of working age (21-64 years old) with visual impairment are employed [1], and a significant barrier to employment is effective computer access. Additionally, in a 1996 survey performed by the American Society for Training and Development (ASTD), 73 percent of training professionals said that computer skills are "essential for employment" [2]. It is predicted that 60 percent of U.S. jobs will require computer skills within the next five years [30].

Manipulating information on the World Wide Web is rapidly becoming a crucial computer skill due to the number of tasks that are being migrated to the web. For example, a significant amount of technical support (in the form of manuals and knowledge bases) is only available on-line, and on-line forums are frequently replacing in-person meetings. However, there are strong indications that current screen readers fail to provide adequate access to the WWW. A survey performed by Earl and Leventhal [3] found that many visually impaired computer users had difficulty performing tasks associated with web browsing. 55 percent of the survey respondents found it difficult or impossible to use the Windows Help feature (which is hypertext based, like web pages) and 56 percent had difficulty filling out forms on the Internet. Another study directly compared able-bodied and visual impaired computer users, and found that visually impaired users were slower at performing several web-browsing tasks, including locating a document using a search engine and retrieving information from a table [4].

A lot of effort has gone into making software applications and web pages more accessible to the visually impaired. The World Wide Web Consortium (W3C) has developed a series of guidelines for authoring accessible web pages [5]. These guidelines describe the effort that must be put into coding documents so that a person using a screen reader has access to the same information as a sighted user of the webpage. In other research, multimodal interfaces that can provide both auditory and haptic feedback were found to have improved the overall performance in mental workload and performance time of users who are blind [6] [7] [8]. Other research focused on capturing simple semantic information from complex HTML structures such as tables, frames and forms, which can facilitate navigation in web pages [9] [10] [11] [12]. These approaches are similar in that they focus on providing an accessible computing environment, rather than facilitating the specific goal (e.g. trying to look for weather broadcast for his or her city) the user is seeking to achieve.

Recently, researchers have begun to take into account users' goals and needs in order to facilitate the tasks users want to perform. Hook et al, developed an adaptive system named PUSH (Plan and User Sensitive Help) that helps non-visually impaired users to search for and filter out the information most relevant to their needs [13]. Hook found the adaptive system required substantially fewer actions and reduced cognitive load. The idea of an intelligent interface that adapts to the user's goals and needs could also be used to achieve effective computer access for visually impaired users.

The goal of this study is to develop, implement and evaluate an intelligent screen reader system (used by the visually disabled to access computer) that responds to changes in task context. In the intelligent screen reader (ISR) system developed for this thesis project, a user performs a sequence of actions, the system observes his or her actions and make inferences about

the user's goal based on those actions (referred to as plan recognition) and then creates a script that automates the task of achieving that same goal in the future. Our work thus far has focused exclusively on tasks that are performed on the World Wide Web (WWW) but the ISR system is expected to apply to all computer access operations.

JAWS and JAWS Macro Recorder

The software under development was integrated with a commercially-available screen reader called JAWS. JAWS for Windows is a stand-alone text to speech navigation screen reader application developed by Freedom Scientific Inc. JAWS functions by capturing information on the computer screen and reading it aloud using a speech synthesizer. It is a commonly used product for providing access to computer operating systems, digital documents and web pages. It is one of the two leaders in screen reader market (the other being Window-Eyes) and is commonly thought of as the application that creates employability for persons with blindness. JAWS was developed by Henter-Joyce in 1995 and it is used by 56% of persons with visual impairments [29].

JAWS was chosen to be the prototype in this study because it is the predominant application used by persons with blindness and severe visual impairment and one of the most sophisticated screen readers available on the market. It provides scripts which are miniature computer programs that combine a number of steps or keystrokes into one operation. Each script file is associated with a specific application. This feature gives a lot of flexibility to users for customizing their tasks within a specific application. A JAWS macro recorder is also provided by Freedom Scientific Inc. It records a user's keystrokes and additional events such as waiting for a new web page to be loaded, and then produces a corresponding script. The script from JAWS macro recorder serves as the observation of the user's actions.

Currently, no screen reading software allows users to associate scripts with individual web pages or sites (linked collections of pages), despite the fact that a web site like Charles Schwab's can be considered a fairly sophisticated financial management application in its own right. The ability to use page- and site-specific scripts will allow screen readers to adapt to the current task context while the user is browsing the WWW, which will increase efficiency and facilitate the performance of complex tasks that may span multiple web pages within a site.

Plan Recognition Netwworks (PRNs) and Automated Synthesis of Plan Recognition Networks (ASPRN)

Plan recognition was chosen to be the back end of the intelligent screen reader system because it supports advice generation, task completion, error detection and task recovery. The plan recognition technology used in this study is based on Plan Recognition Networks (PRNs) structures. PRNs are belief networks that support plan recognition. PRNs calculate probabilities that indicate how closely the user's actions match the plan represented by the PRN based on observations of the user's actions.

PRNs are constructed by Automated Synthesis of Plan Recognition Networks (ASPRN) developed by Intelligent Reasoning Systems Inc. in San Diego, CA. ASPRN provides a powerful representation and reasoning scheme for plan recognition. The ASPRN system is an automated method of creating probabilistic models suitable for plan recognition from plans containing common programming constructs such as sequences, conditional branching, and iteration. In terms of the JAWS screen reading application, a plan for accomplishing a specific task under specific conditions is represented by a JAWS script, and the terms "plan" and "script" will be used interchangeably. Thus, in this study ASPRN will take a plan or a JAWS script as input and outputs a specially-constructed belief network (a PRN).). In this thesis project, three PRNs

representing three scripts that are useful for performing typical web browsing tasks were constructed by the developers at the University of Pittsburgh.

Script Generation Interface (SGI)

SGI is a dialog based application written in visual C++. It allows users to modify the actions recorded by the JAWS macro recorder and generate an optimized script output of plan recognition. It is the middle component in the system that connects with the JAWS macro recorder and ASPRN. Tasks involved in integrating the SGI with the JAWS macro recorder included making modification to certain JAWS script files and obtaining information associated with users' actions. Tasks involved integrating the SGI with ASPRN included constructing PRNs for typical web browsing tasks, submitting observations to the PRNs, querying the probability of each PRN and picking out the plan with the highest probability. Tasks involved in implementing the SGI included implementing the user interface. The SGI-related development project described in this paragraph is finished by the developers at the University of Pittsburgh.

This thesis is structured as follows. Chapter 1 contains a brief overview of intelligent reasoning system architecture. Chapter 2 describes the implementation of three main components in the system. Chapter 3 presents the results of user evaluation with developed intelligent screen reader system. Chapter 4 summarized the work with conclusions and describes the direction for future research and development work.

1. Intelligent Screen Reader System Architecture

Plan recognition is the technique of inferring a user's intentions (or plans) from the user's actions or utterances [14]. As stated, plan recognition can make an interface more intelligent and interactive, and it has been widely applied in the domains of intelligent interfaces, natural language processing and intelligent tutoring systems [15]. There are a number of different plan recognition algorithms. Many algorithms use purely syntactic recognition schemes such as heuristics, finite state automata or plan graphs. But these schemes are useful only in situations with little or no uncertainty [16]. Identifying the goal of a person using a screen reader user may involve a lot of uncertainty, thus plan recognition algorithms that use probabilistic or fuzzy representations are more suitable for screen reader applications than other algorithms [17] [18] [19] [20] [21] [22]. In this paper, the development of an intelligent interface for the JAWS screen reader is described. It uses belief networks based on a probabilistic representation and reasoning framework for plan recognition of what task the JAWS user it attempting to complete. The system architecture is shown in Figure 1.1.
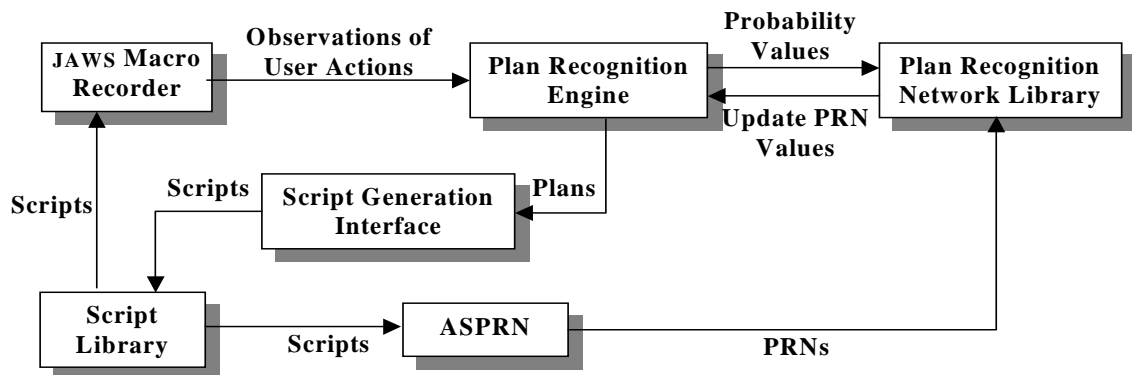

Figure 1.1 Intelligent Screen Reader System Architecture

1.1. JAWS Scripts and Macro Recorder

JAWS for Windows is a stand-alone text-to-speech screen reader application that provides application-specific speech output through script files written specifically for them. Script files are a collection of individual scripts, and each script is a miniature computer program that combines a number of steps or keystrokes into one operation [25]. Freedom Scientific, the company that makes JAWS, has developed sophisticated script files for a number of popular applications. Scripts create an advantage for JAWS users because they are able to use more applications than Microsoft Office or common database applications. A JAWS user only needs to write, modify or obtain the services of someone who can write a script for them in order to use JAWS to access custom or niche market software. This allows JAWS users to efficiently perform tasks required in their unique work place. A sample script for speaking the name of the last opened file in WordPad is shown in Figure 1.2. The sequence of actions in the script can be activated by a single keystroke or two-key combination.

Script LastFile ( )

SpeechOff ( )   ;mute speech output

{Alt+F}   ;press Alt+F keystroke to open File menu

Pause ( )   ;wait until operation is done

NextLine ( )

NextLine ( )

NextLine ( )

NextLine ( )

NextLine ( )

NextLine ( )   ;perform several Down Arrow key to the line of the

```
NextLine ( )            last opened file

SpeechOn ( )            ;turn speech on

SayLine ( )             ;speak out the content of the current line

SpeechOff ( )           ;mute speech output

{escape}

{escape}                ; two Escape keystroke to close File menu

Pause ( )               ;wait for the operation

SpeechOn ( )            ;turn speech on
```

EndScript

Figure 1.2 A Sample Script for speaking the name of the last opened file in WordPad

A macro is a set of commands that are recorded and can be "replayed" to perform a given task [26]. These tasks can be simple (e.g., inserting name and address into a word processed document) or complex (e.g., launching a program, copying data from it, activating another program, and pasting the data into it). Macros provide the similar savings in time and energy as scripts do but do not require programming knowledge on the part of the user. The JAWS macro recorder is a new feature recently added to JAWS that records a user's keystrokes and produces a corresponding script. A sample script produced by the macro recorder is shown in Figure 1.3. In this example, the user first searched for the "Yahoo! Auction" link on the Yahoo homepage, opened a new window for this link, typed "book" in an edit box on this web page, and pressed the "Enter" key to start the search for the book information.

Script S_macro()

       PerformScript Tab()

       PerformScript Tab()

       PerformScript Tab()

       PerformScript Enter()

       WaitForWindowWithName("Yahoo! Auctions - Find Great Deals and

                      Unique Collectibles - Microsoft Internet Explorer")

       PerformScript Tab()

       PerformScript Tab()

       PerformScript Enter()

       TypeKey("B")

       TypeKey("O")

       TypeKey("O")

       TypeKey("K")

       TypeKey("Enter")

       WaitForWindowWithName("book - Find Great Deals on Yahoo! Auctions

                      - Microsoft Internet Explorer")

       PerformScript Tab()

       SayString("macro complete")

EndScript

Figure 1.3 A Sample Script Recorded by JAWS Macro Recorder

1.2. Plan Recognition Network (PRN)

A Plan Recognition Network (PRN) is a belief network that supports plan recognition [22]. A belief network is a directed, acyclic graph representing the dependencies among a set of random variables [23]. Each random variable ranges over a domain of outcomes with a conditional probability distribution specifying the probabilities for all combinations of outcomes and outcomes of the variable's predecessors. Belief networks can be thought of as a means of organizing information to allow the convenient application of a form of Bayes' theorem:

$$\Pr(H \mid e) = \frac{\Pr(e \mid H)\Pr(H)}{\Pr(e)}$$

Here, H is a probabilistic variable that stores the probabilities of three different propositions: (1) the plan associated with H is being pursued, (2) the plan is not being pursued, and (3) the plan has been achieved. e is the set of observations of user's actions. The value of the probability $\Pr(H \mid e)$ represents the probability that the user is following a specific plan given the evidence presented to the belief network. The belief network's structure only requires probability values to be assigned for dependencies and eliminates the need to specify a large number of unnecessary values typically required by probabilistic representations. Therefore belief networks provide a succinct and efficient representation and inference framework for probabilistic reasoning.

PRNs calculate probabilities that indicate how closely the user's actions match the plan represented by the PRN based on observations of the user's actions. PRNs may be constructed

by hand, but in this study PRNs are automatically constructed by Automated Synthesis of Plan Recognition Networks (ASPRN).

1.3.   Automated Synthesis of Plan Recognition Networks (ASPRN)

The Automated Synthesis of Plan Recognition Networks (ASPRN) system [22], developed by Intelligent Reasoning Systems Inc., provides a powerful representation and reasoning scheme for plan recognition. It is a collection of theories and algorithms that take a plan or script representation as input and outputs a specially-constructed belief network (a PRN) that is optimized for performing plan recognition. ASPRN supports a wide range of commonly found plan constructs including action sequences, conditional branches, multilevel plans (i.e., plans with subplans), and iteration. It completely automates the construction process, which tremendously speeds the process, eliminates construction errors, and facilitates dynamic construction. In previous studies [22] and [24], researchers demonstrated ASPRN's capacity to perform plan recognition within very complex, dynamic and fast-paced environment. In our application, plans are represented as JAWS scripts. For example, ASPRN can take the JAWS script shown in Figure 1.4a and automatically construct the PRN shown in Figure 1.4.b.
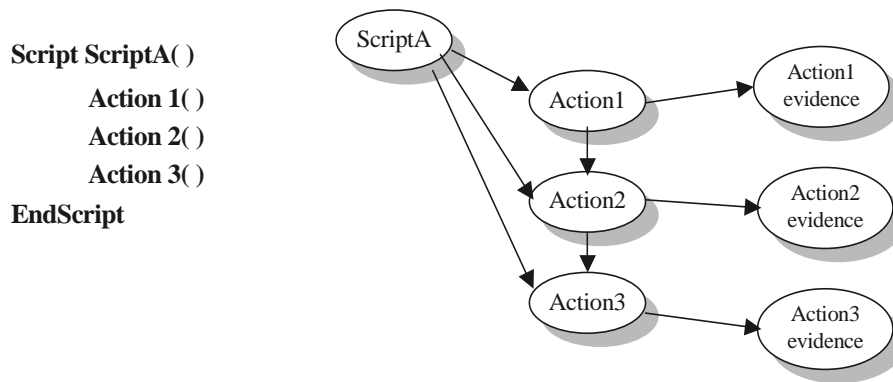
```
Script ScriptA( )
        Action 1( )
        Action 2( )
        Action 3( )
EndScript
```

Figure 1.4.a Script Representation   Figure 1.4.b PRN produces by ASPRN

1.4. Script Generation Interface (SGI)

The Script Generation Interface (SGI) application developed in this study is capable of generating a script from a list of user actions with or without intervention from the user. The Plan Recognition Engine (PRE) has been built into the SGI. It observes the user's actions, makes use of ASPRN's plan recognition networks and picks the plan (represented as a function call) with highest posterior probabilities. Once finished, the output script is labeled by the user and placed in the Script Library. The new script can be fed into ASPRN to generate a new PRN, which will expand the number of plans that the system is capable of recognizing. Besides plan recognition, the SGI also allows users to manually modify the action list to refine the script produced by the SGI.

2. Implementation of the SGI in the Intelligent Screen Reader System

The SGI was implemented using Microsoft Visual C++ as a dialog based application generated by the Microsoft Foundation Classes (MFC) AppWizard.

2.1. Integration with the JAWS Macro Recorder

As shown in Figure 1.3, the JAWS macro recorder records all of the user's keystrokes and some events such as waiting for a web page to load. In order to better understand the user's actions and make more accurate inferences, the SGI needs more information, including the position (represented by line number and column number in the JAWS virtual PC buffer) of the control on which the user performs the action, the type (such as a link, a button, or an edit box) of the control, and the text describing the control (For example, "Auction Page" for the link labeled as "Auction" on Yahoo home page).

Modifications were made to several JAWS script files to obtain the information associated with a user's actions. Modifications were made to the MacroKeyPressedEvent( ) function and the ToggleRecorderOnOff( ) script in Macros.jss, as shown in Figure 2.1 and Figure 2.2.

Script ToggleRecorderOnOff()

var

      string sFileName,

      string Section

ToggleRecorderOnOff( )   ;The original script only has this function.

let sFileName="Macros.jsi"

let Section="Recorder"

if (bIsRecorder==false) then

      Let bIsRecorder = true

      Let globalCommNum = 0

      Let globalRecNum = IniReadInteger (Section,"Index", 0, sFileName) + 1

      IniWriteInteger (Section,"Index", globalRecNum, sFileName)

      IniWriteString (Section,"Title", GetAppTitle(), sFileName)

      IniWriteString (Section,"Date", SysGetDate(), sFileName)

      IniWriteString (Section,"Time", SysGetTime(), sFileName)

else

      IniWriteString (Section,"Name", globalRecName+"_macro", sFileName)

      IniWriteInteger (Section,"ComCount", globalCommNum, sFileName)

      let bIsRecorder = false

endif

EndScript

    Figure 2.1 Modifications Made to the ToggleRecorderOnOff( ) Script

Void Function MacroKeyPressedEvent (int nKey, string strKeyName, int nIsBrailleKey, int nIsScriptKey)

```
var
    int index,
    string sFileName,
    string sSectionName

    if (bIsRecorder==true) then
        Let globalRecName = GetLine()
        Let globalCommNum = globalCommNum + 1
        Let sSectionName = "R"+IntToString(globalRecNum)+"Command"
        Let sFileName = "Macros.jsi"
        IniWriteString (sSectionName+IntToString(globalCommNum), "Line", IntToString (GetCursorRow ()), sFileName)
        IniWriteString (sSectionName+IntToString(globalCommNum), "Col", IntToString (GetCursorCol ()), sFileName)
        IniWriteString (sSectionName+IntToString(globalCommNum), "Type", GetObjectType (), sFileName)
        IniWriteString (sSectionName+IntToString(globalCommNum), "Text", GetLine (), sFileName)
        IniWriteString (sSectionName+IntToString(globalCommNum), "Key", strKeyName, sFileName)
    endif
```

EndFunction

Figure 2.2 Modifications Made to the MacroKeyPressedEvent( ) Function

When the JAWS macro recorder is activated, a JAWS settings initialization (.jsi) file called Macros.jsi is opened (or created if it does not exist). The MacroKeyPressedEvent() function is called following each keystroke while the macro recorder is active. GetCursorRow( ) and GetCursorCol( ) return the row and column numbers of the control on which the most recent keystroke is operated. GetObjectType ( ) returns the type of the control and GetLine ( ) returns the text description for the control. In addition, the number of keystrokes is updated every time when the MacroKeyPressedEvent( ) is called. When the macro recorder is toggled off, the index of the most recently recorded macro, the name of the macro and total number of keystrokes are stored at the beginning of the Macros.jsi file. Users can record multiple macros, and the macro recorder appends the information for each new macro to the end of the Macros.jsi file. The SGI finds the most recently recorded macro based on the information stored at the beginning of the Macros.jsi file.

The Macros.jss and Macros.jsi files are read by the SGI application on start-up. Generally, each line in the body of a macro recorded in Macros.jss corresponds to the information stored for a single keystroke in Macros.jsi, but there are two exceptions. For the WaitForWindow-WithName() function, no keystroke information is found in Macros.jsi since the user doesn't press any keys while waiting for a new web page to load. In this case, the SGI assigns default attributes to WaitForWindowWithName (Line = 0; Type = Virtual; Text = "Wait For Next Window").

Another exception is the MacroSetWindowText( ) function created by the macro recorder and stored in Macros.jss. The parameter in this function is the text string entered into the edit

11

field of the JAWS Find dialogue box. However, a user may perform extra keystrokes while entering the string, and these will be recorded into the Macros.jsi file. For example, the user may misspell the word "book" as "booq" and then use the backspace key to erase the letter "q" and then correct the spelling by adding the letter "k". In this case, the macro stored by the macro recorder in the Macros.jss file will contain a function call to MacroSetWindowText("book") but the file Macros.jsi will contain a record of all the keystrokes (including the "q" and the backspace key). To synchronize these two files, the SGI assumes all keystroke actions associated with the edit field correspond to a single MacroSetWindowText() function call.

2.2. Integration with Automated Synthesis of Plan Recognition (ASPRN)

The SGI is integrated with ASPRN through the ASPRN Application Program Interface (API). The header, object and library files for ASPRN (but not the ASPRN C++ source code), as well as the ASPRN API were provided by Intelligent Reasoning Systems. The API allows the SGI to submit observations to PRNs created by ASPRN, query for posterior probabilities and reset all observations.

```
Script FindKeyWord( )
        PerformScript IEFind( )
        MacroSetWindowText( )
        TypeKey("enter")
EndScript


Script FollowLink( )
        PerformScript Enter( )
```

WaitForWindowWithName( )

EndScript

Script ComboInput( )

PerformScript Enter( )

TypeString( )

TypeKey("enter")

WaitForWindowWithName( )

EndScript

Figure 2.3 Script Representations in Basic.jss

Currently, the script library consists of three scripts saved in the basic.jss file. The contents of basic.jss are shown in Figure 2.3. These three scripts were chosen because they are very typical tasks in web browsing. FindKeyWord( ) represents the task of bringing up the JAWS Find dialog box, typing a text string in the edit field and searching for the string in the web page, FollowLink( ) represents the task of clicking a hyperlink and bringing up a new web page, and ComboInput( ) represents typing some characters or numbers in a search edit box and pressing the "Enter" key to transmit the information to the web page. ASPRN builds a PRN structure similar to the one in Figure 1.4.b for each script representation. In addition, ASPRN creates a connector node that directly connects with the root nodes of each PRN. The output file created by APRRN is called beliefnet.dnet and is used by ASPRN when performing inference. Three PRNs with the connector node in Beliefnet.dnet are shown in Figure 2.4.
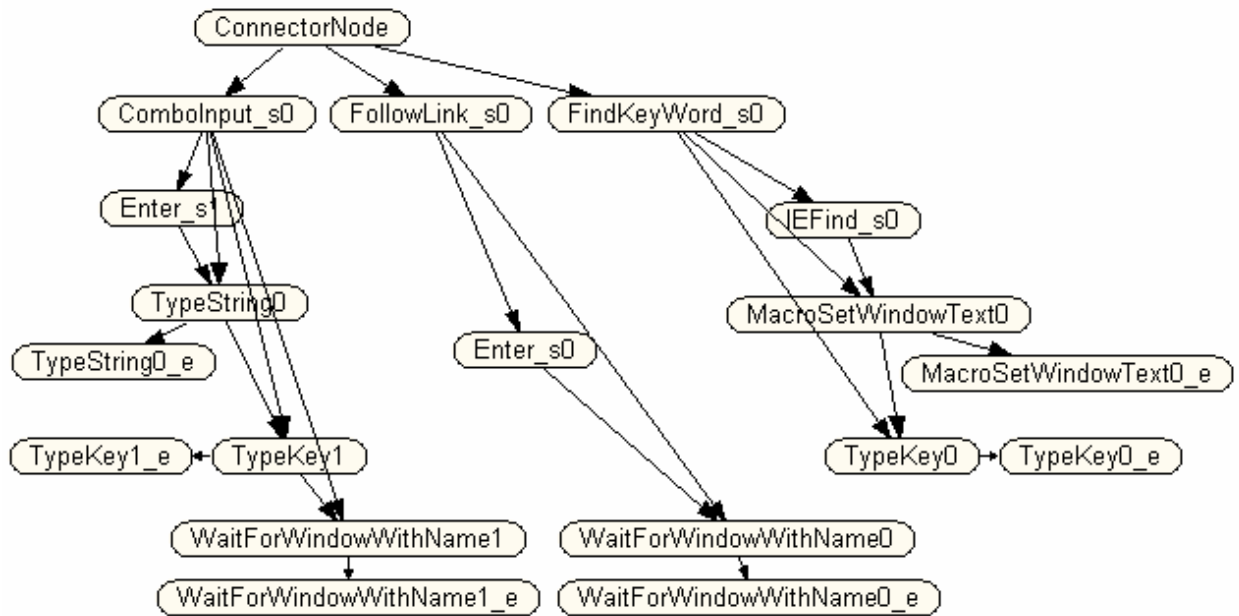
Figure 2.4 PRNs with connector node in beliefnet.dnet

Once a PRN is created and initialized by ASPRN for each script in the script library, the SGI submits the observations in the action list one by one to all the PRNs. As long as the observed actions are consistent with the actions in at least one of the PRNs, the probability of achievement in certain task will increase. Since every task has unique actions associated with it, at some point in time, the probability of one task rises above the other two. The SGI does not conclude that a certain task has been achieved until the probability of achievement for this task reaches above 0.85. After a certain task is found to have probability of achievement above 0.85, the SGI continues to add evidence to the PRNs. As long as the added actions continue to increase the probability of achievement for the selected task, they are also included. Otherwise, the added action is considered unrelated to the selected task.

The SGI replaces the action sequence with a function call such as FollowLink(34, "RESNA home page"). This function call can be viewed as a sub-task of the user's plan. The parameters in the function call are extracted from the attributes of the actions. All observations added to PRNs are then retracted and everything starts over again from the last action not included. At the end of the procedure, the SGI outputs all the sub-tasks (also called the optimized script). Users can then label the optimized script, save it, and create a new PRN based on this script for future use.

2.3. Implementation of the SGI

2.3.1. Important Classes and Functions in the Application

The main function of the SGI application is to process recorded user actions. Class SGIAction builds the central data structure for the SGI. This class is derived from the Cobject class. It defines the name and the other attributes associated with a specific action, as well as some member functions. The SGI application was developed as a dialog based application automatically generated by the MFC AppWizard. The CSGIDlg class (generated by the AppWizard) handles all the controls on the visual interface. These two classes are the most important classes in the SGI application. The implementation of these two classes is described in this section.

The SGIAction class was constructed by hand, rather than generated by the MFC App-Wizard. The definition of class SGIAction is shown in Figure 2.5. Member variable m_Name stands for the name of a specific action; m_Type, m_Text and m_LineNum describe the type, text and PC buffer line number for the control on which the action was performed on; the m_Speech attribute is used to specify mute or speak out options for this action.

```cpp
class SGIAction : public CObject
{
public:
        CString m_Name;

        CString m_Type;

        CString m_Text;

        int m_LineNum;

        int m_Speech;


        SGIAction();

        SGIAction(CString name);

        virtual ~SGIAction();

        int IsEmpty();

        void SetAttr(CString key, CString value);

        CString GetAttr();
}
```

Figure 2.5 SGIAction Class Definition

- void SGIAction::SetAttr(CString key, CString value)

This function sets the value of an attribute.

- CString SGIAction::GetAttr( )

This function retrieves all attribute values for a specific action.

- int SGIAction::IsEmpty( )

If this function returns 0, it means no real action object is substantiated.

Class CSGIDlg is much more complex than class SGIAction. All functions within this class can be put into three categories based on their functionalities.

2.3.1.1. Category A: Action Display

- CObList* CSGIDlg::GetActions( )

This function reads from the Macros.jss and Macros.jsi files. It matches actions in Macros.jss with the corresponding keystroke sections in Macros.jsi and handles the WaitForWindowWith-Name( ) and MacroSetWindowText( ) exceptions. It then instantiates action objects based on the obtained information and puts all actions in an Object List, which is returned upon finishing.

- BOOL CSGIDlg::OnInitDialog( )

This function is invoked when the SGI starts. It calls the GetActions( ) function and displays all recorded user actions in a ListBox. The highlight is set at the first action by default. When users navigate among actions, the associated attributes for the highlighted action are displayed in an EditBox.

- void CSGIDlg::OnReload( )

This function reverts everything to the status that no user operation has been performed.

2.3.1.2. Category B: Action Modification

- void CSGIDlg::OnBUp( )

This function is invoked when the "Up" button is pressed. It moves the highlighted action up one row.

- void CSGIDlg::OnBDown( )

This function is invoked when the "Down" button is pressed. It moves the highlighted action down one row.

- void CSGIDlg::OnRemove( )

This function is invoked when the "Remove" button is pressed. It marks the highlighted action for deletion and adds the word "Deleted!" to the highlighted action, but does not actually remove the action from the action list.

- void CSGIDlg::OnBUnremove( )

This function is invoked when the "UnRemove" button is pressed. It removes the "Deleted!" indication from a highlighted action marked for deletion.

- void CSGIDlg::OnBPurge( )

This function is invoked when the "Purge" button is pressed. It permanently removes actions that are marked for deletion.

- void CSGIDlg::OnRadio4( )

This function is invoked when the radio button labeled Spoken output is checked. The speak-out attribute (m_Speech) for the highlighted action is set to 1.

- void CSGIDlg::OnRadio5( )

This function is invoked when the radio button labeled Not Spoken output is checked. This radio button is checked by default. The speak-out attribute (m_Speech) for the highlighted action is set to 0.

2.3.1.3.   Category C: Plan Recognition

- CStringList* CSGIDlg::GetScriptList( )

This function reads from Basic.jss and returns a CstringList that contains the names of the script representations contained in the file.

- CString CSGIDlg::genSay(POSITION *ppos)

This function replaces an action sequence that has a speak-out attribute of 1 with the function call ReadText (start_line, line_number_count_down).

- CString CSGIDlg::genTypeString(POSITION *ppos)

This function combines successive keystrokes into a text string. For example, it will combine TypeKey("B"), TypeKey("o"), TypeKey("o") and TypeKey("k") into string "Book".

- CString CSGIDlg::ActionEvidence(SGIAction * act,POSITION *ppos)

This function submits observed actions (obtained from JAWS Macro Recorder) to PRNs. It returns a string that is extracted from the parameters in raw actions. For example, it returns "Yahoo Home Page" when given WaitForWindowWith-Name("Yahoo Home Page"). It returns

a text string when given a sequence of TypeKey actions as mentioned above. This returned string is used to construct the optimized output script.

- void CSGIDlg::OnBGenScript( )

This function is invoked when the "Produce Script" button is clicked. It implements the functionality of the Plan Recognition Engine (PRE). As described in section 2.2, the function takes actions in sequence from the action list and submits the observations to the PRNs. When it finds one plan with posterior probability greater than 0.85, it continues to check whether subsequent actions are still within this plan. Once finished, it replaces the action sequence with a function call that represents the plan. Afterwards, the SGI retracts all the observations submitted to the PRNs. The program then continues the inference procedure from the first action not included in the sub-plan. Finally, it outputs all the function calls in an EditBox labeled "Script".

- void CSGIDlg::OnOk( )

This function is invoked when the "OK" button is clicked. The program asks whether to save the optimized script or not. If the user chooses to save, it will ask the user to name the script and then save it on the disk. At the same time, the program calls ASPRN.exe and creates a PRN based on the optimized script.

2.3.2. Visual Interface

The SGI interface has been tested by several visually impaired users, all of whom found the SGI easy to learn and operate. The SGI's user interface is shown in Figure 2.6
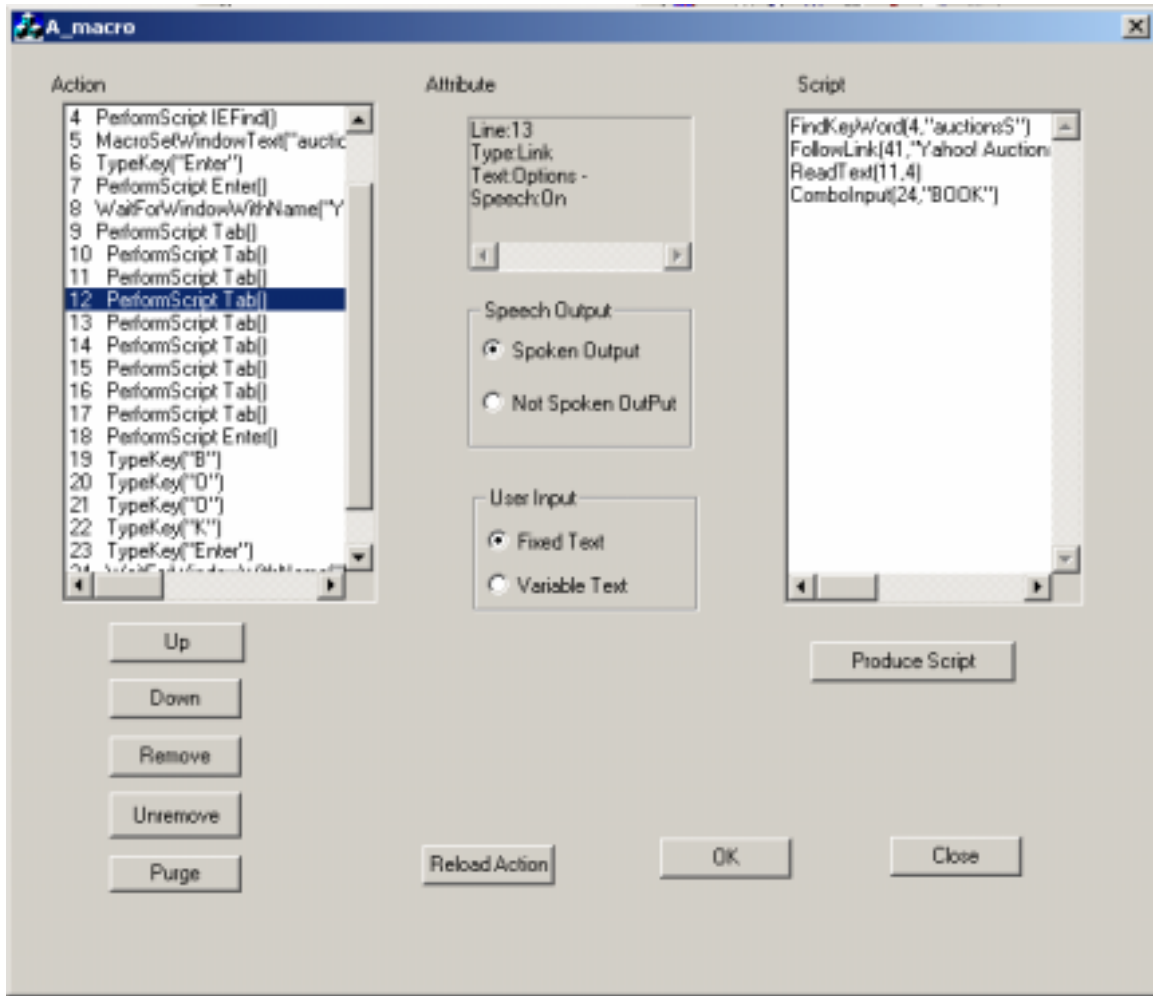
Figure 2.6 Visual Interface of SGI Application

The SGI displays the recorded actions in the left-hand list box. As the user navigates among the actions in the list box, the information associated with each action is displayed and spoken aloud. The buttons below the left list box allow the user to manually modify the action list. The user can move an action up or down within the list, mark an action for deletion, and permanently remove (purge) all actions marked for deletion from the list. The "Speak Output" radio buttons allow the user to specify whether a specific action in the optimized script should produce spoken output. When the user clicks on "Produce Script" button, the optimized script is displayed in the right-hand edit box.

2.4. Scripts playback in JAWS

In order to be able to play back users' actions, new functions are added into JAWS script file that is associated with Internet Explorer (IE) application (i.e. Internet Explorer 5-6.JSS).

- ReadText (int line, int count_num)

This function first moves the cursor to the line indicated by the first parameter, then reads out the contents from this line down n lines as indicated by the second parameter.

- FindKeyWord (int line, string word)

This function first moves the cursor to the line indicated by the first parameter. At this location, it brings up a JAWS Find dialogue box and enters the string indicated by the second parameter into the edit box, then follows with the "Enter" key. It simulates the actions of looking for a key word on a web page.

- FollowLink (int line, string word, string name)

This function simulates the action of clicking on a specific hyperlink and bringing up a new web page. The first parameter is the line number of the link in the JAWS Virtual PC buffer. The second parameter is the window title of the new web page (for example, "Yahoo! Auctions - Find Great Deals and Unique Col"). The third parameter here is the text that describes the hyper link.

Two methods are used to locate at the right spot on a web page. One is the line number of the hyperlink. That is, the cursor moves to the line indicated by the first parameter of this function, and then simulates a left mouse click there. This works well for web pages whose layouts do not change frequently. But some popular web pages like Yahoo and CNN may change their layouts every few hours. Another more stable method is to locate by link text. A new

function called GotoLink (string name) was written to list all the links on a web page. It compares every link text with the name indicated as its' parameter until it finds a link with this name. It then moves the focus to the link and clicks on it.

- ComboInput (int line, string word)

This function simulates actions of finding an edit box and typing a text string into the box. The first parameter is used to locate the edit box on the web page. Once located, a left mouse button click sets the cursor in the edit box and the program types the word indicated as the second parameter into the edit box.

3. Usability Testing with the Intelligent Screen Reader

Usability Testing focuses on determining if a product is easy to learn, satisfying to use and contains the functionality that users desire [27]. Usability testing on the intelligent screen reader system was conducted twice with visually impaired users. The methods and the outcomes of each testing session are presented in this chapter.

3.1. First Usability Testing Session

The first usability tests were conducted in February of 2003. Before this, the SGI had never been used by a person with a visual impairment. The goal of this testing session was to determine:

1) Do users find it is easy to figure out how to operate the SGI?

2) Can users complete specific tasks (such as rearranging items within the action list, removing items from the action list or producing an optimized script) successfully?

3) How quickly can users complete specific tasks?

3.1.1. Methods

The first usability testing session was conducted in February of 2003 over a three day period at the corporate headquarters of Freedom Scientific in St. Petersburg, Florida. Six experienced JAWS users (five of them are legally blind), all employees of Freedom Scientific, participated in the testing (two participants completed the study each day). Before participating, each subject was given a description of the intelligent screen reader project and the role of the SGI within the system, as well as some general information about the SGI's functionality. The participant was then asked to use the "Tab" key to navigate through the components of the SGI's

user interface to become familiar with it. During this testing session, investigators recorded a sequence of actions within Internet Explorer in advance since the purpose of this testing session was to focus on the layout and functionality of the SGI. When the participant was ready, he or she was asked to perform specific tasks such as finding an action at line 8 and moving it up one row. Participants were not given instructions on how to complete these tasks prior to being asked to complete them. This gave investigators an opportunity to observe how experienced JAWS users "problem-solve" when faced with an unfamiliar interface. During this phase, participants were encouraged to "think out loud" to provide as much insight into their behavior as possible. The investigative team collected their feedback by recording their spoken comments about what aspects of the SGI were most confusing, and what functionality was lacking from the SGI. Throughout the course of the three-day testing session, investigators continued to modify the layout and functionality of the SGI in response to participant feedback.

The user interface of the version of the SGI application initially presented to the participants during testing is shown in Figure 3.1. The most significant difference between that original prototype version of the SGI shown there and the final version of the SGI (shown in Figure 2.3) is the "Edit Action" group. In this group, "Edit On/Off" is a toggle button used to activate and deactivate the other three buttons (labeled "Not Assign", "Next Action" and " Delete", respectively). By default, the "Not Assign" button is checked and nothing is done to the action list. If the user chooses the " Next Action", he is trying to specify the action sequence according to his needs. The " Delete" button allows the user to delete an action from the list. Modification made by the user does not take effect until the "Edit On/Off" button is toggled to the off position.

3.1.2. Results

During the first day of trials, the participants found the "Edit Action" group to be very difficult to work with. Changing the order of actions introduced a significant cognitive load, and deleting an action required three steps (first toggle Edit mode on, then choose "delete action", then toggle Edit mode off), which was inconvenient. Participants wanted to be able to delete an action in one step, and have an easier way to change the order of actions in the list. When asked, they pointed to WinAmp as software that provided a simple interface consisting of "Up" and "Down" buttons for reorganizing items within a list (in this case, songs within a playlist). With this approach, it is not only easy for them to make changes, but also easy for them to track changes as they occurred.

On the second day of the trials, the "Edit On/Off" toggle button was removed from the interface. The three radio buttons were replaced by three push buttons labeled "Up", "Down" and "Delete.". With this change in place, participants figured out how to operate the SGI very quickly. When asked to perform a specific task, they found the right method, but it still took them quite a while to finish the task. Participants noted that when they navigated among the actions in the action list, JAWS reads the contents of each line first, and then announced the position of the line within the entire actionlist (e.g. "seven of fifteen" meant there were fifteen total lines of actions and the current line was the seventh of these). This increased the time necessary to locate the right action within the action list, since the participants had to wait for the position information to be spoken. Participants suggested putting the sequence number of each action at the beginning of each line. Therefore, JAWS always read the sequence number first when one line of action was selected, which saved time when finding the right position within the list. Another suggestion was to add buttons like "Remove" and "UnRemove" that allow the

user to retract deletions. Thus, "Remove", "UnRemove" and "Purge" buttons were added to the interface for later testing. The "Remove" button added a "Deleted!" identifier to the highlighted action and "UnRemove" button retracted this identifier. Only the "Purge" button allowed users to permanently remove actions marked for deletion.

By the third and final day of trials, the layout of the SGI was very similar to the final version. Participants felt it was easy to determine how to operate the SGI even if it was the first time they were using the interface. They were able to successfully finish specific tasks such as moving an action at a particular line up or down one row, purging all the actions marked for deletion, and producing an optimized script for the modified actions. Subjects were also satisfied with the speed with which they could perform these tasks.
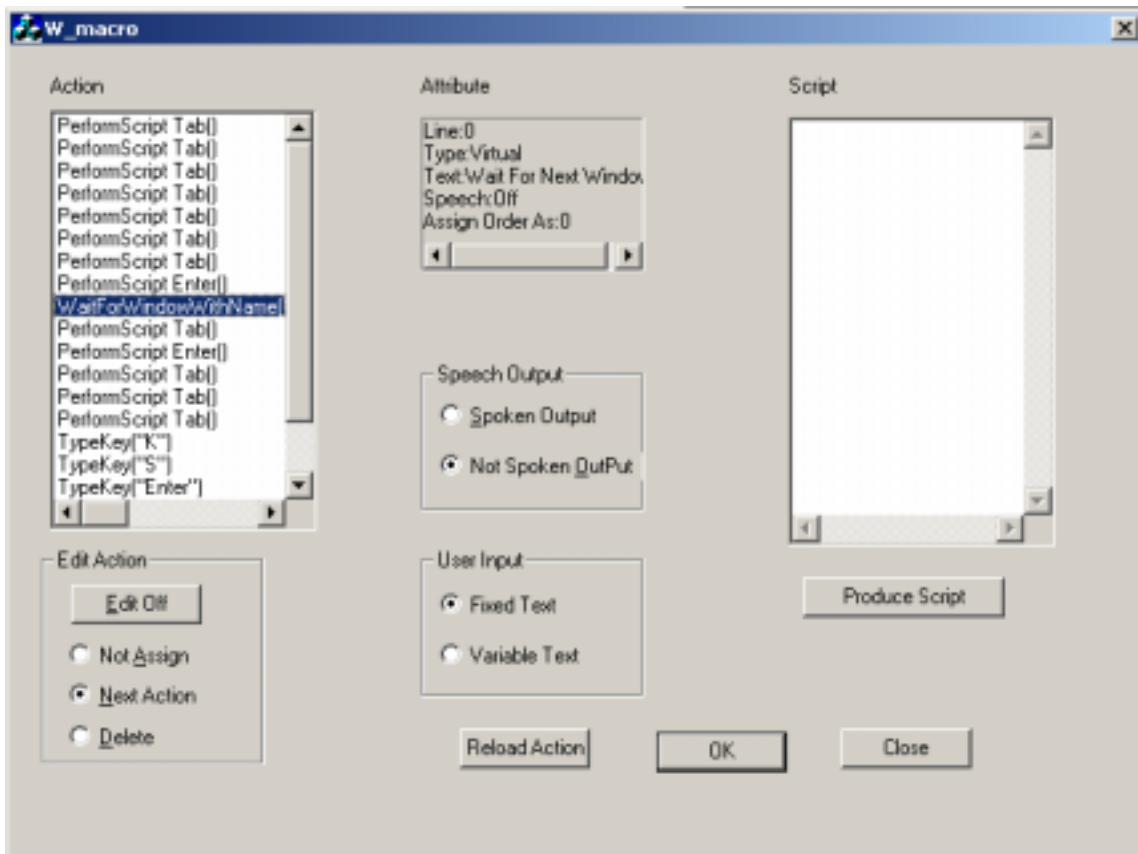

Figure 3.1 The User Interface of the Initial Version of the SGI

3.2. Second Usability Testing

The second round of usability testing session was conducted in July 2003. This usability testing focused on the functionality of the entire intelligent screen reader system. The goal of this testing was to determine:

1) Do users feel it is difficult to create useful scripts through the procedure of recording actions, modifying actions and producing an optimized script with PRNs?

2) Do users fully understand the script that was generated?

3) Do users think it is efficient to create a script with the intelligent screen reader system?

3.2.1. Methods

The second usability testing session was conducted over a two day period at Freedom Scientific. Six experienced JAWS users (only one of them participated in the first testing), all employed by Freedom Scientific, participated in the testing with three participants completing the study each day. The average period of time spent by each participant in usability testing was one half hour. General information on the intelligent screen reader project was given to each participant before the testing. Each participant then was asked to complete the following procedures that are fully described in the next three sections.

Step 1. Browse a predetermined web page and record a sequence of predetermined actions.

Each participant began the experiment with the same web page (www.yahoo.com) loaded on the same browser software (Internet Explorer 6.0). Each participant was asked to perform a specific sequence of actions in order to complete a task. First, the participant was asked to find the "weather" link and activate the link to bring up the yahoo weather web page. Next, the

participant was asked to find the "Enter zip code" edit box, type a five digit zip code into the box, and press the enter key. Once the web page displaying the weather forecast was displayed, the participant was asked to find the "travel weather" link and active the link.

Before recording actions, all the participants were told that the system was limited to a small set of keystrokes (e.g., Tab, Enter, JawsFind) and tasks (FollowLink, FindKeyWord and ComboInput). Participants were also told not to worry about any erroneous keystrokes, since the SGI would allow them to edit or modify their actions during a later step. While completing the task, the participants were allowed to get assistance from the investigator if they needed to be reminded about the sequence of actions they were supposed to perform.

Step 2. Open the SGI and edit the sequence of actions

The participant was asked to open the SGI application and examine the sequence of actions recorded in step 1. If the participant had made any mistakes in step 1, the participant was asked to correct those mistakes using the SGI. Otherwise, the investigator chose specific actions for the participant to modify, and specified the modification to be made.

Step 3. Generate a JAWS Script

Once Step 2 was completed, the participant was asked to generate an optimized script using the SGI. The user was asked to examine the script, and report whether he or she understood the script. Since the functions for script playback had not yet been implemented, the participant was not able to run the script within JAWS to see how it worked. However, the procedure for playing back the optimized script was explained to each participant.

Finally, method used by the belief networks to recognize the tasks being performed by the user was explained to each participant. When each participant understood that the threshold for plan recognition would affect the output script, they were asked to try another SGI

application with a higher threshold for plan recognition. Everything other than the threshold was kept the same as before. The participant was asked to generate an optimized script with the new SGI, examine the output script and compare it with the one generated previously.

3.2.2. Results

Only one participant experienced difficulties in recording actions with the JAWS macro recorder. The participant was not familiar with using an ergonomic keyboard and kept making keyboarding mistakes during recording. At times he even went back and forth between two web pages. It took him 5 minutes longer than the other participants to finish the specific task as instructed and he was presented with a long list of actions by the SGI. He suggested a tutorial system for macro recorder. In contrast, other participants felt very comfortable with recording their actions with JAWS macro recorder. Some followed the instruction very well and only made mistakes when typing a numeric string into the "zip code" edit box. The participants had the opportunity to correct these mistakes in the SGI editor window later. Some participants used JAWS commands that were not understood by the SGI. For example, in order to find "weather" link on Yahoo home page, some participants used the key combination "Ins+F7" to display a list of links within the web page, and then typed "W" (the first letter in weather) several times until they found the weather link, then pressed enter to move the focus to that link. The SGI produced incorrect scripts based on this sequence of user actions since the PRNs were not sophisticated enough to recognize this type of action at this time.

All of the participants found the SGI easy to understand and to operate. All participants successfully completed the modification tasks using SGI. Some of participants even figured out that the SGI supported multiple line selection. They selected multiple actions and clicked on the

"Remove" button to mark all the selected actions for deletion, and then purged them all at one time. They also noticed that the SGI offered accelerator keys for each button, which enabled them to operate more quickly. All of the participants thought the SGI provided an efficient way to generate a useful script.

All of the participants understood how PRNs were used to pick out plans that were most likely to happen. The only thing that was not satisfactory to the users was the fact that PRNs could only deal with limited user actions and plans. Some experienced JAWS users suggested integrating more alternative plans into the PRNs. For example, as an alternative to pressing the "Tab" key many time or using the "JAWS find" dialog to locate at a specific link, they suggested using "Ins+F7" to bring up a link list and find the link in the list. Alternatively, they could use several "Down Arrow" or "Up Arrow" buttons to locate the right spot. The more alternative plans in PRNs, the more powerful Plan Recognition will be. In addition, new plans representing frequently performed tasks such as activating the icon of an application on the desktop using PC cursor or JAWS cursor should also be added to PRNs.

Another recommendation given by the participants was to allow the users to specify the threshold by themselves, rather than hard-coding the threshold into the source file. All the participants understood the threshold would affect the output function calls. They hoped they could adjust the threshold so that it would fit all plans if they added new plans into the library in the future.

4.   Conclusions and Future Work

An intelligent screen reader system has been developed in order to provide effective computer access for persons who are visually impaired. The system consists of three parts: the JAWS macro recorder, the SGI (Script Generation Interface) and ASPRN (Automated Synthesis of Plan Recognition Networks). The SGI obtains the user's actions and action attributes which were recorded by the JAWS macro recorder, allows the user to manually modify the action list, and then makes use of PRNs to infer plans that are most likely to happen based on the user's actions. The script generated by the SGI allows the user to perform this same task more efficiently when it is undertaken again in the future. The work described in this paper focuses on integrating the SGI with JAWS and ASPRN, and implementation of the SGI application.

Preliminary usability testing was conducted with advanced JAWS users who are visually impaired at Freedom Scientific with satisfactory results. These users understood the procedure of recording actions with the macro recorder, modifying actions and generating a useful script with SGI. The intelligent screen reader was easy to learn and operate for these users. . In its current state of development, this system is not powerful enough to demonstrate full functionality, since it only deals with a limited set of user actions and plans. It does however demonstrate that the SGI is a useful concept and can contribute to improved user interface.

In the future, more plans need be developed and added into PRNs. These plans should include alternative plans for those that already exist in the PRN library, as well as new plans that represent commonly performed tasks. It may also improve the usability of the system if users are able to adjust the threshold for plan recognition. This will give them the individual flexibility to add or delete plans in PRNs without affecting overall system performance. In addition, it is

suggested that a tutorial system that is able to teach the new or novice user how to operate this intelligent screen reader be included when the intelligent screen reader is commercialized.

ACKNOWLEDGMENTS

Contact Information for Freedom Scientific Inc.

Freedom Scientific Blind/Low Vision Group

31st Court North

St.Petersburgh, FL 33716

Phone: (800)444-4443 Email: Info@FreedomScientific.com

URL: http://www.freedomscientific.com/

Contact Information for Intelligent Reasoning Systems Inc.

Intelligent Reasoning Systems

4976 Lassen Drive

Oceanside, CA 92056

Phone: (760)806-1497 Email: marcush@home.com

REFERENCES

1. J.McNeil. Americans with disabilities:1991-92. U.S. bureau of the census. Current population reports. Government Printing Office, 1993, pp70-33, Washington, DC.

2. L.Bassi, A.Gallagher, E.Schroer. The ASTD Training Data Book. American Society for Training and Development. 1996, Alexandria, VA.

3. C.Earl, J.Leventhal. A Survey of Windows Screen Reader Users: Recent Improvements in Accessibility. *Journal of Visual Impairment and Blindness*. 1999;93.

4. J.Gunderson, R.Mendelson. Usability of World Wide Web Browser by Persons with Visual Impairments. *Proceedings of the RESNA'97 Annual Conference*, 1997, p330-332.

5. Web Content Accessibility Guidelines 2.0 available at
http://www.w3.org/TR/2003/WD-WCAG20-20030624/

6. Holly S. Vitense, Julie A. Jacko, V. Kathlene Emery. Multimodal Feedback: Establishing A Performance Baseline for Improved Access by Individuals with Visual Impairments. *Proceedings of the fifth international ACM conference on Assistive Technologies*, 2002, pp 49-56, Edinburgh, Scotland.

7. Wai Yu, Stephen Brewster. Multimodal Virtual Reality Versus Printed Medium inVisualization for Blind People. *Proceedings of the fifth international ACM conference on Assistive Technologies*, 2002, pp 57 -64, Edinburgh, Scotland.

8. Chieko Asakawa, Hironobu Takagi, Shuichi Ino, Tohru lfukube. Auditory and Tactile Interfaces for Representing the Visual Effects on the Web. *Proceedings of the fifth international ACM conference on Assistive Technologies*, 2002, pp 65-72, Edinburgh, Scotland.

9. E. Pontelli, W. Xiong, G. Gupta, A.I. Karshmer. A Domain Specific Language Framework for Non-Visual Browsing of Complex HTML Structures. *Proceedings of the fourth international ACM conference on Assistive Technologies*, 2002, pp180 –187, Arlington, VA.

10. E. Pontelli, D. Gillan, G. Gupta, A.I. Karshmer. Navigation of HTML Tables, Frames, and XML Fragments. *Proceedings of the fifth international ACM conference on Assistive Technologies*, 2002, pp 25-32, Edinburgh, Scotland.

11. Andrea Kennel 1, Louis Perrochon I, Alireza Darvishi. WAB: World Wide Web Access for Blind And Visually Impaired Computer Users. *ACM SIGCAPH Computers and the Physically Handicapped*, 1996:55, pp 10-15, New York, NY.

12. Enrico Pontelli, Tran Cao Son. Planning, Reasoning, and Agents for Non-visual Navigation of Tables and Frames. *Proceedings of the fifth international ACM conference on Assistive Technologies*, 2002, pp 73-80, Edinburgh, Scotland.

13. Kristina Hook. Evaluating the Utility and Usability of an Adaptive Hypermedia System. *Proceedings of the 2nd international conference on Intelligent user interfaces*, 1997, pp 179-186, Orlando, FL.

14. Guy Camilleri. A Generic Formal Plan Recognition Theory, Proceedings of IEEE *International Conference on Information, Intelligence and Systems ICIIS'99*, 1999, pp540-547, Rockville, MD.

15. Bradley A. Goodman, Diane J. Litman. Plan Recognition for Intelligent Interface. *Proceedings of the Sixth IEEE Conference on Artificial Intelligence Applications*, 1990, pp 297-303, Santa Barbara, CA.

16. C. A. Broverman, K. E. Hu, V. R. Lesser. The role of plan recognition in design of an intelligent user interface. *Proceedings of the IEEE Systems, Man, and Cybernetics Conference*, 1987, pp 863-868, Atlanta, GA.

17. E. Charniak, R. P. Goldman. A Bayesian model of plan recognition. *Artificial Intelligence*, 64(1), 1993, pp 53-79.

18. M. Huber, E. Durfee. Deciding when to commit to action during observation-based coordination. *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS)*, 1995, pp 163-170, San Francisco, CA.

19. M. Huber, E. Durfee. On acting together: Without communication. *AAAI Spring Symposium on Representing Mental States and Mechanisms (AAAI Press)*, 1995, pp 60-71, Stanford, CA.

20. M. Huber, E. Durfee, M. Wellman. The automated mapping of plans for plan recognition. *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence (UAI)*, 1994, pp 344-351, Seattle, WA.

21. M. Huber, E. Durfee, M. Wellman. The automated mapping of plans for plan recognition. *Proceedings of the 1994 Distributed AI Workshop (DAI Workshop)*, 1994, pp 137-152, Lake Quinalt, WA.

22. Marcus J. Huber. Plan-Based Plan Recognition Models for the Effective Coordination of Agents Through Observation. PhD thesis, 1996, The University of Michigan.

23. J. Pearl. Probabilistic Reasoning in Intelligent Systems. *Networks of Plausible Inference*, volumn 2. Morgan Kaufmann, 1991.

24. Marcus J. Huber, Edmund H. Durfee. An initial assessment of plan-recognition-based coordination for multi-agent teams. *Proceedings of the Second International Conference on multi-Agent Systems*, 1996, pp 126-133, Kyoto, Japan.

25. JAWS 4.5 Help Document

26. Macro Express Help Document

27. TekSci's Dictionary of Technical Terminology. Available at
http://www.teksci.com/teksci/dictiona.asp

28. All Ages Self-Reported Visual Impairment and Low Vision. Available at
http://www.lighthouse.org/vision_impairment_prevalence_all.htm

29. User Survey: Browsers and scripts. Available at
http://www.w3.org/2000/06/scripting-user-survey

30. U.S. Department of Education. Getting America's student Ready for the 21st Century. Washington, DC, 1996 http://www.ed.gov/publications/sensory.html.