Development of a Automated Mechanical Receptionist

Paul Chung
Thomas Jefferson High School for Science And Technology
Alexandria, Va

January 25, 2006

Abstract

The main objective of this project is to continue development of a robotic receptionist for NRL (Naval Research Laboratory). This will involve adjusting the robot's AI code to be more fitting to NRL and adding features involving a card reader. The AI code was originally programmed for Carnegie Mellon University and needs to be adapted in its algorithms for providing directions and responses to users. The card reader program also needs to be modified so that the robot will be able to identify users through an id card. Users will also be able to start up and shutdown demonstrations automatically by swiping certain cards for a set number of times.

The adaptation of the AI code will mainly involve C++ and IPC. IPC (inter-process communications) will allow the process that interprets user input to communicate with other programs running on the robot. Ensuring that the robot interprets requests for general information, directions, and people correctly is a critical part of the project. The requests must then be transferred to the appropriate algorithm, which will provide the response. The card reader program will also use MD5 sums to identify different id cards. MD5 sums, which are different codes assigned to cards, are used to differentiate cards that are authorized and known from unknown cards. The program identifies users based on the MD5 sums stored in a database and uses IPC to let the robot's AI know who the user is. A process management application, Microraptor, will also be used by the card reader to facilitate the automatic startup and shutdown of demonstrations. When a user swipes a certain card for a set number of times, the Microraptor central server will startup programs required to run a certain demonstration. The process management application is necessary to make sure the dependencies of the programs are running and to monitor the status of all of the running processes.

Introduction

The purpose of this project is to adapt a robotic receptionist for NRL and to add features to the currently existing robot and artificial intelligence. Adapting the receptionist mainly involves reprogramming some of the code that handles giving directions and determining responses. The problems with the current receptionist are primarily caused by the originial artificial intelligence, which was programmed for Carnegie Mellon University. When retrieving directions, the receptionist will sometimes give directions to buildings located in CMU. Also, the receptionist greets users as if it were at the campus instead of NRL. There are also some other issues with parsing data from databases that is used to find information about specific people.

Additional features will involve the card reader. One aspect of this goal is to have the receptionist recognize users based on what card they swipe. Currently, the robot can only recognize when a card has been swiped and whether it has been registered before. Another feature that will be added using the card reader will be the automatic startup and shutdown of robot demonstrations. This is somewhat functional in a specific demonstration in that a user can utilize a process management application to run the programs with only one command. There are, however, several other demonstrations that require significant experitse and time in order to run. The automatic startup and shutdown also needs to be run with only a swipe of a card rather than manually entering the process management application and starting the

demonstration.

Adapting the receptionist for NRL will give it more functionality. Once the artificial intelligence is reprogrammed, visitors and other users at NRL will be able to find directions and people easily by asking the receptionist. The automatic startup and shutdown will also save significant time and effort in setting up the demonstrations. The manual method of running the demonstrations requires several hours and knowledge of all of the processes dependencies and environment variables, but the automatic startup will run the demonstrations with a simple card swipe.

Background

The robotic receptionist was programmed to be able to have basic conversations with users as well as direct visitors to certain buildings, rooms, or people. Determining what kind of response the robot should give was based on recognizing certain words and ascertaining the nature of the user's input. If it was determined that the user was asking for the location of a building, room or person, the robot would query certain databases to acquire the requested information. This process involved several different programs that utilized IPC to commmunicate between them. One process, named interact, would parse the user input and if the user needed directions, it would request directions from another program (named giveDirectionsNRL) that would search a database. In the case that the user was looking for a person,

the program would query a database and return the such information as the name, phone number, and the room and building of the person.

The card reader was also previously implemented into the robot, but the receptionist could only detect when a card was swiped and whether it was an "authorized" card. An authorized card was one that was stored in a list that contained an MD5 sum associated to the card and the id of the user with that card. The MD5 sum was obtained from reading the magnetic stripe on cards such as calling or credit cards using a separate program that stored the sums into a list. The card reader program implemented into the rest of the receptionist programming code waited for a card to be swiped, read the MD5 sum from the card and compared it to the ones stored in the file of authorized cards. Using IPC, it communicated with the other programs to let the user know through the interface that a card was swiped and whether it was authorized or not.

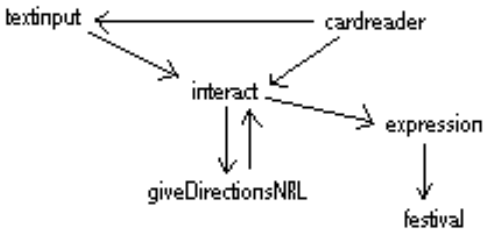
The automation of the startup and shutdown sequence was already somewhat functional in one demonstration. This demonstration was called Malorie and had basic user input through a keyboard. It was limited to giving responses through a monitor and speakers rather than including movement. The processes used by Malorie were listed in a file along with their dependencies, environment variables, directory location and other necessary information. This file was read by a process management application, microraptor, that could run and stop all the processes with one command. Using microraptor made monitoring the processes much easier and starting and stopping

the processes involved less expertise with the dependencies and environment of each program.

Development

The first aspect of this project was adapting the AI for one of the robotic receptionists, Malorie. Research of how the AI determined its responses was required in order to solve the problems of Malorie giving incorrect directions and greetings. Once the setup of the AI was researched, solving the issues was a matter of localizing the source of the problems and studying specific programs in detail. (Appendix A contains some details about this aspect of the project).

A crucial part of the Malorie robot receptionist was IPC. This was used to allow the several programs that formed Malorie's AI to communicate with each other. The passing of data from certain processes to others is diagramed below:



The program that handled user input through a keyboard was called textinput. Textinput passed the user input to Malorie's main AI module, named interact. Interact interpreted the user's input and whether the user wanted directions, information about people, or wanted to have a conversation. The first two cases were not actually distinguished in interact's interpretation methods but in either case the input was passed to another program called giveDirectionsNRL.

The process giveDirectionsNRL used different methods of retrieving information for requests for people and for locations. If the user asked for the whereabouts of a person, the program used the system command to query a NRL database that contained the requested information. The command was whois -h whois.nrl.navy.mil name ¿ filename and its output was a file that contained either no names, one name with associated room number, phone number, e-mail, and so on, or multiple names with a few items of information. A few errors with Malorie giving incorrect responses to input regarding people were caused by parsing errors in this program. giveDirectionsNRL misread the output file from the whois command and sometimes gave incorrect information or claimed there were multiple people at NRL with a name that actually was not in the database. These problems were solved by rewritting the parsing code to more accurately interpret the output file and determine if there was one, multiple or no names found that matched the user query.

In the case that the user needed directions, giveDirectionsNRL searched

a text file containing specific building names, called buildingNames.txt. If the user input included a name included in the file, giveDirectionsNRL obtained a building code associated with the name. It then used the code to find the directions, which were prewritten in a header file, buildingDefines.h. The problem caused by this algorithm was that it sometimes gave incorrect directions. One cause was that the directions were not rewritten for NRL in the bulidingDefines.h file. Another cause was that the algorithm returned a default building code if the user requested the location of a building that was not stored in buildingNames.txt. The default building code was associated to a building in CMU, Newell Simon Hall. Problems that resulted from the first cause were solved by rewritting the directions. The second cause was fixed by changing the default building code to one that was associated to the visitors center at NRL and informing the user when the building requested for was not found.

There were otherer problems with Malorie's AI that caused it to greet users as a receptionist for CMU and occasionally do monlogues written by the drama department at CMU. Both of these problems were caused by the interact program. The first issue was caused by interact loading an old script file used for the receptionist at CMU. The sciprt file had prewritten greetings and responses that were loaded by interact. At first, interact loaded the correct script file but then, as it called a function named initializeExpression, it loaded the incorrect file. The monologues could also be turned off by simply changing a boolean in interact called doMonologues. While compiling

the code, however, there was an issue with tdl (task description language) libraries, which were needed for IPC. Interact could thus not be compiled and the issue was unresolved.

The next aspect of the project was adding a feature to the card reader that would enable Malorie to recognize a user based on a card. The card reader was already implemented into the robot with the program entitled cardreader. This program waited for a card to be swiped, obtained its MD5 sum and compared it to ones previously stored in a file (authfile) that contained a list of authorized cards and the users associated with them. The cardreader program could already recognize the user but there was no way for the other modules of Malorie's AI to know who the user was through the card. This was solved by adding an IPC message that would be sent out by cardreader to textinput. The message would say "my name is ¡user name¿" with the user name being the one stored in authfile. Textinput would then pass the message to interact and it would react as if the user had typed in his name in the keyboard.

The last aspect of the project was automating the startup and shutdown of Malorie and robot demonstrations. The goal was to allow a user to swipe a card once to startup a demonstration and swipe the same card twice to shut it down. This utilized microraptor to run the processes in order according to their dependencies and shut them down quickly. A method of associating certain cards with commands was created that was implemented into the cardreadper program. The method involved a file of entries with each entry

specifying a user id, number of card swipes, and a list of commands to be executed when the card with the specified id had been swiped the necessary number of times. This file was read at the startup of the cardreader program and the entries were stored in a vector.

The cardreader program was also adapted to be a stand alone program that could be used with any robot demonstration. This involved removing the IPC commands that would attempt to contact other processes that were contained in Malorie's AI. A method of counting consecutive card swipes also needed to be added to cardreader. This was done by utilizing signals so that a signal would be sent at the end of a period of time after the previous card swipe. The signal would be caught in a fucntion that would search the vecotr of stored entries for a matching user id and number of card swipes. If a matching entry was found, the commands associated with that entry were executed using the system command.

To use the entry feature of the card reader to automatically start and stop demonstrations, mrterm was necessary. Mrterm is a text based interface to microraptor, which runs as a daemon. Argument can be passed to mrterm, which will then be passed as commands to the daemon. This is ideal for the card reader feature since the command to run or kill all of the processes of a demonstration can be passed to the daemon in a system command. Appendix B contains the automatic startup and shutdown method description and details on the entry file for the card reader.

Results

Most of the goals were met in the first aspect of the project. Malorie now correctly gives directions to buildings inside NRL with the rewritten directions in buildingDefines.h. It also recognizes when requested buildings are missing from the header file. When retrieving information about people, Malorie correctly interprets the output file from the whois command and can identify when there is multiple or no matches of the name requested. The only remaining issue is that interact still causes Malorie to load the wrong script file and engage in extraneous monologues.

The additional card reader feature of recognizing users based on cards is functional with only a few problems. If the user repeatedly swipes a card, Malorie will respond as if the user typed it into the keyboard several times. The response will be along the lines of, "you already said that" when the user technically hasn't put any input at all. There is, however, a built in timer that pauses the detection of cards after a swipe. Thus this glitch would require a user to swipe his card repeatedly over several seconds. Also if textinput isn't running, Malorie will be unable to identify the user since cardreader needs to communicate with textinput.

The automatic startup and shutdown of demonstrations is also functional with problems caused by flaws in the demonstrations rather and microraptor rather than the method itself. The version of microraptor that runs on the robots is outdated and has several glitches (i.e. does not properly terminate

processes, causes some processes to become defunct). Some of the programs in the demonstrations are also not fully functional or need to be running for a certain amount of time before its dependents can be run. Since microraptor will run depedent programs immediately after their dependencies are running, this can cause issues. That problem, however, can be resolved by inserting sleep commands in the entry read by the cardreader program.

Conclusion

Most of the project's problems were solved and goals achieved. Further research into tdl may lead to a solution of the last problems with Malorie's AI and also prevent compiling issues in the future. Updating microraptor on the robot could also be a future goal, since that would solve some issues inherent to the older version.

Appendices

```
\textbf{Appendix A}
Fixes:

- Giving directions to CMU's Newell-Simon Hall when giving directions to another N
This happened when the building being requested was not listed in malorie's direct

Room number parse includes a / which gets spoken
If person's name is not in nrlwhois at all, then it still says multiple entries, b

The code that parsed the building and room number assumed that the building number
```

```
while (getline(phoneFile, currstr)) {
    if (i > 5) {
        cout << "No more info" << endl;       return 0;
      // Do nothing
    } else if (i == 2) {
        //cout << currstr << endl;
        //cout << "Name is " << currstr << endl;
        if(strcmp(currstr.c_str(),"No match found") != 0){
          //cout<< "Found a name" << endl;
          *fullName = extractFullName(currstr);
          cout << "The 15th char is " << endl;
          cout << currstr.substr(15, 1) << endl;
          foundName = true;
        }
        else  {
          //cout<< "Found no matches" <<endl;
          *fullName = "";
          foundName = false;
        }
        //if(!strcmp((*fullName).c_str(),"bad"))
          //return 2;
        //cout << "I think the name is " << *fullName << endl;
//foundName = verifyNamesMatch(name, *fullName);    //<<-- Function returns false
        //cout << foundName << endl;
    } else if (foundName) {
      // Usually, the second line is phone number, but sometimes it is the room
      // Treat phone number as a line containing only digits, spaces, or dashes
      //cout << "found the name, now reading the rest" << endl;
      //cout << i << endl;
        if (i == 3) {
          if (currstr.substr(0, 7) != "E-mail:")
            return 2;
        }
        else if (i == 4){
          j = 0;
          k = 0;
          x = currstr.find("/");
```

```
          *building = atoi((currstr.substr(9,x-9)).c_str());
          cout << "Building is " << *building;
          *roomNum = currstr.substr(x+1,currstr.length()-x-1);
          cout << "The room is " << *roomNum;
        }
    }
    i++;
  }
  return 1;
}
```

(Some of the spacing got messed up when copying from the terminal to this document

Remaining Problems:

When malorie starts up, expression attempts to load the malorie script file, but t

Lines 1478-1490 of challenge/src/tasks/roboceptiontist/interact.cc:

```
#define VALERIE_FSM_LOCN "/src/head/scripts/valerie/valerie%d.fsm"

static void readDefnFile (void)
{
  char* chalRoot = getenv("CHALLENGE_ROOT");
  if (chalRoot == NULL) chalRoot = DEFAULT_CHALLENGE_ROOT;
  char *defnFileName = (char*)malloc(sizeof(char)*(strlen(chalRoot) +
              strlen(VALERIE_FSM_LOCN)));
  sprintf(defnFileName, "%s" VALERIE_FSM_LOCN, chalRoot, monologueWeek);

  HEAD_read_file(defnFileName);
  free(defnFileName);
}
```

(initializeExpression is located just below this function)

Malorie still does monologues, which lock up keyboard input and can take a while t

Expression can still crash if the user types enough jibberish and will sometimes e

Finally, I didn't know how to get to the gym so when replacing the directions to (

\textbf{Appendix B}

To run the wax demo using the cardreader:

Turn on sabre and wait until it's done booting
ssh wax@sabre (use -X if you want to open claw to monitor the processes)
export CENTRALHOST=sabre:1382
cd cardreader/bin
./cardreader -p /dev/ttyS1
swipe the appropriate card
swiping the card twice should shut down all the processes (except name server and

The appropriate card can be found by looking at the cardreader configuration file

The cardreader program for the wax demo was changed so that it does not require ar

The syntax for the cardreader configuration file is:

newentry
number of swipes needed
card name
command(s)
endentry

Any text outside of the newentry and endentry labels is ignored.

The card name is the name associated with the md5 sums in the authfile, which shou

The commands are executed through the system command so it will be as if the comma
-e quit (in newer versions -q will also work).

The configuration file for the cardreader for a demo should look something like th

newentry
1

```
cardname
mrterm -e 'load demo.config' -e 'run -a' -e 'quit'
endentry

newentry
2
cardname
mrterm -e 'kill -a' -e 'quit'
endentry
--- End of Example ---

demo.config should be the microraptor configuration file for the processes involve
The path to mrterm should already defined or there needs to be a link to it in the
export CENTRALHOST=localhost:1382 as one of the commands does not work.  Finally,

This is a copy of the config file for cardreader that is used for the wax demo:

newentry
1
waxdemo
/home/wax/mrterm -e 'load /home/wax/sabre_wax_demo.config' -e 'run base' -e 'quit'
sleep 2
/home/wax/mrterm -e 'run poseminder' -e 'quit'
sleep 2
/home/wax/mrterm -e 'run -a' -e 'quit'
endentry

newentry
2
waxdemo
/home/wax/mrterm -e 'stdin speech quit' -e 'kill -a' -e 'quit'
endentry

newentry
1
fakedemo
/home/wax/claw&
endentry
```

```
newentry
2
fakedemo
killall claw
endentry

Here the link to mrterm was located in /home/wax/ so that had to be specified in t

The authfile for this demo had waxdemo and fake demo listed along with their md5 s

The demo could be run by starting cardreader (by sshing into sabre with user wax a

Adding cards to the authfile:
The buildAuthFile program in the cardreader/bin directory can be used to alter the
```


Acknowledgements

I would like to thank my mentor, William Adams, for explaining the topics involved in this project and helping me through the glitches in the software.

Bibliography

[1] Hall, Brian, "Beej's Guide to Unix Interprocess Communication", http:/www.ecst.csuchico.edu/ beej/guide/ipc

[2] Lessard, David, "Malorie Roboceptionist Documentation", Naval Research Laboratory, 2005

[3] Simmons R., Apfelbaum D., "Task Description Language Reference", http://www.cs.cmu.edu/ tdl/tdl.html

[4] Simmons, R., James, D., "Inter-Process Communication", http://www.cs.cmu.edu/afs/cs/project/TCAS/ftp/IPC_Manual.pdf

[5] Smith T., Sellner B., Urmson C., "Microraptor Manual", http://gs295.sp.cs.cmu.edu/brennan/mraptor/source/software/src/microraptor/docs/mrpator_manual.txt