TAB: Talking Alarm Buddy Report

By the *Sleep Hunters* (Chris Tralie and Chris Koscielny)

Description

The goal is to have the user select a time that he/she wants to wake up. When the alarm goes off, it plays an annoying sound and waits for the user to say "stop," at which point it shuts off. The alarm clock will then go into "conversation mode" where it will speak to the user (using voice synthesis) to fully wake him up and make sure he doesn't go back to sleep. The alarm clock will start by telling the user the current time and reading the day's weather based on a previously specified zip code. It will then proceed to play an

interactive verbal game with the user, where the alarm reads two numbers from 2-13, and the user says back the product. If the user makes too many mistakes or is unresponsive (he probably went back to bed), the alarm with make loud/unpleasant sounds to wake him up again. The alarm asks 10 questions to try to wake the user up initially, then reads the news from a previously-specified RSS feed, and then resumes the math game until the user gets up from bed and terminates the program manually.

Motivations / Target Audience

As put by Chris Koscielny:
I sometimes wake up to my loud alarm clock, only to turn it off and fall back to sleep. While my alarm clock successfully jolts me into a mental state capable of turning it off and going back to bed, it fails to wake me up enough to start the day. I hypothesize that the proposed conversational alarm clock will have more success for the following reasons:

1. It's preferable to shut off the alarm by saying/shouting "stop" rather than by getting out of bed and manually disabling the entire system (especially on cold days). Therefore, I won't be as tempted to get out of bed, disable the verbal interface, then go back to sleep.
2. The verbal interface will serve to gradually wake me up by forcing me to do basic thinking. It will also encourage me to wake up by providing me with useful information (e.g. news, weather). The goal of the interface is to bring me to a conscious mental state so that it won't be unpleasant to get out of bed.
3. The ultimate goal is not only to get the user out of bed, but to have the user feel ready for the day by giving him/her useful information during the waking process, improving overall quality of life.

Most people with busy and stressful schedules who need to maximize their productivity while awake (99.999% of Princeton students) will benifit from this alarm clock.

Related Past Work

- "The Voice Interactive Alarm Clock":
  A physical alarm clock that does some voice recognition for commands like "set alarm," "set time," "check alarm," "what is the date," "alarm sound." It can also tell the user the temperature in the room. But since the alarm clock doesn't have web access, it can't do stuff like finding the temperature outside on weather.com or finding news clips. It also doesn't play games with the user. Our alarm clock would also be less expensive since it relies on existing hardware
- Another voice activated alarm clock, but this one can turn on a TV in the morning. Perhaps our alarm clock can feature turning on internet TV.
- Clocky: A physical annoying alarm clock that goes counter to our mission: to engage the user in positive, constructive interaction upon waking. But it's still cool
- The Easy Wake Wrist Watch: A wrist watch that uses motion detection and heuristics to induce sleep cycle to help figure out the optimal time to wake a subject up

Design

Software Environment:
This project was developed under Ubuntu Linux 8.10 using primarily the C language (the only place we had to use C++ was to plug in Festival audio). We used a **make** to organize library dependencies and interdependencies between source files, and **gcc/g++** to compile everything. We used **subversion** to organize code collaboration efforts (especially helpful over break).

Hardware Environment:
microphone:
cheap microphone that came with a computer Chris K. bought almost ten years ago.
telex brand, patent # D395893

Overview of libraries/modules and their functions:

- Festival: Developed by the University of Edinburgh, this library was used for voice synthesis. We plugged it in through *speak.cpp*, so that any file that includes this module simply has to call the function *speak(char*)*. We needed to do some special work in our makefile to link this to the rest of our program, since the Festival API is written for C++ (and our program was in C)
- Sphinx II: Carnegie Mellon has done a lot of work with voice recognition over the past couple of decades, all grouped together under the name "CMU Sphinx." SphinxII is a fast, tried-and-true system from this project that can be used in C. We developed a module in *sphinx_recognition.c* that class functions from this library. It works by filling a buffer of audio from the default microphone for a specified amount of time (while listening to the user), and sends this off to the sphinx system to be decoded into a string based off an acoustic and language model. For our project, we used a custom dictionary (*alarm.dict*) and custom finite state machines (*alarm.fsg*, *numbers.fsg*) to assist in decoding. The dictionary file consists of a list of words that our program recognizes along with their respective phonemes. The finite state machine uses state transitions to define what combinations of words are recognized (e.g. "three hundred fifty six" and "three hundred and fifty six" are valid sentences, but "three hundred hundred six" isn't); the finite state machine isn't required, but it reduces computations and makes the system more accurate.
- libmrss: A C library that facilitates the process of reading rss feeds. We first used this to read a specified RSS feed for the news (the default is "top stories" on yahoo, but we even had it set to slashdot and celebrity gossip at one point), which is encapsulated in the file *news.c*. We also used this library to help read weather, because weather.com has an rss feed returned based on zip code. The weather functionality is encapsulated in the file *weather.c*
- strftime: This is a library in C that can be used to return systen time in any format desired. It is used both to tell the user the current time when he/she wakes up (implemented in *weather.c*) and to check the current hour and minute to see when to set the alarm off (implemented in *alarmconfig.c* with the functions getCurrentHour() and getCurrentMinute())
- glib: This library had some useful functions for manipulating C strings and creating an argc/argv array combination from a given string (the festival library actually requires its performance tweaking parameters in this format). We also used this library to help spawn command line processes, which enabled a simple solution for playing alarm sounds; we simply called the Unix program "*play*" from within the command line to play the alarm sound. This approach is implemented in the file *play.c*.

Other Technical Details:

- We use a very simple configuration file scheme to specify the name of the user, the RSS feed to use, the max number of RSS feed items to read, the zip code for obtaining weather, and the minute and the hour to wake the user up (specified in 24 hour format to disambiguate). The user runs the program from the console and specifies one parameter: the location of this configuration file (e.g. *./TAB config*). The configuration file is then opened and parsed in the file *alarmconfig.c*
- The alarm clock sleeps and occasionally checks to see how close it's getting to the alarm time, and then continues sleeping
- When the alarm first goes off, it plays a loud sound for a couple of seconds, and then listens for three seconds to hear if the user says "STOP." It stays in this loop until it has determined that the user has said STOP

Source Code

A snapshot of our source code is located here

Group Member Contributions

Chris Tralie:

- Got a basic interface to connect with Festival Speech Synthesis System
- Found a C library, libmrss, for reading in rss feeds. Used this library to create modules to read the news (specifying a specific RSS feed) and the weather (weather.com rss feed with a zipcode specified)
- Learned how to read in system time and output in a manner the user can easily understand
- Worked on configuration file scheme for easy tweaking and testing
- Created the base implementation for obtaining audio streams and sending them to be decoded into text using the sphinx2 library
- Learned how to play alarm sounds in the console, and experimented with other ways to load and play raw audio data using the Sphinx library

Chris Koscielny:

- Set up and maintained subversion server that was used for code collaboration throughout the project
- Worked with finite state grammar and dictionary for sphinx voice recognition
- Used the dictionary to help create the arithmetic game that the user plays
- Experimented with an tweaked voice recognition system
-
- Worked on linking C and C++ code together (so that a more elegant solution could be used to call Festival), creating a makefile, improving the modularity of the program, and solving library dependency issues
- Tied all of the modules together into a main program that blocks until the alarm is ready to be set off, and then does logic to determine when the user is awake.

Testing for success

Instead of testing qualitative aspects of the alarm, we decided to quantitatively test features using ourselves as the subjects. This was also better given the time constraints, because not only does Chris T.'s soundcard not work properly under Linux (making recording impossible), but most people do not have linux (and getting the dependencies working so that it can be compile under Windows would be a mess right now), so the logistics of using test subjects to qualitatively test features at this time is not feasible.
Our main metric for success is the accuracy of the voice recognition during the number game. We created a test that reads 30 numbers 0 to 999, and we repeat them back to the microphone. We lied in bed, head facing towards the ceiling instead of towards the microphone resting 7 feet away on a table. This is how we would use the system if we were waking up in the morning. The logic for performing this test is contained in function *number_test()* in the file *number_game.c*

Results (the first column is what we said, and the second column is what the computer thought we said):

Chris K. First time: 21/30 (70%) 9 Wrong: 880 808 522 542 383 308 869 868 63 62 490 408 623 620 877 873 580 558	Chris K. Second Time: 21/30 (70%) 9 Wrong: 763 752 324 824 297 293 408 48 480 48 522 520 158 150 518 588 885 883
Chris T: First Time: 16/30 (53%) Wrong 14 out of 30 5 3 417 416 727 626 803 883 397 396 307 306 697 696 409 12 247 646 653 683 706 76 899 811 360 368 667 663	Chris T: Second Time: 16/30 (53%) Wrong 14 out of 30 161 86 49 45 837 836 992 982 907 307 707 76 239 235 135 133 937 936 665 663 680 63 704 604 432 436 843 823
Mike (a friend down the hall) : 17/30 (57%) 13/30 Wrong: 721 71 997 97	

956 56 889 808 59 55 964 26 291 281 59 58 552 3 219 298 887 87 671 68 281 40	

Overall accuracy: 61%
The two consistent trends here are mixing 5 and 3 and 6 and 7 (especially for Chris T on the latter).

Conclusions / Future Work

Overall, the voice synthesis and the integration with news, weather, and time worked out extremely well. The voice recognition accuracy definitely needs to be improved, however. Chris K. also used the alarm to wake him up one morning when he knew he wouldn't be getting much sleep (so that he would have trouble getting up when he heard the alarm). He reported that it was nice to get news and weather, but that it would have been more soothing to have played music to wake him up initially instead of the annoying alarm.

What we have done is only a very basic prototype for what could be done with this concept. We can still greatly improve voice detection and voice synthesis (better voices are coming out on Festival, and we have the option to make our own voices for the Festival system). There are also heuristics we can use to help improve the accuracy of the number game itself. For example, if we have a three digit number and two out of the three digits are correct and in the correct location, then assume it's right, even if the third digit is off. This would have marked 4 additional numbers correct in the first test for Chris K, 6 more in the second test for Chris K, 8 more in the first test for Chris T, and 10 more for the second test for Chris T, for an overall corrected accuracy of 85% for the Chrises. Since we're just trying to get an overall feel for if the user is awake, these heuristics may be acceptable, even though the probability of accepting wrong answers is slightly higher with this scheme. Also note that if the user got the least significant digits right, then the probability that the whole thing is right is much higher (the philosophy used in md5 checksums). For instance, when 324 was confused with 824, it is clear that it is a computer error.
Another thing that could really help is to have the user calibrate the system by saying a bunch of numbers. This increases user independence. Note that Chris T's accuracy is lower, so Chris T may have benifited from calibration. Note also that sometimes the first digit gets cut off because the user starts talking before the device is ready to record, so perhaps also include heuristics for this.
One of the main features we originally planned on having but had to abandon was a hueristic to deduce sleep cycle, and to wake a user up before the set time if necessary so that they don't fall into deep sleep again. The The Easy Wake Wrist Watch made use of the fact that a person tends to move more while in light sleep, so we could use a similar concept with the alarm clock. And in the spirit of this project (which is supposed to rely on existing hardware), we could use a webcam to help detect motion. There's apparently a library called video4linux that can be used to help with this effort. Another idea is to use a microphone as a motion detector (placing it near the bed).

Aside from adding sleep cycle capabilities, we would also like to add more games that the user can play aside from the arithmetic one, such as a pitch matching game or auditory tic tac toe. We have also discussed making a plugin system so that people from the community can develop their own games for the alarm clock. There is also definitely work to be done with the speech recognition accuracy before the latter can be feasible.

We are both planning to put this up on sourceforge.net and to continue development as an open source project.

References / Acknowledgements

- Festival Speech Synthesis System
- CMU Sphinx. speech recognition library
  - Sphinx II: User Guide
  - Sphinx II: Examples
- Gnome Library Tools: Spawning processes, String utility functions
- GNOME Voice Control (used as an example to help understand spawning processes and a system that uses SphinxII, and we used many of the command line parameters for sphinx from here)
- cplusplus.com tutorial on strftime for outputting system time in a user friendly manner
- The libmrss library for C: A library used to facilitate reading RSS feeds
- Microcontrollor Programming site (helped us to figure out library dependencies needed for festival)
- SVN
- Ubuntu Linux 8.10