SpeechClipse – An Eclipse Speech Plug-in

Shairaj Shaik, Raymond Corvin, Rajesh Sudarsan, Faizan Javed, Qasim Ijaz,
Suman Roychoudhury, Jeff Gray, Barrett Bryant

Department of Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, Alabama 35294-1170 USA

{shaiksj, corvinr, sudarsar, javedf, ijazq, roychous, gray, bryant}@cis.uab.edu

Abstract

Much has been accomplished through the years to enhance the capabilities of individuals that are physically challenged. The goal of computer-assisted adaptive technology is to support the physically challenged in performing tasks on a computer. In addition, adaptive technologies also provide opportunities to enrich a programmer's environment and to increase productivity. It is within these broad parameters that SpeechClipse was developed in order to demonstrate the feasibility of incorporating speech recognition into a popular integrated development environment (Eclipse). The preliminary work described in this paper suggests that various categories of Eclipse users would benefit from a speech-enabling plug-in that uses the Java Speech API.

1. Motivation

The integration of adaptive speech technology into a software development environment has several advantages. The primary goal of this research is to provide a tool to facilitate software developers with physical disabilities to work in a programming environment without the use of a keyboard. Nielsen [5] stresses the importance of speech technology for the physically impaired to adapt to the real world, and cites multiple examples of such instances. It is frequently the case that assisted technologies for the physically impaired drive innovation in other applications.

Our experience from this project suggests that speech enabling technologies are beneficial to users who need to accomplish a task in the Integrated Development Environment (IDE), while still involved in another task. We have observed that this multi-tasking capability often improved our own productivity while experimenting with SpeechClipse.

2. Background

Speech technology constitutes a vocal communication channel between a human and a machine [1]. A vision for research in this area has been stated as, "to create state-of-the-art spoken language components, and investigate how these disparate components can come together with other modes of human-computer interaction to form a unified, consistent computing environment" [4].

Speech technology can be subdivided into Speech Synthesis and Speech Recognition. Speech Synthesis, also referred to as text-to-speech or TTS, converts text into spoken language using structure analysis and text pre-processing. SpeechClipse is not a speech synthesis tool, but rather focuses on speech recognition. Speech recognition is the process by which spoken language is converted into text or some other form. The major steps of a speech recognizer are grammar design, signal processing, phoneme recognition, and word recognition and result generation.

Recognition of the complexities of a programmer's environment is essential to offering a facility to provide enhancements to productivity. Not only is the typical programmer engrossed in the process of coding software, but he/she is also immersed in a selected IDE. The Eclipse IDE was chosen as a platform for this research project because of its robust and extensible architecture. The research focused on providing a productive enhancement in the form of an adaptive speech 'plug-in' for the Eclipse IDE. [7]

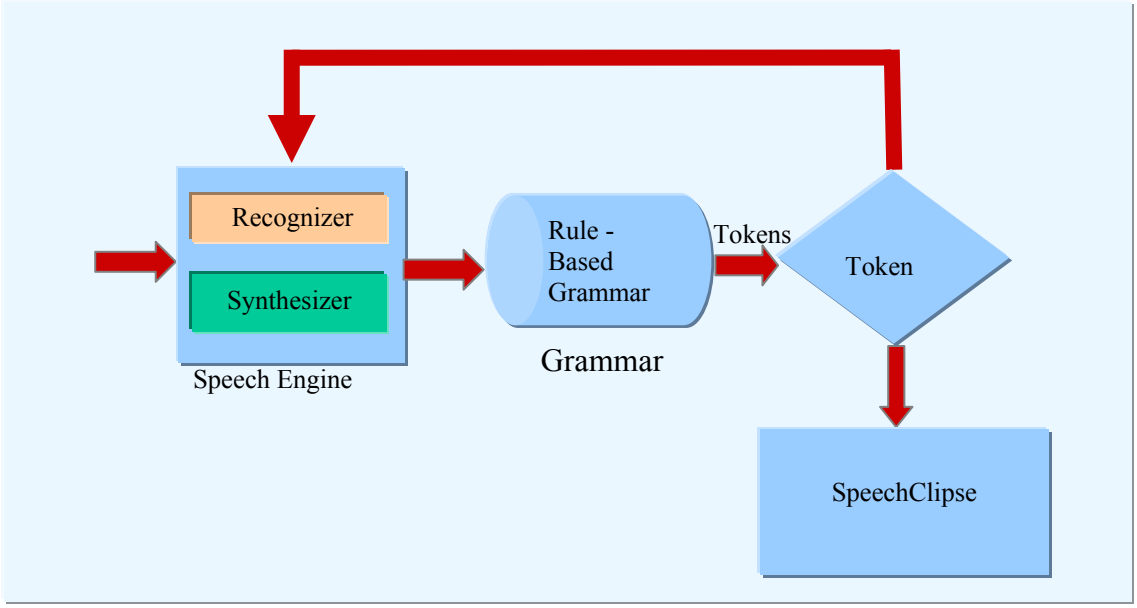

Figure 1: Control Flow Diagram of SpeechClipse

The research focus area is one that offers a sizable benefit in return for the potential work effort. It also appears that little research and development is being done on this topic within Eclipse.

This implementation of the Speech API as an Eclipse plug-in provided the broad reach necessary to offer control of almost any system facility within the chosen environment. Also, it is reasonably easy to use. A programmer's task continuously includes the invocation of external utilities. These utilities can now be summoned vocally without leaving the Eclipse IDE.

3. Implementation Architecture

SpeechClipse uses the CloudGarden[1] implementation of Sun's Java Speech API [3] as the speech recognition API, running on the Microsoft speech recognition engine in the Windows environment. The plug-in is designed in a manner such that the current implementation of the speech recognition API can be replaced easily with a new implementation to support other platforms.

The Java Speech Grammar Format[2] , a platform-independent vendor-independent textual representation of grammars, was used as a standard to specify the grammar for SpeechClipse. This grammar was used to establish the communication path for translating commands into events.

The Java Speech API supports both rule-based grammars and dictation grammars. The current version of SpeechClipse uses only the rule-based grammars to recognize the best possible word from the spoken input. Dictation grammars are more flexible and closest to the goal of unrestricted natural speech input to computers. The implementation of the dictation grammar, along with a rule-based grammar, is a part of our future work. Figure 1 provides details regarding the control flow in the SpeechClipse plug-in design structure.

In the figure, the speech input from the microphone is given to the speech recognizer in the speech engine. The recognized word is then checked with the grammar file to get the best possible token recognition. The tokens are then

[1] http://www.cloudgarden.com

[2] http://java.sun.com/products/java-media/speech/forDevelopers/JSGF

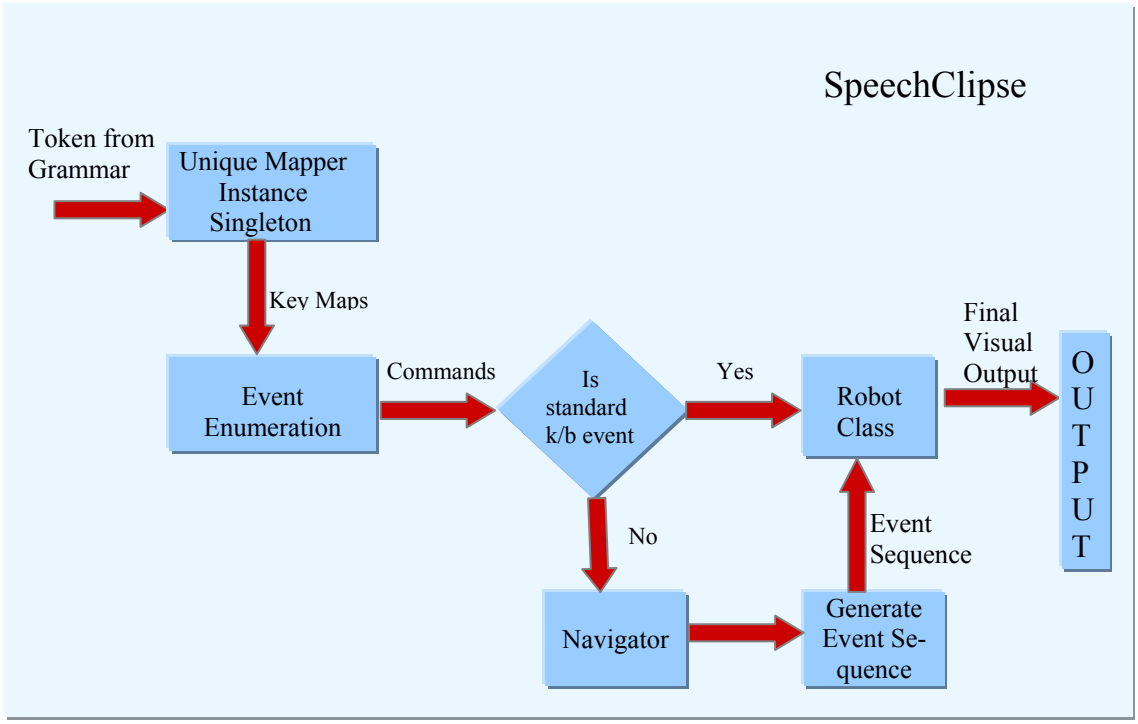

Figure 2: Control Flow within SpeechClipse

passed to SpeechClipse, which uses the tokens as commands to invoke appropriate actions. The recognizer then continues to listen for the next set of tokens.

The final closure of the path between the recognized word and the instantiation of the expected action in the IDE is achieved using Java's 'Robot' class [2]. This class translates the recognized command words from the speech recognition process into simulated keyboard activity. If the command requires initiating multiple actions in the Eclipse IDE, then those corresponding events are implicitly generated and the final action is invoked by the plug-in. Figure 2 represents the internal implementation of the SpeechClipse plug-in.

The tokens received from the grammar are sent to a single instance Mapper which generates the corresponding key maps for action. The Event Enumeration then generates the equivalent commands. These are then checked to see if the event is a standard keyboard event or an Eclipse command. If it is an Eclipse command then the Navigator generates the equivalent event sequence. In either case, the final control is passed to the Robot Class, which produces the final output.

In order to effectively control all the desired aspects of the Eclipse environment it was required to fully understand how to navigate to all the appropriate 'hotkeys' using only keyboard commands. To provide this functionality, Java's Robot class was used to interface the commands with the IDE. This necessitated the invocation of those functions through the buttons and drop-down boxes available to the ordinary user. The current design of the plug-in provides the capability for future developers to add additional functionality to the Eclipse IDE through SpeechClipse's extension points.

Token recognition begins with proper definitions in the speech grammar. At this point, most of the driving force behind the grammar is to try to provide enough functionality to demonstrate the full potential of this plug-in. Other additional work will be to refine and improve this grammar set.

Flexibility has been added through various utilities that handle multiple editors. For instance, 'Notepad', 'MS Word', and other editors can be started by the speech recognition capability of the plug-in.

4. Results

The development of SpeechClipse shows the feasibility to control the Eclipse IDE with voice commands. Additional efforts will improve the range of functionality with the goal of improved usability. Through experimentation and use of SpeechClipse, we have shown that not only is it possible to provide an environment for a 'physically challenged' individual, but that it is part of the natural evolution of programming tools in providing incremental enhancements.

The current version of SpeechClipse can handle various accents. Most of the individuals on this project are not native English speakers, but the recognition process proved to be extremely reliable with a wide range of speakers and dialects. Multi-language support can be provided through extension points for additional plug-ins.

5. Risks and Limitations

A minimum hardware configuration is required for the proper operation of this plug-in. The minimum requirement is a computer with at least a P III, with 800 MHz processor and 512 MB RAM. For best results, a P4 processor with 512 MB RAM is suggested.

The speech engine is sensitive to external noises and extraneous tokens may be recognized incorrectly. Any extraneous noise through the microphone causes invocation of undesirable actions in the IDE. Possible improvements might be achieved through the use of a noise limiting device or improvements in the grammar to help distinguish between valid and invalid commands.

Generally speaking, spoken language has severe limitations when applied to human-computer interactions. Speech is inherently slow and transient, which makes it difficult to present and edit information. However, speech has proved to be useful as alerts in busy areas, for store-and-forward messages, and for enhancing the user interface of environments for the physically impaired. Although the SpeechClipse plug-in is far from providing a comprehensive I/O system for the physically impaired, we believe the system has the potential to improve on what has already been implemented so far.

6. Future Work

The expansion of the capability of the SpeechClipse plug-in tool can be made by improving the grammar and the implementation of potential error recognition and recovery. Currently, the grammar doesn't make use of recursion or weighted-tokens, both of which can make the recognition process faster.

Additional extension points could be added to allow implementations for speech synthesis, and multi-lingual support. SpeechClipse currently supports a basic form of spoken code writing facility called **LazyTyping**. Users can dictate well known programming language keywords, but more work is needed in making LazyTyping flexible. Code template structures ('for' loops, 'switch' statements, etc) and the ability to customize the grammar for user-defined words or phrases are some additional capabilities also under consideration for future implementation.

It will also be necessary to provide easy navigation through ALL system functions, such that any system related activity can by handled without the need for mouse or keyboard control. Examples include access to 'start' button, 'search' function, and other necessary support functions for the general programming environment. Additional features are planned to expand the various navigator views to include other perspectives provided by other plug-ins.

Ambiguity in recognition ("for" or "fore"?) also provides important issues to look into. An example can be for the LazyTyping subsystem; if a user wants the system to type the digit "1", how should the system recognize the spoken input to print "1" instead of "one"? This issue will be explored within the general area of speech enabling technology, and Eclipse provides the perfect platform for conducting such research.

There are still occasions when it would be beneficial to make use of the voice enabling property of SpeechClipse in conjunction with the keyboard and/or mouse functionalities. Schneideman [6] mentions an interesting observation: cognitive resources decrease significantly when a user is restricted to speech only; multi-modal usage often results in an increase in available cognitive resources. Schneiderman reports that humans find it difficult to speak and think at the same time, yet using a keyboard in tandem with the speech system resulted in a 15-30% increase in speed-up. Parallelism of spoken output with keyboard typing needs to be investigated for SpeechClipse and the LazyTyping subsystem.

It is believed that other recognition engines would work equally well, or perhaps even better, than CloudGarden. A planned extension of the work is to support IBM ViaVoice [8] in the near term. It is expected that ViaVoice will be more compatible with the project's long-term goal involving continued research on the Eclipse IDE.

7. Conclusion

This paper describes the implementation and research effort carried out while designing a voice-enabling plug-in for the Eclipse IDE. SpeechClipse adds important functionality to the Eclipse IDE and represents one of the first plug-ins of its kind. The implications of voice-enabling the Eclipse IDE can result in making the programming environment available to a wider demographic as well as providing an alternative to keyboard and mouse input. The paper also describes the observations, risks and limitations in the current implementation of SpeechClipse, along with its future directions.

As an accompaniment to the presentation of SpeechClipse, we are prepared to demo the technology at the workshop.

About the Authors

The team responsible for this research was originally formed as a group within the Object-Oriented Distributed Computing laboratory at the University of Alabama at Birmingham (UAB). The research goal was to add adaptive/assistive technology to the Eclipse IDE through plug-in mechanisms. Faizan Javed, Rajesh Sudarsan, Raymond Corvin, and Suman Roychoudhury are Ph.D. students in computer science at UAB. Qasim Ijaz and Shairaj Shaik are M.S. students in Computer Science at UAB. Jeff Gray is Assistant Professor, and Barrett Bryant is Professor in the Computer Science department at UAB.

References

1. James F. Allen, Donna K. Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, Amanda Stent, "Towards Conversational Human-Computer Interaction," *A.I. Magazine*, Winter 2001, pp. 27-38.
2. Richard Baldwin, "Introduction to the Java Robot Class in Java," http://www.developer.com/java/other/article.php/2212401
3. *Java Speech API*, http://java.sun.com/products/java-media/speech/
4. Microsoft Corporation, "What is Speech Technology?" http://www.microsoft.com/speech/evaluation/techover/
5. Harry Nielsen, "Speech Turns People with Disabilities into Technological Leaders," *Speech Technology Magazine*, May 1997, http://www.speechtechmag.com/pub/2_2/cover/532-1.html
6. Schneiderman B., "The Limits of Speech Recognition", *Communications of the ACM*, September 2000, pp. 63–65.
7. Sherry Shavor, Jim D'Anjou, Dan Kehn, Scott Fairbrother, John Kellerman, Pat McCarthy, *The Java Developer's Guide to Eclipse*, Addison-Wesley, 2003.
8. Satish Swaroop, "Do you hear what I hear?" *IBM developerWorks*, November 2001, http://www-106.ibm.com/developerworks/ibm/library/i-voice