HotOS-VII, March 29-30, 1999,Rico Rio, Arizona
Version 2.0 March 1, 1999
Printed 03/11/99 2:36 PM

Command Management System for Next-Generation User Input

Mitchell Tsai, Peter Reiher, and Gerald J. Popek♦
Computer Science Department, University of California, Los Angeles
{tsai, reiher, popek}@cs.ucla.edu

Abstract

Next-generation user input and sensor technologies (such as speech, handwriting, vision, and physical location) present new possibilities for adaptable easy-to-use commands. These input modalities also present problems of noise and ambiguity, for which current "ad hoc" command processing services (e.g., undo) are often inconsistent and insufficient. Applications should work with sensor processors to interpret context-sensitive commands and to prevent irreversible errors. Moreover, sensor processor performance can increase with preknowledge about expected commands, from application designers or from real-time analysis of user behavior. These observations suggest that operating system services for analyzing and refining user commands may be important for enabling robust, versatile human-computer dialogue systems. We propose a context-sensitive command management system as a cornerstone for next-generation user input services.

1. Introduction

Improvements in computer performance have been limited traditionally to the speeds and capacities of processing, memory, or network bandwidth. However, a fast, robust input system would greatly benefit palmtops and car computers.

One reason for the success of the 3Com Palm Pilot™ is that most commands bypass the error-prone graffiti process, relying on special buttons and tablet-like menu pointing. When using graffiti shortcuts for commands, 30 minutes of careful input can be easily lost when "cut" is misinterpreted as "paste", since very few Palm Pilot™ operations are undoable. Irreversible errors painfully demonstrate the need for robust command management services.

Researchers have been driving for basic speed and accuracy in speech and handwriting analysis, but they often neglect two important factors - the ease of error correction and the cost of proofreading. Consider a dictation system with 95% accuracy at 120 wpm. If each error takes 10 sec to correct, approximately 50% of our time is spent correcting errors. Moreover, proofreading time goes up dramatically as recognition accuracy decreases.

If the entire speech recognition process is too slow, then writing on a Post-It Note™ and speaking into a credit-card-sized voice recorder will be faster and more convenient. We suggest that the processing of user commands is not merely a user interface issue, but an integral problem of system support for next-generation user inputs. Leveraging new communication modes effectively involves robust support for easy-to-use and adaptable commands, not just fast recognition and passable accuracy.

In this paper, we present our approach to a command management system - a comprehensive system for analyzing and refining user commands.

1) Application commands can be exported to the operating system, where they can benefit from a richer family of system support services.
2) For tighter integration with sensor processors, applications can export command details such as safety, reversibility, expected usage patterns, cost, and execution time.
3) Analyzing behavior patterns in command sequences to identify "context" can be a key method for handling noise and ambiguity, leading in the future to more sophisticated context-sensitive conversational analysis.

2. New Sensors with Today's Systems

Consider how a typical application might implement speech recognition using a commercial speech engine (See Figure 1).

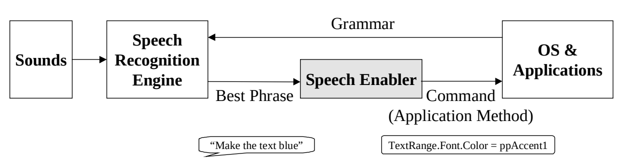

Figure 1: Speech-enabled application

First, the speech engine is preloaded with a grammar containing all allowable phrases. The speech recognition engine (F) processes each incoming sound to return the

♦Gerald Popek is also associated with PLATINUM technology, inc.

most likely phrase, and a Speech Enabler (G) maps phrases to application commands:

Phrase = F(Sound)
Command = G(Phrase, Context)
= G(F(Sound), Context)

Context is necessary because an application may use different "methods" for the same command, depending on whether it is in Editing mode or SlideShow mode.

Using Context-Free Grammars (CFGs), this design may achieve accuracies of 80-99% depending on the size and nature of the grammar. One reason for a more advanced command handling system is to assist speech engines in improving the overall recognition accuracy.

2.1. Experience with Difficulties

However, superimposing sensor processors on GUI applications presents many difficulties beyond this basic description. The ones we describe are the result of:

1) Our efforts to speech-enable PowerPoint, a common slide presentation application
2) Our work with DragonDictate, a speech recognition program for command and control of Microsoft Windows applications
3) Experiences of full-time users of speech-enabled word processors and operating systems, such as IBM ViaVoice, DragonDictate, and Dragon NaturallySpeaking
4) Experiences of blind users of speech-output (keyboard input) systems, such as JAWS for Windows
5) Handwriting and graffiti systems on palmtop computers, such as the 3Com Palm Pilot™

2.2. Major Issues

To move beyond current systems, there are certain key issues which must be handled.

For commands similar to today's GUI commands:

- What if a sensor processor does not choose the desired command? How can we help it make the correct choice?
- If a sensor processor delivers an incorrect command to an application, can the command be reversed (undo) or trapped before execution?
- What happens if the computer misinterprets critical commands, especially corrections and confirmations?
- If error rates are very high, does this change the nature of the command process?

For commands similar to natural human communication:

- How should the computer process deliberately (or accidentally) ambiguous user inputs?
- How should inputs from multiple sources be synchronized and integrated into commands?
- For context-sensitive and dialogue-sensitive commands, how should the computer store information about context and the ongoing dialogue with the user?

In real systems, these issues are quite intertwined, which is one reason that application-specific solutions may be insufficient. *Ad hoc* solutions can also be inconsistent and confusing to real-world users. Next, we present two major issues which illustrate the impact of these issues.

2.2.1. History – Linear Undo Stacks vs. Context

A common problem for speech systems is hearing and executing a series of commands before the user realizes that the first command has been misinterpreted. Consequently, many heavy speech users try to choose applications that support multiple undos. Surprisingly, most applications don't support undos except when editing text or other content, and more than one level of undo is uncommon.

Even multiple undos are not foolproof. A serious complaint among users of speech-enabled word processors is the unexpected "Exit without Save" phenomenon. Two explanations have come to light:

1) After "Exit" is accidentally selected, the word "Do you want to save the changes you made to 'Document X'" dialog presents a "Yes, No, Cancel" dialog box. Then "No" is also accidentally selected.
2) When dictating text, somehow the menu bars are activated (e.g., using control keys for menu items), and a series of somewhat random commands causes an application error, closing the application. Replicated by extremely fast typists, this is not a problem exclusive to speech software.

The undo stack depth varies widely between applications, and the stack may unexpectedly clear. Commands are aggregated in the stack in an *ad hoc* fashion, and some commands never appear. The way that text undos and redos are implemented varies between applications, requiring custom speech interfaces.

Some examples of limited and inconsistent operation:

1) Word Processors – If you undo some operations and accidentally type the letter "a", all of the other operations are lost. This is a major design flaw of all linear undo stacks.
2) Microsoft Outlook – It only offers one level of undo for e-mail, and offers very few undos elsewhere.
3) Netscape Mail – It offers one level of undo; however, multiple text entries separated by mouse clicks are treated as one entry.
4) Quicken – If you enter a transaction and accidentally hit the "Cancel" key, it is permanently lost.
5) Microsoft Word – "Outline View" is not undoable, which confuses novices when accidentally activated.

6) Microsoft PowerPoint and Word have different undo philosophies. In Word, “display changes” are ignored on the undo stack, so backtracking through changes keeps you in the same view. In PowerPoint, as you backtrack, the view changes to match the original view where you performed your editing.

Possible supplements to undo stacks are Unix™ command shell histories and newer Web browser histories, where you can access a web site no longer on the forward-back stack. At least, we need multiple levels of undo standardized in some fashion (content changes, view changes, application settings) that are not vulnerable to the play/replay problems of a linear stack.

We believe that command processing would be better served by a rich contextual command storage structure, similar to what humans use in conversation. It should be accessible through various indexes and context-sensitive searches.

2.2.2. Context-Dependent Commands

This discussion of undo stacks has assumed that commands are as simple as “Exit” and “Save”, but commands may require historical knowledge about the context of other commands – “Shrink these boxes like the circles [that were shrunk 10 commands ago].”

Commands can depend on many types of context:

- Internal (operating system and application info)
- Interface (dialogue history, input device history)
- External (physical and social environment)

For example, consider these car commands:

- Localized Behavior: Reissuing a command such as “Where is the nearest entrance to Highway 405?” is time-dependent and location-dependent.
- Appropriate Behavior: “Read my schedule” should not read confidential files when passengers are present.

Proper command handling can also involve context-dependency in the processing of commands. Pointing may occur using a 2-button mouse, tiny keyboard J-stick, laser pointer, speech, or a visually-processed gesture. Each input mode has unique problems - biases, flaky output patterns, noise levels – which may produce different behavior patterns and user confirmation preferences.

When a sensor processor receives a noisy or uncertain command, it may wish to use contextual information to determine whether to ask a user for clarification. Does the user ever use this command? Is this command dangerous? Is it reversible? How much does it cost?

Safety may be compromised since this is separate from application-centered methods for confirming actions, such as the “yes/no/cancel” dialog box. Applications do not expect flaky responses to confirmation dialog boxes.

These context-dependency issues may not be efficiently resolved by a distinct user interface layer, requiring deeper interactions between sensor processors, applications, and the operating system.

3. Command Management

We postulate that it is time to rethink the interactions between commands, macro recorders, help systems, wizards, undo mechanisms, history files, and other related services. Our approach to a command management system includes the following components.

1) System support services for recording and editing application commands.
2) For tighter integration with sensor processors, applications export command details such as safety, reversibility, cost, and execution time.
3) User behavior analysis of patterns in command sequences to identify “context”, as a method for handling noise and ambiguity.

3.1. Recording and Editing System

In order to handle higher error rates, there must be some standardized system for recording, replaying, editing, and refining commands. When errors and ambiguity are common, the editing system is often the deciding factor for real-world users.

One topic we are not investigating involves the redesign of applications in a transactional model. Database-like features such as rollback and two-phase commit would be very useful in high error situations.

To work with current applications, our approach is to filter application input as much as possible with a command management system. When this fails, the undo stack may be non-existent or unusable, so undos are sometimes implemented by issuing a different command to reverse the effect, which is not always possible. The interactions between sensor editing and application undos can get quite intricate and unpredictable, which is one reason we suggest tighter integration is a necessity.

3.2. Tighter Integration of Sensor Processors and Applications

Application designers may know important details about application tasks, safety and reversibility of various commands, and common behavior patterns. Combining these details with sensor processor analysis provides more information to correctly determine the intended command (and to determine when to ask a user for clarification).

Frank Linton performed an 18 month study of 16 people using Microsoft Word at MITRE which showed that the average person (technical MITRE user) used only 25 distinct commands/month out of 642 available commands [4]. For speech systems, each command has multiple possible phrases, and items in commands (e.g. “box”, “line”) must also be included. Personal experience with an airline reservation system has shown that 3,000 and 60,000 word vocabularies can have a 2-4X difference in error rate (e.g., 5% versus 15%).

3.3. User Behavior Analysis and Contexts

Analyzing behavior patterns in command sequences (including timing) may provide additional information to correctly determine the intended command. These methods can come from statistics, artificial intelligence, applied linguistics, and other fields. Some examples are:

- HMM (Hidden Markov Model) and Bayesian Nets
- AI Scripts (Partially-ordered sequences of tasks)
- Dialogue analysis for tracking conversations

We suggest that contexts can be based on user behavior analysis in addition to conventional definitions of context (a list of conditions):

P(Command | Context, User Behavior Context)

3.4. Command Management Taxonomy

Many tasks handled today using *ad hoc* techniques could benefit from knowledge about user commands. Our taxonomy includes five major categories:

Identification and Access

Identification: Separate commands from noise
Creation: Mapping sensor data to commands
Access: Commands accessible in current context

Automatic Filters

Safety. Security.
Extension: Browsers which fill in "http://"

Multimodal Coordination

Integration. Synchronization.

User Involvement

Lookup: Available commands. Menu browsing.
Verification. **Editing.**

Automatic Assistance

Wizards. **Macros.**

4. Design of a System

At UCLA, we are developing a Command Management System to support some of these command management functions. Current prototypes run on a 166 MHz Pentium with Windows NT 4.0, and handle various combinations of speech, keyboard, and mouse.

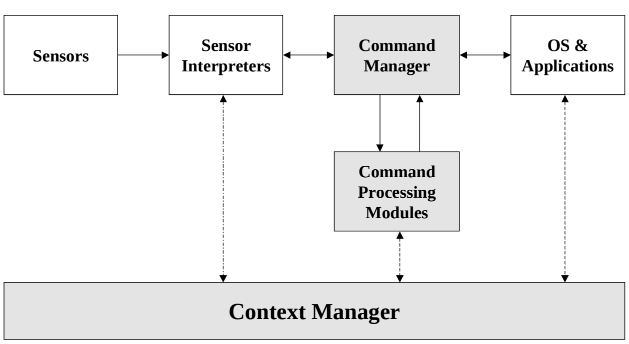

Figure 2: Command Management System

4.1. Context Manager

The context manager records all user commands and performs analysis of patterns in user behavior (real-time and off-line). It provides a central repository for storage of command properties, such as reversibility and cost.

4.2. Command Manager

The command manager intercepts communication between sensor processors and applications. It uses knowledge from the context manager to provide help the sensor processors select better commands and filter commands before they reach the application. These functions are often provided by sending information to the Command Processing Modules. A "Top 10" module and a safety module are two examples.

It also interacts with the user for clarification.

4.3. Command Processing Modules

These modules process commands from the command manager, taking uncertain data, and returning it in a processed or higher-level form. Command Management Tasks such as wizards and multimodal integrators function in this area. In our system, we are currently focusing on context-sensitive use of "Top 10" speech output, context-sensitive mouse movements, safety filters using reversibility information, and multimodal integration of speech, mouse, and keyboard inputs.

4.4. Multi-Level Events with Probabilistic Data

To process ambiguous data, communication between modules uses "command objects" in events. This is an example of one "command object":

Type = Speech
$P_{Clarification} = 0.6$
$N_{Commands} = 3$
Command[1] = "Thicken line 11", L[1] = 0.61
Command[2] = "Thicken line 13", L[2] = 0.24
Command[3] = "Quit", L[3] = 0.15

Our system uses distributed multi-level events, in addition to traditional centralized handling of context information. Each data stream consists of low-level events such as mouse clicks or speech fragments. Interpreters and analyzers produce higher-level events like words or commands. These higher-level events can be cues or behavior patterns, such as "Microsoft Word user in reading mode or article-writing mode."

4.5. "Top 10" Lists & User Behavior Patterns

What if the recognized phrase is not the one desired? Speech recognition engines can return a "top 10" list of phrases with recognition scores.

Score (Phrase | Sound) = –100 to 100

If we have some knowledge about frequencies of command usage (through application designer pre-analysis and/or real-time user monitoring),

P(Command | Context) = 0 to 1
we can combine these data to select a more likely phrase.

We combine this information using likelihoods(L). Note that data for L(Phrase | Sound, Context) is unavailable, so we estimate with L(Phrase | Sound).

L(Command | Sound, Context) =
L(Command | Context)
* L(Command | Phrase, Context)
* L(Phrase | Sound)
where $L(A) = F(A) / (\Sigma_A F(A) - F(A))$
and F(A) can be P(A) or some other scoring function.

4.6. Other Methods for Behavior Patterns

We are investigating the use of different methods for detecting user behavior, such as:

a) Pre-defined contexts as sets of probabilities, with simple heuristics for switching contexts
b) Variations on Least Recently Used (LRU) for defining contexts
c) Simple Markov Models updated through hand-coded rules
d) Hidden Markov Models (HMMs), although these do not work for modeling multiple modalities in an integrated fashion
e) Bayesian Networks, more flexible, but require lots of training data

This appears to be a very rich area for statistical analysis and heavier-weight machine learning algorithms.

4.7. Performance Evaluation

A common research approach to evaluating speech systems measures the error rate of the best choice phrase (e.g., $Error_{phrase}$ = 5% or 15%). Real systems must consider two additional factors, which often dominate the total time to complete a task: (1) $T_{corrections}$, Time required to correct mistakes and undo operations, (2) T_{check}, Time required to check for errors.

$T_{operation} = T_{actions} + T_{corrections} + T_{check}$

To our knowledge, previous analyses have never considered the effect of $T_{check.}$ Differences between two types of errors, $Error_{minor}$ and $Error_{serious,}$ have also been ignored in performance measurements.

4.8. User Clarification and Confirmation

If we define Cost and reversibility ($L_{Reversible}$) for each command, we can produce the two outputs for a user clarification system - $Command_{ML}$ (the most likely command) and $P_{Clarification}$ (the probability that we should clarify the command with the user):

$P_{Clarification} =$ [1-L($Command_{ML}$, Context)]
* $L_{Reversible}$($Command_{ML}$, Context)
* L_{Cost}($Command_{ML}$, Context)
where $L_{Reversible}$ = 0 to 1 (1 means fully reversible)

Real formulas are more complicated and involve the top most likely commands. In the following case, we should ask the user “Command A or Command B?”

Score(Command A | Sound) =75
Score(Command B | Sound) =70
Score(Command C | Sound) =30

These efforts try to reduce $Error_{serious}$ by providing better clarification strategies, but there is a tradeoff between $T_{actions}$ and $Error_{serious}$, as $T_{actions}$ will increase if our efforts slow the user down unnecessarily. The design of effective rules and representations for combining command properties is a research topic we are exploring.

5. Applications and Benefits

5.1. Multimodal PowerPoint

We are demonstrating the Command Management System by adding multimodal functionality to Microsoft PowerPoint. Simply speech-enabling the program with existing commands provides few benefits. We are exploring the advantages and disadvantages of context-sensitive speech and mouse commands.

The support of the command management system enables easier implementation of context-sensitive commands (e.g., “Make these arrows thicker. Oops. Except this one.”) and allows new functionality (e.g., editing in SlideShow mode, even during a presentation). Obviously, any improvements we make could be implemented in a custom application, which is how current speech-enabled word processors are implemented.

Currently, researchers are uncertain how contextual processing should be partitioned between applications and independent speech understanding programs. The nature of the communication interface is also a wide open area for research. PowerPoint allows us a concrete real-world environment to study the relative advantages of different command management interfaces.

Prototype systems demonstrate that the amount of time and number of commands necessary for some actions can be greatly reduced. Some speech and mouse commands can require 20-50 equivalent keyboard and mouse actions. For example, “Make this [Box 11] the same shape [as previous Box 12].” exists as a PowerPoint method, but it is unavailable in the GUI interface.

One key advantage of speech is that a speech command does not require the user to move the mouse, unlike toolbars and menus. Keyboard control-keys can serve the same purpose (if we can remember them).

We are using speech and mouse-movement to select objects from among nested groups; i.e. “Make the big boxes blue.” while ambiguously pointing to boxes. Contextual information helps decide which boxes should be selected, using physical distance and command history.

5.2. Speech Command and Control

In Dragon Dictate, we plan to use the Command Management System to increase the safety of issuing commands to multiple applications. We can provide some automatic support for the management of vocabularies and topics among applications.

5.3. Adaptive Computing

Command management services can be a key information source for unified resource management. Adaptive performance can improve latency and resource usage through preknowledge and anticipation of user commands. Pre-fetching of resources can potentially produce order of magnitude differences in response time.

6. Related Work

This work builds on a wide variety of research, both in computer science and in other fields, such as conversation analysis in applied linguistics. We summarize some of the most directly related work.

- *Context-Handling Infrastructures*

Cross-application context-handling infrastructures have just recently been developed. At Georgia Tech, Daniel Salber and Gregory Abowd have recently designed a centralized context server for developing context-aware applications [5].

SitComp (Situated Computing) provides situation information and context data for mobile ParcTab devices and other computers [2]. Schilit wrote his thesis at Columbia University on an architecture for context-aware mobile computing [6].

Their focus is mainly using simple external sensors such as GPS receivers and user identification devices to provide background information to assist commands.

- *Multimodal Architectures*

At the Oregon Graduate Institute, they have developed a distributed, agent-based, multimodal system called QuickSet [1]. It uses typed feature structures to store multimodal inputs in a common meaning representation, and uses temporally sensitive unification for multimodal integration.

At CMU, Vo and Waibel have developed a set of grammar-based Java tools for constructing multimodal input processing modules [7].

Both of these systems provide good methods for information storage and integration; however, the main emphasis is on improving interpretation accuracy as much as possible. Little attention is given to error-handling, safety, or editing of previous commands.

- *Context for Adaptive Operating Systems*

With the advent of faster processors and large amounts of memory, context is being used by many groups to adaptively improve performance, mostly in recompilation and dynamic feedback techniques.

At UC Santa Cruz, Kroeger [3] used multi-order context modeling to predict file system actions from prior events, achieving 15% more cache hits than LRU.

7. Conclusion

We believe that user input services are a neglected area crucial to the real world success of computers using next-generation input sensors. The application-centric management of user commands in current GUI systems is insufficient for the problems presented in these new environments – noise, ambiguity, errors, multiple fragmented input sources.

Moreover, leveraging new input sources for more powerful, natural human-style commands will require general system support for user behavior analysis and for creating and managing contextual information at multiple levels. What changes in application and operating system design will be necessary for sensor analysis and interaction? We believe that a key foundation for versatile dialogue management is an operating system service for managing and analyzing user commands.

Hopefully, our Command Management System will provide a framework for the development and evaluation of new learning, adaptation, and support methods for human-machine communication.

8. Acknowledgements

Xuedong Huang (Speech Technology Group) and Bruno Alabiso (Language Enabled APplications) at Microsoft Research for valuable experience working with the Microsoft Speech SDK, LEAP, and NLP systems during a research internship. Richard Guy, Andy Wang, and Mark Yarvis for proofreading and suggestions.

9. References

[1] Cohen, P., Smith, I., Chen, L., Clow, J. “QuickSet: Multimodal Interaction for Distributed Applications,” Proceedings of the Fifth Annual International Multimodal Conference, 1997.

[2] Hull, R., Neaves, P., Bedrod-Roberts J. “Towards Situated Computing,” Proceedings of ISWC ’97, First Intl. Symposium on Wearable Computers, Cambridge, MA, 1997.

[3] Kroeger, T., Long, D. “Predicting File System Actions from Prior Events,” 1996 USENIX Technical Conference, 1996.

[4] Linton, F. “Microsoft Word's Usage Patterns: A Naturalistic Inquiry,” Presentation at Microsoft Research from MITRE Corporation, May 18, 1998.

[5] Salber, D., Abowd, G. “The Design and Use of a Generic Context Server,” Technical Report GIT-GVU-98-32, Georgia Institute of Technology, 1998.

[6] Schilit, W., System Architecture for Context-Aware Mobile Computing, Columbia University Ph.D. Thesis, 1995.

[7] Vo, M., Waibel, A. “Modeling and Interpreting Multimodal Inputs: A Semantic Integration Approach,” Technical Report CMU-CS-97-192, Carnegie Mellon University, December 1997.