Operating System Services for Managing Context

Mitchell Tsai, Peter Reiher, Jerry Popek

Computer Science Department, University of California, Los Angeles
Los Angeles, CA 90095
{tsai, reiher, popek}@cs.ucla.edu

Abstract

Simple approaches to context can improve performance of existing applications and operating systems, especially when providing lightweight interpreters for user inputs such as speech and vision. A central service for internal computer contexts and external user contexts can also provide a means for improved resource management. User behavior analysis and *command properties* (such as safety, reversibility, and cost) can produce sophisticated contexts which can enable safer and more accurate interpretation of human-computer communication. Our "*Baby Steps*" system is a set of operating system services and a framework for providing basic dialogue management. We hope that it will provide a foundation for interfacing more versatile dialogue and resource management systems with conventional applications and operating systems.

Introduction

With the advent of faster processors, larger memory chips, and more sophisticated user input sensors, we may wish to reexamine the simple planning and allocation strategies used in operating systems today. CPUs, disk drives, memory, and other resources are managed individually by lightweight algorithms. Keyboard and mouse commands are very simple context-independent actions, easily interpreted with low overhead requirements.

Our idea is to use contexts and dialogue management in our *Baby Steps* system to bring some artificial intelligence ideas and techniques into general-purpose operating systems.

In general, we have the following goals:

1) Improve planning and use of individual data streams (i.e. CPU, disk drive, network)
2) Use contextual knowledge of other data streams when managing one data stream.
3) Improve basic performance of next-generation user inputs (i.e. speech, voice, position)
4) Enable sophisticated, natural use of next-generation user inputs.

Newer user inputs - such as speech, gesture, handwriting, vision, and position sensors - have many characteristics that make older systems inefficient and inadequate.

1) Noise (much higher levels and many types)
2) Errors (by the person and the computer)
3) Ambiguity (accidental and deliberate)
4) Fragmentation (of input across different sensors)

They may require more information about their surroundings for proper processing and interpretation. This helpful information or "context" is distributed both temporally (in conversation history) and spatially (in different data streams and devices) in the following types of data streams:

1) Internal (application, operating system, network)
2) Interface (conversational, user interface)
3) External (psychological, environmental, social)

Good design of dialogue systems is vital for computer applications that may perform dangerous and/or irreversible actions. Irretrievable error loss forces users to discard many programs as too dangerous for their needs. This is a very common problem with today's speech recognition users, who abandon programs without adequate undo support. Error correction methods are a critical aspect of dialogue systems. Dragon NaturallySpeaking is often preferred over IBM ViaVoice because ViaVoice does not allow correction using speech-only communication.

We believe that the active use of contexts and *command management* (Tsai *et al*. 1999), a subset of dialogue management that handles basic command services using *command properties*, may provide important safeguards when using natural conversation for real-world tasks.

How should "context" influence computer processing? Should we mimic human cognitive properties of memory, reasoning, and decision-making such as priming effects, framing effects, forgetting curves, activation, and reinforcement? Which of these features will improve computer processing and human-computer communication in the short-term?

To investigate some of these ideas, we are developing *Baby Steps*, a dialogue management system which:

1) Is a general system for interfacing and handling multiple representations of context
2) Allows us to test simple representations of context for improving some tasks
3) Serves as a testbed for lightweight learning, adaptation, and support methods

4) Uses real-world conventional applications to test ideas, such as context-sensitive speech and mouse commands.

In *Baby Steps*, the current situation is defined as an overlay of many *contexts*, which may contain widely varying types of information. Some sample *contexts* are the following:

Context A: [PowerPoint user giving real presentation]
Context B: [PowerPoint user in editing mode]
Context M: [User is in a rush, deadline approaching]
Context N: [User is relaxed]
Context Y: [User prefers video output]

Some *contexts* are produced by analyzing behavior patterns or explicitly asking the user. Other *contexts* are produced from external sensor data and internal computer states. For example, the current situation may be described as:

Current Situation = 80% Context A + 20% Context B

Why are contexts weighted, rather than simply placed on a stack? For speech recognition, a very important component of *contexts* is a list of vocabulary phrases or a grammar. Speech recognition engines must be preloaded with grammars containing acceptable phrases, and their performance improves with more specific vocabularies.

Frank Linton (1998) performed an 18 month study of 16 technical users of Microsoft Word at MITRE which showed that they only used 25 distinct commands per month, out of the 642 available commands.

Overlapping *contexts* allow us to define different partial grammars containing probabilities and likelihoods in a space-efficient manner.

To handle the effects of dialogue-induced actions, many *contexts* contain *command properties*, characteristics as safety, reversibility, expected usage patterns, cost, execution time, and reversal command – the specific command needed to reverse or undo a command. These *command properties* are used in interpreting ambiguous speech and mouse movements, deciding when and how to clarify commands with the user. "Did you mean Command A or Command B?"

One application area for *Baby Steps* is Microsoft PowerPoint, where we are supporting new context-sensitive speech and mouse commands in editing and slideshow modes. Examples include "Make these boxes smaller [fuzzy mouse movement around some boxes]", "Move to the slide with the graph", "Delete this arrow", "Make this box the same shape as the Sensors box". Contextual information allows us to resolve the ambiguities and to combine context-sensitive mouse and speech input.

We are using the PowerPoint environment to explore issues of interaction in real and virtual environments. Consider this hypothetical conversation between people and an intelligent house.

Susan: "Please turn on this light."

Susan is looking towards two lamps in the living room.
Susan is not pointing to anything.

House: "Should I turn on the small lamp?"
Susan: "No, the tall one."

Susan moves from living room into kitchen.
Susan opens refrigerator and starts preparing some food.

Mark: "Is Susan still reading?"
House: "No, she's cooking in the kitchen."
Mark: "Can you play Ravel's Bolero for her?"
House: "Ok"
Mark: "Susan, would you like some help?"

House directs message to Susan through kitchen speaker.

Proper management of contextual information is crucial towards resolving these ambiguities and multiple input sources. In addition to the semantic issues that are present in this perfect dialogue, contextual information is crucial for handling noise and errors in real dialogues. Future dialogue systems must be safe and effective, or most people will stick with simpler reliable mechanisms.

Architecture of a Dialogue Manager

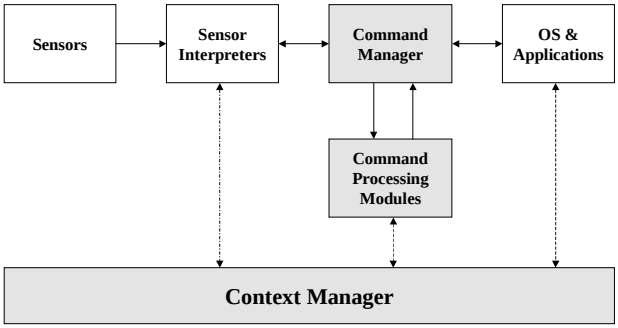

Figure 1: *Baby Steps* Dialogue Management System

Baby Steps is a Dialogue Management System (Figure 1) which adds three main components to operating systems:

The Command Manager intercepts communication between sensor processors and applications. It helps the sensor interpreters select better commands and filter commands before they reach the application.

The Command Processing Modules process commands from the command manager, taking uncertain data and returning it in a processed or higher-level form. The current modules focus on context-sensitive use of "Top 10" speech engine output, context-sensitive mouse movements, safety filters, and multimodal integration.

The Context Manager monitors data streams in the background, analyzing patterns in user behavior to detect when the situation or context has changed. It also provides a central repository for storage of *command properties*, such as reversibility and cost, which help the dialogue system determine when and how to take action.

Current *Baby Steps* prototypes run on 166-333 MHz Pentiums under Windows NT 4.0. They handle speech, keyboard, and mouse communication.

Simple Approaches to Context

Consider a naïve implementation of speech recognition using a commercial speech engine (See Figure 2).

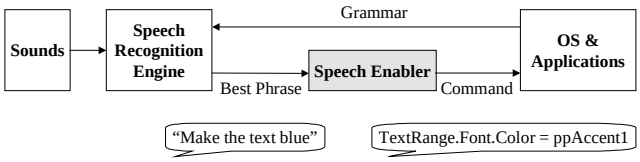

Figure 2: Speech-enabled application

The speech engine is first preloaded with a simple grammar containing all allowable phrases. Then the speech recognition engine (F) processes each incoming sound to return the most likely phrase, and a Speech Enabler (G) maps phrases to application commands:

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

Contexts merely record the application state. For example, Microsoft PowerPoint uses different "methods" for the same command in Editing mode and SlideShow mode.

Even this bare-bones design presents important issues:

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

Improving Speech Recognition Performance

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

Relative frequencies of commands may be available:

P(Command | Context) = 0 to 1

We can combine these data to select a more likely phrase using likelihoods (L).

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.

"Baby Steps" Context Design

Baby Steps contexts contain three main sections:

a) Definition
b) Usage : properties, reasoners
How do we use this *context* to conduct dialogue?
c) Meta-Usage : activation/deactivation, meta-reasoners
When do we activate this *context*?

The Usage section of the following simplified *context* includes a grammar containing command likelihoods. The Meta-Usage section contains some rules about activating and deactivating this *context*.

Context (ID = 1)
Name = "Viewing Slides"
(*optional* manually-created name)
Number of Commands = 9
Grammar = "Open", "Close", "Quit",
"Next|Previous -|Slide", "View|End Slide Show"
Properties Matrix = "Open" L=0.1
"Close" L=0.1
"Quit" L=0.05
"Next Slide" | "Next" L=0.3
"Previous Slide" | "Previous" L=0.3
"View Slide Show" L=0.1
"End Slide Show" L=0.1
Meta-Usage =
Activate when "View Slide Show" is heard
Deactivate when an editing command is heard

By switching between this *context* and other *contexts* (such as an "Editing Slides" *context*), we can focus the recognition of commands towards expected phrases, as well as use different initiative models to decide which commands to clarify and confirm.

Active Situation Defined by Overlaying Contexts

The active situation is defined by blending various *contexts*. We are mostly concerned with the creation of appropriate vocabularies and grammars for speech recognition engines. But we are also interested in combining knowledge of possible behavior patterns. How does user behavior change as they shift from "Viewing Slides" to "Proof-Reading Slides" to "Editing Slides", and how do we capture this in our *contexts*?

Current Situation = 40% Context C + 30% Context D
+ 10% Context G + 10% Context H
+ 5% Context X + 5% Context Y
Current State:
1) Active contexts: Context C, Context D
2) Suspended contexts: Context X, Context Y
3) Pre-active contexts: Context G, Context H

Pre-active *contexts* are ones that we predict may occur in the near future. For example, if the computer hears one editing command, that may be an accident. It may not be useful to activate all the editing commands immediately, since this may harm speech engine performance. The actual context switch also adds an additional cost.

Knowledge About Actions

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).

Contexts may include many of these *Command Properties*:

"Move box"	L=0.45, Reversible=1, Safety=1, Cost=0.1, UndoCommand ="Move box"
"Delete box"	L=0.2 , Reversible=1, Safety=0.5, Cost=0.2, UndoCommand ="Add box"
"Quit"	L=0.1 , Reversible=0, Safety=0, Cost=0.8, UndoCommand =null

Deciding When To Clarify and Confirm

Dangerous commands must be clarified, but too many requests for clarification will make the system slower and very annoying. If we define cost (L_{Cost}) and reversibility ($L_{Reversible}$) for each command, we can produce the a probability that we should clarify the command with the user, $P_{Clarification}$:

$$P_{Clarification} = [1 - L(Command_{ML}, Context)] * L_{Reversible}(Command_{ML}, Context) * L_{Cost}(Command_{ML}, Context)$$

$Command_{ML}$ = the most likely command
$L_{Reversible}$ = 0 to 1 (1 means fully reversible)
L_{Cost} = 0 to 1 (a normalized version of cost)

Deciding What To Ask When Clarifying

Pattern analysis of the top most likely commands can produce more useful questions. In the following case, the top two commands have very close scores, so we should ask the user "Did you want Command A or Command B?"

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

If Command A is "Move the red box here." and Command B is "Move the box here", we may decide they are identical commands, so there is no need to ask the user. If only one command has a high score, we can ask about that command.

There are many other useful patterns which can be handled by a dialog manager. For instance, if the user is repeating some previous commands, we might ask "are you trying to do Command X again?" Rather than ask "Which box do you want to move?", we could say "I'm still not sure which box you want." or "Which box?"

Multi-Level Events with Probabilistic Data

Baby Steps uses events to communicate ambiguous information between different parts of the system. These events contain "command objects" such as the following:

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."

If sensor processors, applications, and other dialogue programs can trade ambiguous data that contains dialogue and context information, this can produce better performance than systems that force input events into deterministic states.

Performance Evaluation

Current metrics for measuring performance of dialogue systems are inadequate. Real users are often concerned with "Did it speed up my task?" or "Was it easier to use?", rather than "How many errors occurred?"

A common approach to evaluating speech dictation systems measures the error rate of the best choice phrase (e.g., Error = 5% or 15%). Real systems must consider many additional factors, which often dominate the total time to complete a task (See Figure 3):

1) T_{speech} Time of actual speech
2) T_{delay} Delay before all words appear
3) $T_{corrections}$ Time to correct mistakes
4) T_{check} Time to check for errors

$$T_{total} = T_{speech}\ T_{delay} + T_{corrections} + T_{check}$$

Type	Time (sec)	Speed (wpm)	% Total Time
T_{speech}	38	**160**	16%
T_{delay}	33	85	14%
$T_{corrections}$	131	30	57%
T_{check}	29	26	13%
T_{total}	230	**26**	100%

Figure 3: Speech Dictation Performance

Error Types & Minimizing Impact of Errors

Measuring application performance is more difficult, since errors may be dangerous and irreversible. Dictation programs can provide better performance by changing words after hearing more speech, but current applications are not usually so fortunate. We provide a simple hierarchy of error types (from best to worst):

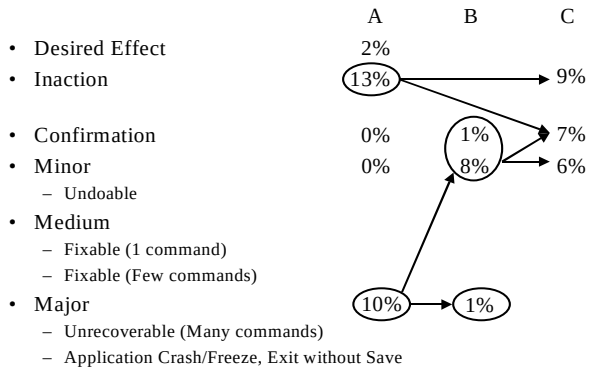

Figure 4: Speech Application Performance

In Figure 4, compare three different versions of *Baby Steps* controlling Microsoft PowerPoint, each with an overall 25% error rate. Most errors in the naive speech-enabled PowerPoint (version A) are not undoable because the application treats all PowerPoint method calls as one unit when issuing "Undo." A fixable command is one that can be manually reversed with a different command (or set of commands).

Related Work

Context-Handling Infrastructures

Cross-application context-handling infrastructures have just recently been appearing. At Georgia Tech, Daniel Salber and Gregory Abowd are working on a Java-based Context Toolkit (1999) to filter noisy sensor and actuator data for applications.

SitComp (Situated Computing) provides situation information and context data for mobile ParcTab devices and other computers (Hull, 1997). Schilit (1995) wrote his thesis at Columbia University on an architecture for context-aware mobile computing.

The systems focus on simple external sensors such as GPS receivers and user identification devices to provide background information.

Multimodal Architectures

At the Oregon Graduate Institute, they have developed a distributed, agent-based, multimodal system called QuickSet (Cohen *et al*. 1997). 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 (1997) have developed a set of grammar-based Java tools for constructing multimodal input processing modules.

Context for Adaptive Operating Systems

At UC Santa Cruz, Kroeger used multi-order context modeling to predict file system actions from prior events. A simple last successor model can correctly predict the next file access 72% of the time (1999).

Conclusion

Simple algorithms in current GUI systems are insufficient for many problems presented by new input sensors – noise, ambiguity, errors, multiple fragmented input sources. 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.

We believe that an operating system service for creating and managing contexts is a more versatile way to deal with dialogue management. "Baby Steps" is intended to provide a framework for the development and evaluation of new learning, adaptation, and support methods for human-machine communication.

References

Abella, A., Brown M. K., Buntschuh, B. M. 1996. Development Principles for Dialog-based Interfaces. In *Proceedings of the ECAI-96 Workshop on Dialog Processing in Spoken Language Systems*

Cohen, P., Smith, I., Chen, L., and Clow, J. 1997. QuickSet: Multimodal Interaction for Distributed Applications. In *Proceedings of the Fifth Annual International Multimodal Conference*.

Donaldson, T., Cohen, R. 1997. A Constraint Satisfaction Framework for Managing Mixed-Initiative Discourse. In *AAAI-97 Symposium on Mixed-Initiative Interaction.*

Hull, R., Neaves, P., Bedrod-Roberts J. 1997 "Towards Situated Computing. In *Proceedings of ISWC '97, First Intl. Symposium on Wearable Computers*, Cambridge, MA.

Kroeger, T., Long, D. 1999. The Case for Adaptive File Access Pattern Matching. In *Proceedings of the Seventh Workshop in Hot Topics in Operating Systems (HotOS-VII).*

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

Salber, D., Abowd, G. 1999. The Context Toolkit : Aiding the Development of Context-Enabled Applications. In Proceedings of *Conference on Human Factors in Computing Systems CHI-99*. Forthcoming.

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

Tsai, M., Reiher, P., Popek, G. 1999. Command Management System for Next-Generation User Input. In *Proceedings of the Seventh Workshop in Hot Topics in Operating Systems (HotOS-VII).*

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