Final Report,
*covering the period January 1989–Sept 1989**

48 P

AN ERROR-RESISTANT LINGUISTIC PROTOCOL FOR AIR TRAFFIC CONTROL

Principal Investigator:

Steven Cushing, Ph.D.

Boston University, Computer Science B-4
755 Huntington Avenue Boston, MA 02215

Graduate Research Assistants (Boston University):

Protocol: Suzanne Artemieff and Gabriel Elkin

Interface: Barry Paine and Susan Willard

Undergraduate Research Assistants (Stonehill College):

Database: Ann Sisco and David Ross

Contract No. NAG 2–564
between NASA-Ames Research Center, Moffett Field, CA 94035
and Stonehill College, North Easton, MA 02357

* The **NASA Technical Officer** for this grant is Alfred Lee, Ph.D.,
NASA-Ames Research Center, MS 239-21, Moffett Field, CA 94035

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

The research results described here are intended to enhance the effectiveness of the DATALINK interface that is scheduled by the Federal Aviation Administration (FAA) to be deployed during the 1990's to improve the safety of various aspects of aviation. While voice has a natural appeal as the preferred means of communication both among humans themselves and between humans and machines — as the form of communication that people find most convenient —, the complexity and flexibility of natural language are problematic, because of the confusions and misunderstandings that can arise as a result of ambiguity, unclear reference, intonation peculiarities, implicit inference, and presupposition. The DATALINK interface will avoid many of these problems by replacing voice with vision and speech with written instructions. This report describes results achieved to date on an on-going research effort to refine the protocol of the DATALINK system so as to avoid many of the linguistic problems that still remain in the visual mode. In particular, a working prototype DATALINK simulator system has been developed consisting of an unambiguous, context-free grammar and parser, based on the current air-traffic-control language and incorporated into a visual display involving simulated touch-screen buttons and three levels of menu screens. The system is written in the ***C*** programming language and runs on the Macintosh II computer. After reviewing work already done on the project, new tasks for further development are described.

1. The Prototype DATALINK Simulator

Work to this point has resulted in the construction of a prototype DATALINK simulator, consisting of an unambiguous, context-free grammar and parser, based on the current air-traffic-control (ATC) language and incorporated into a visual display involving simulated touch-screen buttons and three levels of menu screens. The system is written in the ***C*** programming language and runs on the Macintosh II computer.

a. The Visual Interface

Examples of the sorts of message that the system can now handle are listed in Figure 1. A software block diagram for the system as a whole, with the independently modifiable parser indicated in dotted lines, is shown in Figure 2. Figure 3 illustrates a mock top-level screen in the upper left; the actual system has more — and more meaningful — icons. The upper right shows the upper portion of a mock second-level screen of tokens; in the actual system, these depend on which top-level icon was invoked to call the second-level screen. Below the second-level screen is a smaller window, which shares the screen with the window above it. This smaller window echoes the user's selections and enables the user to continue on with a message, or to start over in the event of error. If a message is legal, as stated in the grammar, then it is transmitted to the recipient and cleared from the window; the user can then return to the top-level window, or continue with another message from the same window. If the message is illegal, it remains in the window, and an alert box appears. Figure 4 shows the over-all screen design, which includes an upper window for scrolling in-coming messages. The top-level window uses the entire balance of the screen for icons; second-level windows include the confirmation/echo window at the bottom.

In its present form, the simulator demonstrates the developing grammar, exercises the grammar in a realistic context, and paves the way for a more fully developed DATALINK

system. In effect, it functions as a proof-of-concept by revealing non-obvious design issues and demonstrating the relative feasibility of relevant design choices. It also raises some ergonomic issues which will be dealt with in the next stage of research. A more detailed account of the innards of the system is given in Appendix A.

Weather area between 1 o'clock and 3 o'clock 7 miles.

4 mile band of chaff from 10 miles south of Boston VORTAC to 20 miles north of Baltimore VORTAC.

Traffic alert 9 o'clock, 5 miles, eastbound, converging. Advise you turn right heading 045 and climb to flight level 190 immediately.

Hold short of runway.

Flock of geese, 6 o'clock 4 miles northbound, last reported at altitude 15 thousand 7 hundred.

Contact Logan ground 131.1.

Wind shear alerts all quadrants. Centerfield wind north at 30 knots varying to northeast at 20 knots.

Maintain flight level 203 10 miles past Chicago VORTAC.

Reduce speed by 30 knots.

Descend and maintain altitude 16 thousand 3 hundred. Then reduce speed by 10 knots.

Figure 1: Examples of Legal Messages for the DATALINK Simulator System

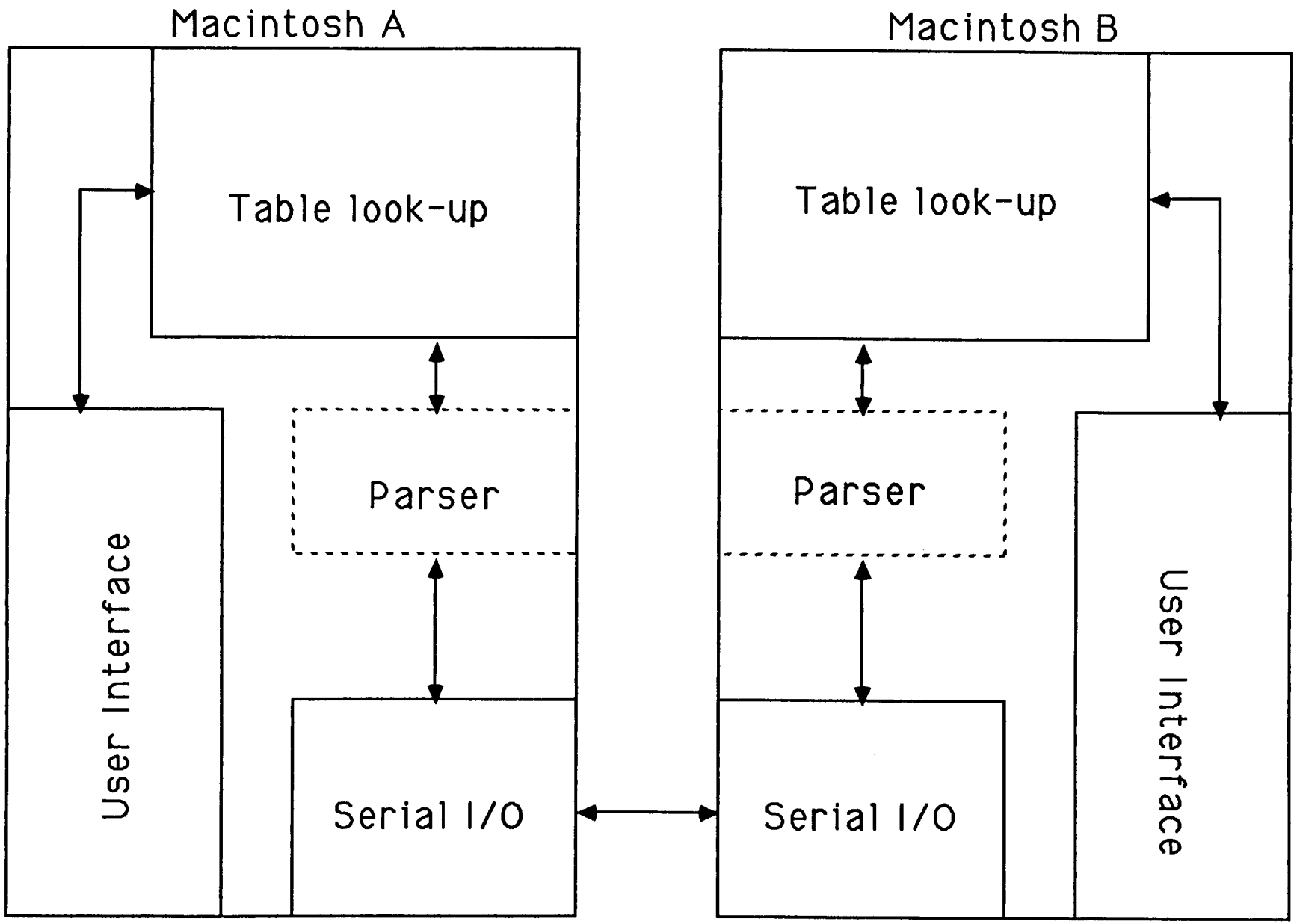


Figure 2: DATALINK Simulator Software Block Diagram

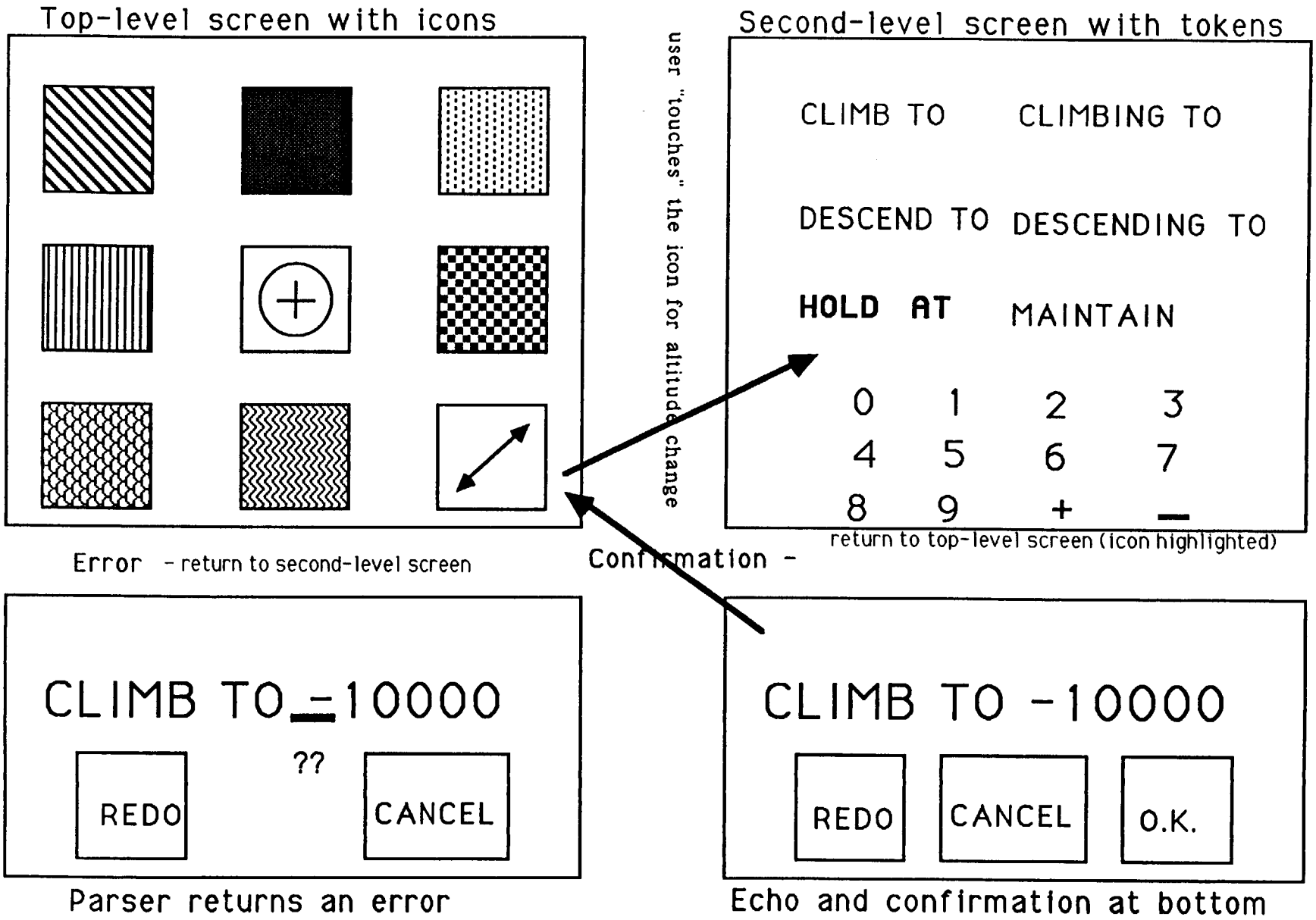


Figure 3. DATALINK User Interface: Typical Screens

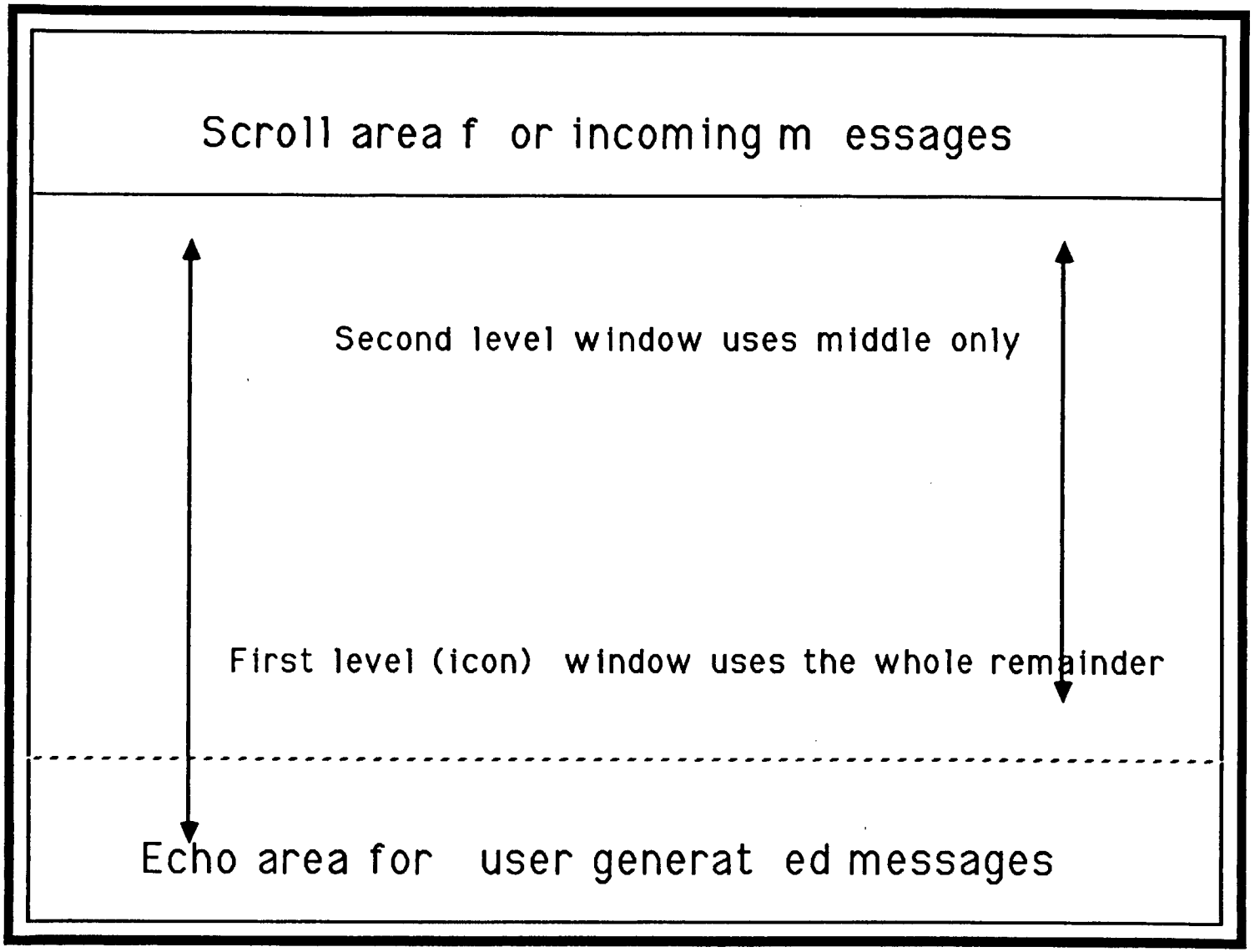


Figure 4: DATALINK Screen Design

b. The Protocol, Grammar, and Parser

The grammar that the parser accepts is based on the communications protocol, or phraseology, specified in the Air Traffic Control handbook (document number 7110.65E), with change pages effective June 6, 1989. The syntax used to specify the phraseology in the handbook is often ambiguous; therefore, a more precise syntax is used here to document the grammar accepted by the parser. That syntax is presented in Appendix B.

The grammar itself comprises a set of tokens and a set of phrases. The tokens are basic units of the grammar that are used throughout the phrases, for example, ***fix***, ***altitude*** and ***heading***. With only a few exceptions, both the tokens and phrases accepted by the parser follow precisely the specification contained in the handbook. The only exceptions fall in one of two categories: either the token/phrase is incomplete — i.e., the parser accepts a restricted version of that specified in the handbook — or the phrase has been intentionally modified. The latter category includes only three phrases, all of which were developed in the earliest stage of the project, when there was still some anticipation of a near-future migration to a voice recognition interface. Fairly minor changes were made to those phrases in an attempt to eliminate areas of potential confusion that would be especially troublesome in an audio (rather than a visual) system interface environment. Further such changes can still be introduced later, in the event that the migration to audio does become desirable.

The incomplete and revised tokens and phrases are listed in Appendix C. The complete, non–modified tokens and phrases are listed in Appendix D. Each token and phrase is preceded by the paragraph number in the handbook where it is specified. In the case of the revised phrases, the original phrase is listed first, exactly as it appears in the handbook, with the modified version following, in the syntax presented in Appendix B.

A number of tokens and phrases described in the handbook were incorporated into the grammar accepted by the parser only partially or not at all because of ambiguity in the handbook specification or because of a total lack of phraseology. Instances of inconsistencies in the handbook specification are listed in Appendix E, while instances of insufficient phraseology are presented in Appendix F. Appendices E and F cover material up through Section 3-8 of the handbook.

Technical notes on the parser itself are provided in Appendix G. These notes pertain only to the parser generated by lex and yacc/bison. They do not cover the modifications made to port the parser to the Macintosh. Notes on these are given in appendix A.

2. The *CALLBACK* Language-errors Database

Work to this point has also resulted in the construction of a database of actual aviation-related language errors, compiled from NASA-Ames Research Center's ***CALLBACK*** newsletter. All language errors reported in ***CALLBACK*** through March 1989 are included in the database. The database was written using the ***Factfinder*** database management system for free-form text and runs on the Macintosh Plus computer. Each instance of an aviation-related language error is recorded on a separate "fact-sheet" record, along with an identification of the issue of ***CALLBACK*** in which it was reported, and can be accessed by means of key words that describe the grammatical nature of the error, the aviation-relevance of the error, by whom it was reported, and other potentially relevant characteristics. Two example records are given in Figure 5. The first is accessed by entering any of the keywords: **1979, August 79, controller, interference, No. 2, noise,** or **not**; the second is accessed by entering any of the keywords: **1986, clearance, controller, inaccuracies in content, No. 87, pilot,** or **September 86**. The designation "p. 1a" indicates that this is the first listed example from page 1 of the indicated issue of ***CALLBACK***. A complete list of all keywords through which records may be accessed, with their frequencies, as printed by the database itself, is given in Appendix H.

3. Further Work

There are four tasks that remain to be carried out for the completion of the DATALINK simulator system: **1. interface optimization**, **2. parser optimization**, **3. grammar augmentation**, and **4. interface augmentation**. It is anticipated that that these tasks will be completed during the coming year. Tasks 1 and 2, though conceptually distinct, necessarily overlap and influence each other, and the same is true of tasks 3 and 4. Tasks 1 and 2 lay the groundwork for phases 3 and 4, which lead to a finished product. It is anticipated that the result of these four tasks, to be completed by September, 1990, will be a working prototype system suitable for testing on controllers and control subjects, with full-scale testing to take place at NASA-Ames Research Center during the summer of 1991. The months leading up to the summer of 1991, will be concerned primarily with designing and arranging for suitable experimental tests. Some refinement of the working system may also be done at that time, as relevant ideas emerge through the test design process.

a. Interface Optimization

The first task for the coming year will consist in incorporating into the prototype *interface* features that are ergonomically desirable but that have not as yet been implemented. For example, there is a need to reorganize the windows in such a way as to distinguish clearly between messages that are needed regularly in *routine* work and messages that need be made available only in *special* circumstances, such as emergencies, dangerous weather, or exceptionally heavy traffic. This is one of the features that the FAA has listed as desirable characteristics of a DATALINK interface. It will also make for a more efficient classification of the types of messages required in the two kinds of circumstances. Furthermore, there is a need to introduce pop-up windows for alternatives, rather than having them always present on the screen. This will serve to reduce the number of items on the screen at one time by ensuring that items appear

only when they are actually relevant to the current concerns of the user. As well as eliminating distractions, this will also provide more room on the screen for larger buttons for the relevant items, a feature that will become increasingly important, as the system moves from the current Macintosh mouse simulation to an actual touch screen.

b. Parser Optimization

The second task for the coming year will consist in incorporating into the prototype *parsing system* features that are ergonomically desirable but that have not as yet been implemented. For example, there is a need to have parsing done in *real time*, so that the user is informed immediately when he has entered an erroneous message, rather than having to enter a corrected one after the first message is rejected. Rejection should also be accompanied by some information as to just what was wrong with the incorrect statement and, perhaps, some suggestions for how to correct it. The current character-by-character backspacing mechanism must also be made more flexible, to maximize ease of correcting. The issue of parser *vs.* interface must be addressed as well. Since the interface is *visual*, with something of a *menu* character, some of the constraints required by the grammar can be incorporated directly in the interface design. The need for such features as range restrictions, co-occurrence relations, and optionality of items makes it unlikely that all grammatical constraints can be handled in this way, without massive overburdening of computer memory capacity, but exactly how much can and should, without sacrificing efficiency and ergonomic considerations, remains to be determined.

c. Grammar Augmentation

The third task for the coming year will consist in expanding to the maximum possible extent the fragment of the protocol that the system is able to handle. Once the issues in tasks a. and b. have been resolved and their solutions implemented, it will be a relatively simple matter to incorporate much more of the language into the grammar,

thereby making it amenable to incorporation into the interface. No new issues or problems of principle are anticipated in this phase, but considerations of memory limitation and/or processor speed may have to be taken into account.

d. Interface Augmentation

The fourth task for the coming year will consist in expanding the interface to accommodate the growing fragment of the protocol, as more and more of the language is incorporated into the grammar. As in phase c., no new issues or problems of principle are anticipated in this phase; however, even more attention will have to be paid to considerations of memory limitation and/or processor speed, especially if the dynamic nesting of pop-up windows requires more than three levels.

The current issue of the excellent Journal of The Air Traffic Control Association has an editorial on air/ground communications and the dangers of partially blocked messages, non-standard phraseology, and plain misunderstandings. As an example, the Editor cites the use of "NOT". "Some controllers still use such non-standard phraseologies as 'CROSS WARWICK NOT ABOVE 6000 FEET'. Consider the implications if a spike of noise happens to come along and blot out the word 'NOT'."

- Callback, No. 2, August 1979, p. 1

. . . level at FL 230. Controller requested our flight " maintain 280 (two eight oh)". Since we were in light turbulence I said "ok" to the first officer; he read back that we were "cleared to maintain 280 (two eight oh)"; I started to climb. At 240 the controller said, "return to 230; that 280 was for airspeed." (all very calmly). . .

- Callback, No. 87, September 1986, p. 1a

Figure 5: Two Example Records from CALLBACK Language-errors Database

REFERENCES

Cushing, S. 1983. "Abstract Control Structures and the Semantics of Quantifiers." ***Proceedings of the First Conference of the European Chapter of the Association for Computational Linguistics.*** Pisa, Italy. August 1983.

Cushing, S. 1984. "Dynamic Model Selection in the Interpretation of Discourse." In Vaina, L. and Hintikka, J. (eds.). 1984. ***Cognitive Constraints on Communication: Representations and Processes***. Dordrecht: Reidel.

Cushing, S. 1987a. "Two Explanatory Principles in Semantics." In Vaina, L., (ed.). ***Matters of Intelligence***. Dordrecht: Reidel.

Cushing, S. 1987b. "Some Quantifiers Require Two–predicate Scopes." ***Artificial Intelligence***. May 1987.

Cushing, S. 1988. "Language and Communication-related Problems of Aviation Safety." **ERIC** Report No. ED 296 595, FL 017 504. Washington, D.C.: U.S. Department of Education. (Supported by an ASEE Summer Faculty Research Fellowship at NASA–Ames Research Center)

Ehrich, R.W. and Williges, R.C. (eds.). 1986. ***Human–Computer Dialogue Design***. Amsterdam: Elsevier.

Falzon, P. 1984. "The Analysis and Understanding of an Operative Language." **IFIP–Interact 84**.

Hopper, K. and Newman, I.A. (eds.). 1986. ***Foundation for Human–Computer Communication: Proceedings of the IFIP WG 2.6 Working Conference on the Future of Command Languages (Rome, Italy, 23-27 September, 1985)***. Amsterdam: North–Holland.

Martin, J. 1985. ***Software Design from Provably Correct Constructs***. Englewood Cliffs, NJ: Prentice-Hall.

NASA-Aviation Safety Reporting System. July 1979–March 1989. ***Callback*** Newsletter. Mountain View, CA: NASA-Ames Research Center.

Parkison, R.C.. 1980. ***An Effective Computational Approach to the Comprehension of Purposeful English Dialogue***. Ph.D. dissertation. Department of Computer Science, Stanford University.

Reilly, R.G. (ed.). 1987. ***Communication Failure in Dialogue and Discourse: Detection and Repair Processes***. Amsterdam: North-Holland.

Robinson, C.P. and Eberts, R.E. 1987, "Comparison of Speech and Pictorial Displays in a Cockpit Environment." ***Human Factors***. **29(1)**:31–44.

Shackel, B. (ed.). 1981. ***Man-Computer Interaction: Human Factors Aspects of Computers and People: Proceedings of the NATO Advanced Study Institute on Man-Computer Interaction (Mati, Greece, September 5-18,1976)***. Alphen aan den Rijn, The Netherlands: Sijthoff & Noordhoff.

Shackel, B., (ed.). 1985. ***Human-Computer Interaction - Interact '84: Proceedings of the IFIP Conference organized by the Task Group on Human–Computer Interaction - formerly IFIP WG 6.3 (London, U.K., 4-7 September, 1984)***. Amsterdam: North-Holland.

APPENDIX A: The DATALINK Simulator System

The simulator partitions the Macintosh II screen into three rectangular windows. The uppermost of these windows displays incoming messages from another system, such as an aircraft cockpit. The middle screen is a collection of icons representing various topics in the ATC vocabulary, such as bird warnings or vectoring. Double clicking an icon will cause a "menu" (not to be confused with a Macintosh pull-down menu) to be displayed in the middle window, overlaying the icon menu. A given menu consists of a series of buttons containing key words or phrases in the lexicon. The buttons are arranged to correspond to legal sentences in the grammar. Some items, such as numbers, occur with such frequency that they have been placed near the bottom of each menu. In some cases, such as weather warnings, a menu may be so complex that it requires sub-menus. In this instance the user must pass through an additional layer of icons before reaching a button menu.

The user selects the text from a button by clicking it with a mouse; the keyboard is not used at all in this touch-screen simulation. As each button is clicked, the corresponding text is added to a sentence being developed in the undermost window. A backspace button is provided to correct erroneous choices; an OK button indicates when the sentence is complete. Once the sentence is developed and OK'ed, it is sent to a parser. Failure to parse is indicated by an alert box, while successfully parsed sentences are passed to the serial port and on to a recipient, another Macintosh II. The user can then select CANCEL to return to the previous level of menu or can continue to create sentences from the current menu.

Program Development and Organization

The simulator software was built around a public domain application skeleton called

TransSkel. The TransSkel package provided the basic tools to manipulate Macintosh windows and to respond to events. TransSkel (and the simulator) are based on the Lightspeed ***C*** compiler for the Macintosh computer. Lightspeed provides a ***C*** development environment for the Macintosh; documentation for that environment can be obtained from Symantec Corp., which distributes the compiler. Generally, ***C*** programs are organized as entities called "projects" in the Lightspeed vernacular and are indicated by an icon of a black square within a white square. Double clicking a project icon will reveal a series of menus and a list of program modules in the project (a project is a single linked piece of code).

The main program in the DATALINK simulator is called datalink.c. The "main" routine can be found at the end. Datalink.c is an engine which displays and interprets icons and menus and builds sentences for the parser based on user selected buttons. It can be termed an engine because it knows little or nothing about the contents of a given menu. All menus and icons are contained in a Macintosh "resource file" called "datalink.proj.rsrc". The contents of this file can be altered or completely replaced to produce an entirely different result when processed by the DATALINK program, as long as the parser has also been appropriately replaced. The support/expansion of the project will occur in three major areas:

The DATALINK engine (datalink.c)
The parser (y.tab.c and lex.yy.c)
The resource file (datalink.rsrc)

<u>The Engine</u>

The datalink engine operates through several dialogues/windows on the Macintosh. Most of the dialogues have an assortment of "buttons" to "press". Each button represents a text string; the whole dialogue represents a datalink "menu" such as

"altitude". In some cases, such as "weather", a dialogue/menu may be complex enough to warrant several sub-dialogues/sub-menus. In this case the top-level dialogue will have icons representing each sub-dialogue that may be selected. The main menu (highest level) has only icons, each representing a specific DATALINK topic, such as bird reports, weather, and so on.

In addition to the "menu" dialogues two specialized dialogues exist. These two each consist only of a static text box. One dialogue is for the display of incoming messages and the other is for the display of outgoing messages; the messages formed as a user picks buttons in one of the menus. These two dialogues go at the top and bottom of the screen (respectively) while the menu dialogues overlay each other in the center.

A mouse "hit" by a user generates an "event" in the Macintosh. These events are handled by a routine called "Event". The routine determines if the item "picked" by the mouse (causing the event to be generated) was an icon or some text. In the case of an icon, a mask is used to determine the ID of the menu to display (which is then displayed). A text hit may be a special case (e.g. "QUIT") or a text string from a menu button. In the latter case the string is appended to the message/sentence being formed and displayed in the "outgoing" box. An "OK" button signals that the text should be parsed and then sent out if the parse is successful. An unsuccessful parse will generate an alert box. Once the appropriate action has been taken the engine waits for the next event.

The primary purpose of the engine is to execute a "forever" loop of event processing and dialogue display. The dialogues are all obtained from a resource file (see below) and are initialized as part of the software start-up. The resource file must have a dialogptr (dialogue pointer) initialized for each dialogue in the datalink.c module.

Existing dialogues can be modified without touching the engine; only the resource file needs to be updated (a resource file can be buried in an application). New menus/dialogues need to have "their" pointers and ID's added to the datalink.c file.

The main routine of datalink.c is located near the bottom of the file. This routine does some initialization. In addition to it and the Event routine, a number of small utility routines exist within datalink.c:

TmpDialog	creates second and third level dialogues (menus)
SetDCtl	turns the debug check box on/off in the main menu
SetDText	sets the string in a dialogs static text
pprintf	replaces printf — used to write text to dialog box instead of console
display_incoming	prints received serial data in a dialog window
Close	closes a dialog box
Clobber	remove dialog resource and close
DoFileMenu	handles the file menu at screen top
DoAbout	under the apple menu - the "about datalink"
DemoDialog	sets up first level dialogues
OpenSerial	opens the serial communications port
GetData	gets incoming serial data into a local buffer
SendData	outputs successfully parsed messages to a serial port

In addition to these local routines some routines are found in the TransSkel.c module where they are documented. These include routines such as SkelWoa ("kill" the program) or SkelInit. Books such as Inside Macintosh, Volumes I - V document all of the routines in the Macintosh library.

The Parser

The parser has been developed on the Vax computer using two standard Unix tools, lex and yacc, and then ported to the Macintosh II. There are two source files containing the definition of the parser, which are called here lex.src and yacc.src and which must be processed by the corresponding utility to yield lex.yy.c and y.tab.c. Several relatively painless changes must be made in order to make the Vax version of the parser compatible with the Macintosh DATALINK engine:

The word "OVER" was used in the original Vax version to indicate an end of session. In the Macintosh version it is just another phrase; a "caret" (^) now terminates each phrase/phrase group to force the parser to return a status. The change must be made in both lex.src and yacc.src. To lex.src the line

```
"^"          (caret);
```

was added, right after a similar line for "OVER"; this location is not critical, however. In yacc.src the definition of "OVER" had to be changed to be a simple phrase, so the lines

```
| phrase OVER

    yyclearin;
```

were added right after the lines for "STOP" (but before the terminating ";"). The "definition" for "end" was then changed from "OVER" to "caret". The actual lines are

```
end   : caret
      ;
```

It is also necessary that the word caret appear in the TOKEN LIST at the start of the file.

The other major change is to change all occurrences of "printf" to "pprintf" in yacc.src. The parser ordinarily prints out an inverted parse tree via the printf statements. This must not happen by default in the Macintosh version. Furthermore, printf defaults to the system console, a special window on the Macintosh.; pprintf is a routine in the DATALINK engine that will print to the DATALINK's own window.

Once the above changes have been made, lex.src and yacc.src can be processed. A Sun workstation was used to do this, but any Unix (or compatible) system should do just as well. The commands to do this are:

```
lex lex.src
yacc yacc.src
```

The resulting files, lex.yy.c and y.tab.c, could then be included in the DATALINK simulator. However, there are several changes to lex.yy.c which must be made. First, there is a redundant definition which must be removed; the Lightspeed compiler will not accept it. The actual line:

```
#define NLSTATE yyprevious=YYNEWLINE
```

occurs both near the top of lex.yy.c and at around line 2329; the second occurrence was commented out. Also, the following two lines must be deleted:

```
#define output(c) putc(c,yyout)
#define input() (((yytchar=yysptr>yysbuf?U(*--yysptr): getc(yyin))
                ==10?(yylineno++,yytchar):yytchar==EOF?0:yytchar)
```

The first of these will cause output to the console under certain circumstances. It is replaced by a null routine in the file "lex.h", which also includes a redefinition of the second line. The new definition looks for input from a special structure called yystr, a pointer to a string to be parsed, instead of from yyin (standard in). The DATALINK engine points yystr to the user-generated phrase to be parsed and then calls yyparse() to do the parsing.

Finally, to get all of the new definitions, the file "lex.h" must be included. It was included right after the line

```
#define NEWLINE 10
```

so that the new version looks like this:

```
#define NEWLINE 10
#include "lex.h"
```

There is a potential problem with the Lightspeed compiler as the grammar grows in size. Static memory is limited to 32k bytes. If a future version of the parser causes this limit to be exceeded then remedial action must be taken. This is described in the Lightspeed manual and involves allocating memory from the heap. Since the parser uses a lot of pre-initialized tables, the allocated memory must be similarly initialized, perhaps from an input file. At this point these issues are merely a concern, rather than an actual problem.

The modified lex.yy.c and y.tab.c can be downloaded to a Macintosh (or even modified on the Macintosh) and included in the DATALINK simulator folder. They should replace previous versions of lex.yy.c and y.tab.c.

The Resource File

Modifying an Existing Resource

ResEdit is a utility used to modify a resource file. The resource file is accessed by the "engine" portion of DATALINK in order to display menus/icons/alerts. Since the information in a resource file is nearly transparent to the engine, the appearance and/or information content of one or more menus (DATALINK menus = collection of icons and text buttons) can be altered without changes or re-compilation to the "project".

Double clicking on ResEdit will start the editor, and a list of disks/folders will be displayed. Repeated double clicking of folders will reach the folder which contains the resource desired for editing. In some instances only the executable program (e.g. "demo") will need to be edited. Alterations to the executable code affect only it; future versions built from the "project" file will not contain any changes. Alterations to the project resource file (which is incorporated into each application built by the "project" software) will be put into each new application (such as demo). The project resource file is currently named "datalink.proj.rsrc", although this name could be changed in the future.

Once an application or resource file has been double clicked, a list of resource categories (e.g. DLOG, ICON, DATA, ALRT) will be presented. ICONs and DLOGs are the categories most commonly used by the DATALINK software. To change an ICON double click on "ICON" in the list and then double click the "replica" of the ICON whose alteration is desired. This will bring up the icon in a Macpaint type "fat bits" editor (an enlarged 32 x 32 grid). Toggle the squares (bits) to be turned on/off by clicking on them with the mouse. The results can be saved by clicking the "close" box of each window until an alert box appears which asks "Save 'filename' before closing?"; answer "Yes". "Quit" on the pull-down menu for "File" may then be selected. The changes made will

appear in the application that was edited when the application is run. If a project resource file was edited, the changes will appear in the next application built or when "Run" is chosen in one of the project pull-down menus.

A DLOG (dialogue) may be edited in a similar way. Double click DLOG, and then double click the appropriate DLOG ID. The window will contain a miniature replica of the menu displayed by DATALINK (e.g. a menu for "chk speed"). Double clicking on the replica will expand it to a full sized "menu" displayed in a window. Button entries can be selected by clicking them once. They can be dragged or re-sized; all windows in ResEdit can be dragged, if necessary. Double clicking a button will open it so that the text it contains can be altered. To get rid of a button, select it and then choose "Cut" from the "Edit" pull-down menu. To create a new button choose "New" from the "File" menu. This will result in a window with some selections to make. "Button" is the default choice; there are other types of controls which DATALINK typically does not use. The button is "enabled" by default. This means when a button is clicked (running the application) an "event" will happen; it should be kept that way, or the button will be dead. There will be an editable text box. Change the text from "NEW" to the word or phrase desired for the button. The ATC grammar is case-sensitive, so upper and lower case should be carefully distinguished. Once all of the changes have been made (to one or more DLOGs), then close as described above for ICONs.

Adding a New Resource

New menus and their icons can be added to the file "datalink.proj.rsrc". To add an icon proceed as above until the window with the icon replicas is open. Choose "New" from the "File" pull-down menu. The same fat bits editor will be activated; create the desired icon. The new icon must have an ID that will correspond to the menu (DLOG) it is being created for. The following system is used. We add 10,000 to the ID of the DLOG to yield

the icon ID. When the icon is double clicked by a user we subtract 10,000 to determine which menu (DLOG) to display. By convention level one (main menu) DLOGs (menus) have ID's between 1000 and 1999, level two (sub-menus) between 2000 and 2999, etc. Determine the next available DLOG ID which will correspond to the icon before beginning to create an icon. The following process is used to give an icon a specific ID (by default all new resources are given a "random" non-conflicting ID which can be changed to another unique number by the ResEdit user):

With the current ICON open or selected, choose "Get Info" from the the "file" pull-down menu.
Change the ID number to the desired one (corresponding DLOG ID +10000).

To create a new DLOG proceed as in editing an existing DLOG but choose "New" from the "File" pull-down menu once the list of DLOGs is displayed. A window with the new menu will be displayed. The window will display the DLOG ID in its title/drag bar. The new menu (displayed in miniature as an "inner" window) will need to be sized. Click on the forenamed title/drag bar of the "outer" window. Then use the "Display as Text" from the "DLOG" pull-down menu (top menu bar - right-most) to open an information box. The window coordinates can be entered in the appropriate box; they should match those for the "main menu". A title can be added, if desired. The same box will also receive the ResID for the DLOG; this is explained below. Double click (open) the "inner" window (menu being created) and create buttons and text as above. Once the menu is as desired, the ID can be changed to be the appropriate one as follows:

With the DLOG opened or selected, choose "Get Info" from the "File" pull-down menu. Change the ID to the appropriate one.

With the DLOG opened, click on its title/drag bar. Choose "Display as Text" from the pull-down called "DLOG". Change the resID to the DLOG ID number you used in the last step.

Double click the created "menu" within the DLOG window. Choose "Get Info" from the "File" pull-down menu. Change the DITL ID to be the same as the DLOG ID.

Note: Sometimes a new icon or DLOG will not show up in a list, until the ICON or DLOG list is closed and re-opened.

Appropriate defines and DialogPtrs to the datalink.c program will also need to be added.

APPENDIX B: SYNTAX FOR GRAMMAR DEFINITION

- All upper case letters indicate that the words are to be spoken verbatim
- All lower case letters enclosed in angle brackets indicate "tokens", ie, variables the syntax of which has been previously defined
- Text in italics followed by a colon is used to specify conditions for alternative phrases
- Brackets indicate that the enclosed data may or may not be applicable
- Braces indicate that the enclosed data represents a description of what is to be said
- Parentheses are used for grouping or to set off explanatory text (indicated by italics)
- A slash indicates that one of the two words which the slash separates is to be selected
- A vertical line, generally used in conjunction with parentheses, indicates that one of the two groups of words which the vertical line separates is to be selected
- An ampersand indicates that both words or phrases separated by the ampersand are to be used
- When alternatives are provided for entire sentences, they are separated by an "or" indented on one line
- A double plus sign indicates one or more repetitions of the preceding token; a plus sign followed by a number indicates that the preceding token should be repeated for a total equal to the given number

Example:

MAINTAIN/CRUISE <altitude>.
    or
MAINTAIN <altitude> (UNTIL <time> | PAST <fix> | <digit++> MILES/MINUTES PAST <fix>).

where altitude has been previously defined as:

If the number of feet is less than 18,000:
    ALTITUDE <digit++> THOUSAND [<digit> HUNDRED]
else:
    FLIGHT LEVEL <digit++>

where digit, time and fix have also been previously defined.

APPENDIX C: REVISED & INCOMPLETE GRAMMAR FRAGMENTS

Token Definitions:

(2-85):	<altitude>	*If the number of feet is less than 18,000:* ALTITUDE <digit++>* THOUSAND [<digit> HUNDRED] *else:* FLIGHT LEVEL <digit++>* NOTE: this definition is incomplete in that it does not provide for MDA/DH altitudes (2-85b(3))
(2-17b):	<ffunction>	(*facility function*) NOTE: currently includes several functions, eg. Center
(2-92):	<fix>	(<lname> <navaid> | <lname> (DME FIX | WAYPOINT | <radial> | <localizer> | <fix-azimuth>)) NOTE: definitions for radial, localizer and fix-azimuth (unnamed fixes--2-92a) have not yet been provided, as there are too many unknowns
(2-86):	<fname>	(*facility name*) NOTE: currently includes several facilities, eg. Logan
(2-85k):	<frequency>	<digit++> [(.<digit>[<digit>] | KHZ)]*
(2-85h):	<heading>	HEADING <digit++>*
(2-17b):	<lname>	(*location name*) NOTE: currently includes several cities, eg. Boston
(2-85c):	<ltime>	(*local time indicator*) NOTE: currently includes only EST, M, PT
(2-91):	<navaid>	VOR/VORTAC/TACAN
(2-85l):	<speed>	<digit++>* KNOTS
(2-85c):	<time>	<digit++>[ZULU [<digit++> <ltime>]]* NOTE: currently there is no space between the fourth digit and the word ZULU; this should be changed.
(2-21a(1)):	<azimuth>	<digit++>* O'CLOCK
(2-85f):	<altimeter>	ALTIMETER <digit++>*

* Eventually, a subroutine needs to be added to check that the given number is within an appropriate range for the given token.

(2-88): <craft-type> *type of aircraft*
NOTE: currently includes only DC-8 and Apache

(2-90c): <LMF-color> color of L/MF airway
NOTE: currently includes only red and blue (not clear whether more are needed).

(2-22a): <bird-species> *type of bird*
NOTE: currently includes only ducks, geese, gulls, sparrows

(2-22a): <bird-size> *size of bird*
NOTE: currently includes only small, large

Incomplete phrases due to ambiguity in ATC Manual:

(2-103c): (WEATHER | CHAFF) AREA BETWEEN <azimuth> AND <azimuth> <miles>,
or
<digit++> MILE BAND OF (WEATHER | CHAFF) FROM [<miles> <direction> OF] <fix> TO [<miles> <direction> OF] <fix>,
or
<digit> INTENSE WEATHER ECHO BETWEEN <azimuth> AND <azimuth> <miles>, MOVING <direction> AT <digit++> KNOTS TOPS <altitude>.
NOTE: the initial digit and the following adjective need to fall within a prescribed range.
or
DEVIATION APPROVED.
NOTE: this complete phrase alternative reads as follows:
DEVIATION APPROVED, (restrictions if necessary), ADVISE WHEN ABLE TO:
RETURN TO COURSE
or
RESUME NORMAL NAVIGATION.
or
FLY <heading>,
or
PROCEED DIRECT TO <fix>. UNABLE DEVIATION.
NOTE: this complete phrase alternative includes the following directions at the end of the phrase:
(state possible alternative courses of action)

Revised phrases related to confusion between homonyms:

(5-101a): INCREASE/REDUCE SPEED TO (specified speed in knots), or TO MACH (mach number), or (number of knots) KNOTS.

Revised: (ACCELERATE | [IF PRACTICAL,] SLOW) TO (SPEED <speed> | <mach number>)
or
(INCREASE | [IF PRACTICAL,] REDUCE SPEED BY (<number> KNOTS | <mach number>).

APPENDIX D: NON-MODIFIED (AND COMPLETE) GRAMMAR PHRASES

Token Definitions:

(2-85a):	<digit>	0/1/2/3/4/5/6/7/8/9
(2-85l):	<mach number>	MACH [1].<digit> [<digit>]
(2-21a(2)):	<direction>	<quad> | <location>
(2-21a(2)):	<quad>	NORTH | SOUTH | EAST | WEST
(2-21a(2)):	<location>	NORTHEAST | NORTHWEST | SOUTHEAST | SOUTHWEST
(2-21a(3)):	<miles>	<digit++> MILES
(2-21a(4)):	<relative-movement>	CLOSING | CONVERGING | PARALLEL | OPPOSITE | DIVERGING | OVERTAKING | CROSSING (LEFT TO RIGHT | RIGHT TO LEFT)
(2-90):	<route>	VICTOR <digit++> [ROMEO | <location>]. *or* J <digit++> [ROMEO], *or* <LMF-color> <digit++>, *or* NORTH AMERICAN ROUTE <digit++>, *or* (IR | VR) <digit++>.

Legal Phrases:

(2-17b): CONTACT (<fname> | <lname>) <ffunction> [<frequency>] [AT (<time> | <fix> | <altitude>)].

(2-17c): CHANGE TO MY FREQUENCY <frequency>.

(2-17d): REMAIN THIS FREQUENCY.

(2-6a): LOW ALTITUDE ALERT, CHECK YOUR ALTITUDE IMMEDIATELY. THE ((MEA | MVA | MOCA | MIA) IN YOUR AREA | MDA | DH) IS <altitude>.

(2-6b): TRAFFIC ALERT [<azimuth> | <direction>, <miles>, [<quad>BOUND], <relative-movement>]. ADVISE YOU [TURN LEFT | RIGHT [<heading>] AND] CLIMB | DESCEND [TO <altitude>] IMMEDIATELY.

(2-21a): TRAFFIC, <azimuth> | <direction>, <miles>, [<quad>BOUND], <relative-movement>, [<craft-type>,] <altitude> | ALTITUDE UNKNOWN.

(2-21a(*8): [<azimuth> | <direction>] TRAFFIC NO LONGER A FACTOR.

(2-21b): TRAFFIC, (<miles> | <digit++> MINUTES) <direction> OF <fname> | <fix>, <direction>BOUND, [<craft-type>,] <alt :uc >> | ALTITUDE UNKNOWN, ESTIMATED <fix> <time>.
or
TRAFFIC, NUMEROUS TARGETS VICINITY <fname> | <fix>.

(2-22a): FLOCK OF (<bird-species> | [<bird-size>] BIRDS), <direction>BOUND ALONG <route> | <azimuth> <miles> <direction>BOUND | VICINITY (<fname | <fix>)), (LAST REPORTED AT <altitude> | ALTITUDE UNKNOWN),
or
NUMEROUS FLOCKS (<bird-species> | [<bird-size>] BIRDS), VICINITY (<fname | <fix>), (LAST REPORTED AT <altitude> | ALTITUDE UNKNOWN).

(2-102c): REQUEST FLIGHT CONDITIONS. REQUEST FLIGHT CONDITIONS (OVER <fix> | ALONG PRESENT ROUTE | BETWEEN <fix> and <fix>).

(2-110b): THE <fname> [<time>] <altimeter>.

(3-7b): HOLD (SHORT OF RUNWAY) | (IN POSITION).

(3-8): WIND SHEAR (ALERT) | (ALERTS (TWO | SEVERAL | ALL) QUADRANTS. CENTERFIELD WIND <direction> AT <speed> (. <direction> BOUNDARY WIND <direction> AT <speed>) | (VARYING TO <direction> AT <speed>).

APPENDIX E: INCONSISTENCIES IN THE GRAMMAR SPECIFICATION

1-24: MLS (Microwave Landing System) has been omitted from the list of abbreviations.

2-21: Phraseology is ambiguou . From procedural description and examples, it appears that phraseology should read:

TRAFFIC, (<number> O'CLOCK | <direction>), <number> MILES, (<direction>–BOUND &/| <relative movement>, *& if known:* [<aircraft type>], (<altitude> | ALTITUDE UNKNOWN).

2-21a: The third example is misplaced; it belongs under Section 2-21b.

2-22: The phraseology is incorrect:

FLOCK OF (species, if known) (size, if known) BIRDS...
should read:
FLOCK OF (*if known:* {species} | *if known:* {size} BIRDS),...

2-85k(3): The second example contains the digit "nine"; everywhere else that digit reads "niner."

3-9a(3): The phraseology is not consistent with the Note. The first alternative for the phraseology reads:

(Identification), PROCEED (direction)–BOUND, (other instructions or information as necessary).

However, the Note states: *It is important that the pilot be aware of the fact that the directions or headings being provided are suggestions or are advisory in nature...*

Proposed revision:
<Identification>, SUGGEST PROCEED <direction>–BOUND.

4-89: The phraseology is inconsistent with the procedural description: the orders in which the various pieces of information are listed differ.

5-101b(1): The phraseology is not consistent with that of 5-101b(2) and 5-101a: it does not provide for specification of speed in terms of mach. This has been corrected in the revised grammar that is accepted by the parser (see Appendix B).

(5-101b): REDUCE SPEED: TO (specified speed), or (number of knots) KNOTS. THEN, DESCEND AND MAINTAIN (altitude).

or

DESCEND AND MAINTAIN (altitude). THEN, REDUCE SPEED: TO (specified speed in knots), or TO MACH (mach number), or (number of knots) KNOTS.

Revised: (SLOW TO (SPEED <speed> | <mach number>) | REDUCE SPEED BY (<number> KNOTS | <mach number>)). THEN, DESCEND AND MAINTAIN <altitude>.

or

DESCEND AND MAINTAIN <altitude>. THEN, (SLOW TO (SPEED <speed> | <mach number>) | REDUCE SPEED BY (<number> KNOTS | <mach number>)).

Revised phrases related to confusion among various measurements:

(4-45a): MAINTAIN/CRUISE ALTITUDE. MAINTAIN (altitude) UNTIL (time), or PAST (fix), or (number of miles or minutes) MILES/MINUTES PAST (fix).

Revised: MAINTAIN/CRUISE ALTITUDE.

or

MAINTAIN <altitude> (UNTIL <time> | PAST <fix> | (FOR A DISTANCE OF <number> MILES | FOR A TIME OF <number> MINUTES) PAST <fix>).

APPENDIX F: INSUFFICIENCIES IN THE GRAMMAR SPECIFICATION

2-6a: (Identification) LOW ALTITUDE ALERT. CHECK YOUR ALTITUDE IMMEDIATELY. THE, *as appropriate*, MEA/MVA/MOCA/MIA IN YOUR AREA IS (altitude),
or if past the final approach fix (nonprecision approach), or the outer marker, or the fix used in lieu of the outer marker (precision marker),
THE, *as appropriate*, MDA/DH (*if known*) IS (altitude).

The last part of phraseology is unclear, ie, to what the phrase "if known" refers, and what should be spoken if it's not known.

2-18: (Requested operation) APPROVED.
UNABLE (requested operation).
When necessary,
(reason and/or additional instructions).

Phraseology needed for "requested operation".

2-20: CAUTION WAKE TURBULENCE (traffic information).

What is meant by "traffic information"? Does that pertain to phraseology specified for use when alerting aircraft about other aircraft in close proximity? Need clarification/phraseology.

2-21a(7), 2-34, 2-73, 2-75, 2-79, 2-83: Total lack of phraseology.

2-21b: First example is inconsistent with the phraseology. The phrase "descending to one six thousand" is not accounted for. Is the phraseology insufficient?

2-55: What are the approved codes to designate aircraft type, as listed in the "Contractions Handbook"?

2-90c: What are all the possible colors that can be used to designate an L/MF airway?

2-90e: What are all the possible letter combinations for MTRs?

2-101c: ATTENTION ALL AIRCRAFT, SIGMET/CONVECTIVE SIGMET/CENTER WEATHER ADVISORY (ident.). (Brief description of area affected and type of weather.)

To whom/what does "ident." refer? Need phraseology for "Brief description of area affected and type of weather." Can we make use of existing phraseology, eg. that given in 2-103 to describe "area affected"?

2-102d, 2-106, 2-107: Total lack of phraseology.

APPENDIX G: NOTES ON THE PARSER

Code Structure

The principle function of the parser is yyparse, which repeatedly calls the principle function generated by lex, yylex. Very simple main and error handling functions are provided in the yacc library. Although the current p rser makes use of these standard library routines, more elaborate routines could be written to replace them.

Description of Executable

The parser is run by entering the command *a.out*. The rules in yacc have been specified such that the parser will continue to accept input until either 1) an error is encountered, or 2) the phrase *Over* (with upper or lower case *O* and with or without a period) is entered. Obviously, the choice of *Over* to signal the end of a session was arbitrary and any other phrase or character combination could be substituted quite easily. If an error is encountered, ie, an incorrect token is returned from yylex, the standard error routine is called, which prints the token and message *syntax error*, and the session is terminated.

If yylex encounters input that does not match any rule, it simply prints that input to the standard output (the screen in this case) and continues processing the input until it recognizes a token, which it then returns to yyparse. In other words, yyparse is never made aware of the superfluous input, so a syntax error is not generated.

In order to force the generation of an error in such cases, a "wild card" rule was added to the lex input so that any string of alphanumeric characters not recognized by one of the other rules would return the token *ERR*. Although this yielded the desired result, it has the unfortunate side effect of greatly limiting the number of rules that could be defined in the lex input. The wild card rule generates such a huge number of transitions in the deterministic finite automaton, that it limits to an unacceptable degree the number of other rules that can be specified.

Therefore, the wild card rule has been removed from the lex input, so the current parser merely prints out any extraneous input without generating an error. This could be changed by writing a different version of yylex by hand. Alternatively, there may be a fairly straight forward way of accomplishing it via the lex input file, but that needs further investigation.

Division of Rules between L ex and Yacc

The guiding principle used in the parser developed thus far has been to define in lex basic units of the grammar that are used in multiple fragments unless these units are extremely complex. For example, TIME is defined as a token. On the other hand, although "fix" is used as a basic component throughout the phraseology, the definition of fix is much too complex to justify its definition as a token. Moreover, several of its subcomponents are used (as basic components in and of themselves) in other fragments. ALTITUDE was originally defined as a token in lex that could be returned by one of two rules, one for altitude and the other for flight level. However, the two rules were subsequently transferred to yacc, and the original token was split into two separate tokens (one for altitude and one for flight level) because altitude (in the narrow sense of the word) was explicitly used in the phraseology.

An important note here is that items such as location names and facility names have been implemented as tokens such that the actual proper nouns are listed as separate lex rules that return the appropriate token, eg, Boston, Baltimore, Chicago, Atlanta and Los Angeles all return the token LNAME. The proper way to handle this is to define a marker, such as a per cent sign, to prefix any proper name, and then have an action associated with the yacc rule containing the token, that calls a subroutine to check a table (probably a B-Tree file) containing all the specific valid strings for that token.

Similarly, tokens that include numbers, such as TIME and HEADING, need to be screened for invalid numbers. This can be accomplished fairly easily by associating an action with each yacc rule that contains such a token. The action should call a subroutine that takes a literal to identify the token type and the value of the specific token instance. The subroutine could then do a simple table look-up and return a flag indicating whether the value is valid.

As the grammar increased in size, a number of problems were encountered, mostly related to restrictions in yacc. The best overall solution would be to switch from yacc to bison. Bison, which is distributed by the Free Software Foundation, is almost identical to yacc, but with fewer restrictions. No changes need to be made to the yacc input file before running bison. This was tested using the yacc input file for the parser described here, and no problems were encountered.

APPENDIX H:

Keywords for the ***CALLBACK*** Language-errors Database,

with Frequencies,

as printed by the Database Itself

(9)	Created on 7/12/88	(1)	ACKNOWLEDGE
(5)	Created on 7/22/88	(6)	ADDRESSEE
(3)	Created on 7/29/88	(10)	ALTIMETER
(4)	Created on 8/10/88	(46)	ALTITUDE BUST
(14)	Created on 3/21/89	(27)	AMBIGUOUS PHRASEOLOGY
(8)	Created on 3/22/89	(10)	APPROACH
(9)	Created on 3/30/89	(5)	APRIL 81
(6)	Created on 4/3/89	(1)	APRIL 82
(8)	Created on 4/4/89	(1)	APRIL 83
(12)	Created on 4/6/89	(2)	APRIL 84
(5)	Created on 4/7/89	(1)	APRIL 85
(8)	Created on 4/11/89	(2)	APRIL 86
(15)	Created on 4/12/89	(4)	APRIL 87
(6)	Created on 4/19/89	(1)	APRIL 88
(7)	Created on 4/25/89	(2)	ASRS
(7)	Created on 4/26/89	(28)	ASSUMPTION
(15)	Created on 4/27/89	(1)	ATC
(6)	Created on 4/29/89	(6)	ATIS
(1)	Created on 4/30/89	(1)	AUGUST 79
(8)	Created on 5/4/89	(4)	AUGUST 80
(11)	Created on 5/5/89	(1)	AUGUST 81
(22)	Created on 5/13/89	(2)	AUGUST 83
(28)	Created on 5/14/89	(1)	AUGUST 84
(217)	Modified on 8/30/89	(4)	AUGUST 85
(12)	1979	(3)	AUGUST 86
(21)	1980	(4)	AUGUST 87
(27)	1981	(8)	CALL SIGN
(23)	1982	(7)	CAPTAIN
(18)	1983	(4)	CENTER
(17)	1984	(2)	CLEAR
(28)	1985	(59)	CLEARANCE
(24)	1986	(7)	CLIPPING
(23)	1987	(66)	CONTROLLER
(22)	1988	(1)	COPILOT
(2)	1989	(1)	DECEMBER 79
(3)	ABSENT-EQUIPMENT FAILURE	(2)	DECEMBER 80
(8)	ABSENT-NOT SENT	(1)	DECEMBER 81

(4)	DECEMBER 82	(4)	JANUARY 87
(1)	DECEMBER 84	(2)	JANUARY 88
(4)	DECEMBER 85	(1)	JULY 79
(2)	DECEMBER 86	(1)	JULY 82
(2)	DECEMBER 87	(3)	JULY 83
(3)	DECEMBER 88	(3)	JULY 84
(13)	DELIBERATE	(3)	JULY 85
(3)	FAILURE TO REPLY	(1)	JULY 86
(2)	FATIGUE	(1)	JUNE 80
(4)	FEBRUARY 80	(2)	JUNE 81
(1)	FEBRUARY 81	(1)	JUNE 82
(1)	FEBRUARY 82	(1)	JUNE 85
(2)	FEBRUARY 84	(2)	JUNE 86
(1)	FEBRUARY 85	(2)	JUNE 88
(2)	FEBRUARY 86	(3)	KEYING
(2)	FEBRUARY 88	(1)	LTSS
(1)	FEBRUARY 89	(4)	MARCH 80
(3)	FIRST OFFICER	(2)	MARCH 82
(19)	FLIGHT LEVEL	(2)	MARCH 83
(16)	FREQUENCY	(4)	MARCH 84
(3)	GARBLED PHRASEOLOGY	(1)	MARCH 85
(3)	HARRASSMENT	(1)	MARCH 86
(13)	HEADING	(3)	MARCH 87
(1)	HELICOPTER	(3)	MARCH 88
(6)	IDENTIFIER	(1)	MARCH 89
(16)	INACCURACIES IN CONTENT	(1)	MAY 85
(23)	INACCURATE - TRANSPOSITION	(1)	MAY 80
(1)	INACURRATE - TRANSPOSITION	(1)	MAY 82
(5)	INCOMPLETE CONTENT	(2)	MAY 83
(5)	INTERFERENCE	(4)	MAY 85
(5)	IRONIC	(2)	MAY 86
(1)	JANUARY 80	(2)	MAY 87
(11)	JANUARY 81	(1)	MAY 88
(5)	JANUARY 82	(4)	MICROPHONE
(1)	JANUARY 83	(26)	MISINTERPRETABLE
(1)	JANUARY 84	(1)	MISSING
(2)	JANUARY 86	(22)	MISUNDERSTANDING

(3)	MODE	(1)	NO. 35
(14)	NO RADIO	(1)	NO. 36
(1)	NO. 1	(1)	NO. 37
(3)	NO. 100	(2)	NO. 39
(2)	NO. 102	(2)	NO. 4
(2)	NO. 103	(1)	NO. 40
(2)	NO. 104	(4)	NO. 41
(3)	NO. 105	(4)	NO. 42
(1)	NO. 106	(1)	NO. 43
(1)	NO. 107	(2)	NO. 45
(2)	NO. 108	(1)	NO. 46
(1)	NO. 11	(2)	NO. 47
(6)	NO. 112	(3)	NO. 49
(2)	NO. 113	(2)	NO. 5
(3)	NO. 114	(2)	NO. 50
(1)	NO. 116	(1)	NO. 51
(1)	NO. 117	(5)	NO. 52
(1)	NO. 12	(1)	NO. 53
(4)	NO. 14	(1)	NO. 55
(2)	NO. 15	(2)	NO. 56
(1)	NO. 16	(4)	NO. 57
(1)	NO. 17	(2)	NO. 58
(2)	NO. 18	(1)	NO. 6
(11)	NO. 19	(3)	NO. 61
(1)	NO. 2	(1)	NO. 62
(1)	NO. 20	(1)	NO. 64
(5)	NO. 22	(2)	NO. 65
(2)	NO. 24	(1)	NO. 66
(1)	NO. 26	(1)	NO. 68
(2)	NO. 27	(1)	NO. 69
(4)	NO. 28	(1)	NO. 7
(5)	NO. 3	(1)	NO. 70
(1)	NO. 30	(3)	NO. 71
(5)	NO. 31	(1)	NO. 72
(1)	NO. 32	(2)	NO. 73
(2)	NO. 33	(4)	NO. 74
(1)	NO. 34	(3)	NO. 75

(3) NO. 76
(2) NO. 77
(4) NO. 78
(2) NO. 79
(4) NO. 8
(2) NO. 80
(1) NO. 81
(2) NO. 82
(2) NO. 83
(2) NO. 84
(1) NO. 85
(3) NO. 86
(4) NO. 87
(2) NO. 88
(1) NO. 89
(4) NO. 9
(2) NO. 90
(4) NO. 91
(3) NO. 93
(4) NO. 94
(2) NO. 95
(4) NO. 98
(1) NO. 99
(2) NO.71
(3) NOISE
(6) NON-COOPERATION
(1) NON-COOPERATIVE
(10) NON-RESPONSE
(15) NON-STANDARD
(1) NOT
(2) NOVEMBER 79
(1) NOVEMBER 80
(4) NOVEMBER 82
(1) NOVEMBER 83
(2) NOVEMBER 84
(2) NOVEMBER 85
(1) NOVEMBER 86
(2) NOVEMBER 88
(22) NUMBERS
(2) OCTOBER 79
(1) OCTOBER 80
(4) OCTOBER 81
(1) OCTOBER 82
(5) OCTOBER 83
(1) OCTOBER 84
(3) OCTOBER 85
(2) OCTOBER 86
(3) OCTOBER 87
(6) OCTOBER 88
(14) PHONOLOGY
(33) PILOT
(28) PILOT DISTRACTION
(59) PILOT ERROR
(2) POSITION
(1) PRAGMATIC
(8) PRESUPPOSITION
(45) READBACK
(16) RECIPIENT NOT MONITORING
(4) RESTRICTIONS
(7) RUDE
(5) SEPTEMBER 79
(2) SEPTEMBER 80
(2) SEPTEMBER 81
(2) SEPTEMBER 82
(1) SEPTEMBER 83
(3) SEPTEMBER 85
(4) SEPTEMBER 86
(1) SEPTEMBER 87
(8) SPEED
(4) SQUAWK
(2) TCA
(2) TERMS
(3) TO
(13) TRAFFIC

(2) TRANSMITTER
(3) TWO
(2) UNCONTROLLED AIRPORT
(3) UNTIMELY TRANSMISSION
(5) VECTOR
(1) VOLUME
(3) WARNING SOUNDED

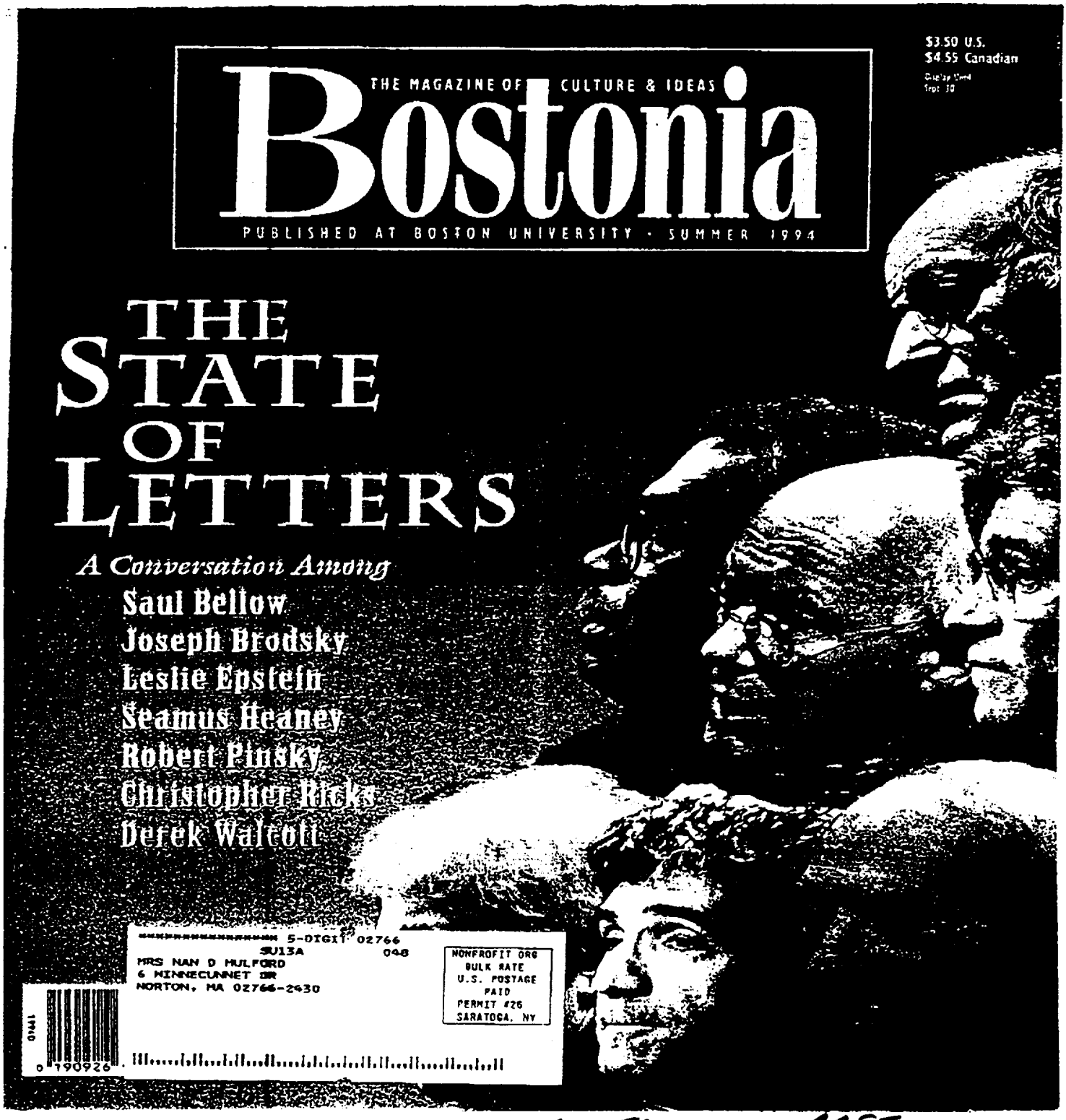
Post-it™ brand
Fax Transmittal Memo 7672

To Barrie Caldwell	No. of Pages 6	Today's Date 6/30/94	Time 1:00p.m.
Company NASA	From Kathy Conroy		
Location Moffett Field, CA	Company Stonehill College		
Fax # (415) 604-4646	Location	Dept. Charge 4100	
Comments	Fax # 508-238-9253	Telephone # 508-230-1069	
	Original Disposition ☐ Destroy ☐ Return ☐ Call for pickup		

NAG-2-564

Barrie:

Nan Mulford thought you might like to have a copy of this article written by Steven Cushing about his book "Fatal Words: Communication Clashes & Aircraft Crashes."



TM CASI

adoption committees, educational publishers, testing services, and a host of professional organizations. To those organizations Annenberg has added yet another: the Annenberg National Institute for School Reform.

It must be said that a handful of these professionals understand the problems they face and work hard to solve them. Theodore R. Sizer of Brown University, to whom Annenberg has given $50,000,000 for his Coalition of Essential Schools, is an example of an honest man speaking truth to the profession and the public in books like *Horace's Compromise: Redesigning the American High School*. More often, however, true reform has come from outsiders like Wendy Kopp, whose organization, Teach For America, offers a route into blighted urban classrooms for outstanding college graduates — though not without opposition. Schools of education and union officials know a threat to their certification monopoly when they see one.

Sizer has been given significant authority by Annenberg, but it is hard to imagine how one man, even with allies, can steal a victory from the champions of the status quo. The problem is that the culture of education — especially in academia — has a pronounced distaste for mavericks like Sizer and Kopp. It drives away those who don't play the funding and research game. Jonathan Kozol argues this compellingly in *Illiterate America*. In a typical example, he observes that when he set up the National Literacy Center, he "was accused of compromising academic interests by [his] failure to assign the first funds raised to salaries for doctoral assistants." Kozol would have happily supported a vital constellation of talent that yielded real work on the outside, but as a veteran of literacy wars he had no illusions about the incentives that drive educational professionalism.

Unfortunately, there is little evidence that Annenberg has heeded the lessons of Kozol or Kopp or considered the resounding critiques of the education profession—such as *Ed School*, by the University of California at Berkeley's Geraldine J. Clifford, to name just one. According to the December 18, 1993, *New York Times*, the Annenberg Foundation will make its funding decisions not only with Sizer's help but with the "assistance" of "education experts." Add to this the fact that Annenberg's largesse is contingent on matching grants, and it's easy to see how bland consensus will triumph over individual brilliance. A recent *Boston Globe* story explains that even an application for help through Sizer's Coalition of Essential Schools had first to be approved by the Boston City School Committee. Hurdles like this soon exhaust the idealism of even the most committed reformers. Steven F. Wilson, advocate of charter schools and author of *Reinventing the Schools: A Radical Plan for Boston*, is quoted in the *Globe* as warning of "a substantial danger that Mr. Annenberg will discover he has only fed a light lunch to the bureaucratic beast of the public school monopoly." To the extent that regnant powers are reinforced, Annenberg's $500,000,000 grant will actually harm the cause of school reform.

In short, if the Annenberg Foundation relies on the judgments of the experts, they'll only recommend themselves. Funding decisions will benefit the current professional culture, which has little stake in reform except as a rhetorical shield for holding conferences, publishing research monographs, producing doctors, masters, and bachelors of education, creating new institutes and journals, and hiring professors and curriculum specialists. A half billion dollars in *TV Guide* proceeds can do a lot of that, but it won't help teenagers who can't calculate percentages or read this page without difficulty.

I speak from experience about the fate of the philanthropic dollar, and I'm sure others can speak to similar effect. When I was a graduate student, for example, a big private funding effort for school reform in Michigan made headlines, thanks to Alfred Taubman, the mall developer, who plunked down $50,000,000. Shortly after the news broke I was encouraged to apply for one of the many new graduate fellowships that were certain to be had. Never mind that I had never taught in a public school, and was engaged in studies that had little discernible relation to the Taubman project. This was payday. If Annenberg's hundreds of millions enjoy a similar fate, he would have accomplished more by creating 50,000 scholarships of $10,000 each for private school tuition for the needy, or by dividing the whole check among Sizer, Kopp, Kozol, and a few other outliers.

I concede it's not nice to strike such dissonant chords when the mood is festive. Big-time philanthropy has a way of making people care and raising their hopes. Here's mine: If Walter Annenberg really has dropped a bomb, as media coverage would have it, we should hope it detonates, and on an appropriate target. The decline of public education can only be reversed if we begin with demolition work.

William Rice teaches in the Expository Writing Program at Harvard University and has written on academic issues for The Associated Writing Programs Chronicle, *the* Quarterly Review of Doublespeak, *and other journals.*

Fatal Words: Communication clashes and aircraft crashes

BY STEVEN CUSHING

The kind of misstatements and misunderstandings that we all make and experience in ordinary conversation could have fatal consequences in the communication between a pilot and an air-traffic controller. On March 27, 1977, the pilot of a KLM 747 radioed "We are now at take-off," as his plane began rolling down the runway in Tenerife, the Canary Islands (*Figure 1*). The air-traffic controller mistakenly took this statement to mean that the plane was at the take-off point, waiting for further instructions, and so did not warn the pilot that another plane, a Pan Am 747 that was not visible in the thick fog, was already on the runway. The resulting crash killed 583 people in what

Figure 1: Los Rodeos Airport, *Tenerife, The Canary Islands, March 27, 1977*

1705:44.6 *KLM 4805:* The KLM *four eight zero five* is now ready for takeoff and we are waiting for our ATC clearance (1705:50.77).

1705:53.41 *Tower:* KLM *eight seven zero five* you are cleared to the Papa Beacon, climb to and maintain flight level nine zero, right turn after takeoff, proceed with heading four zero until intercepting the three two five radial from Las Palmas VOR (1706:08.09).

1706:09.61 *KLM 4805:* Ah — Roger sir, we are cleared to the Papa Beacon, flight level nine zero until intercepting the three two five. *We are now at takeoff* (1706:17.79).

1706:18.19 *Tower:* O.K. . . . *Standby for takeoff,* I will call you (1706:21.79).

[Note: A squeal starts at 1706:19.39 and ends at 1706:22.06]

[*PAA:* And we're still taxiing down the runway. The Clipper one seven three six (1706:23.6)]

1706:21.92 *PAA 1736:* Clipper one seven three six (1706:23.39).

1706:25.47 *Tower:* Ah — Papa Alpha one seven three six report the runway clear (1706:28.89).

1706:29.59 *PAA 1736:* O.K., will report when we're clear (1706:30.69).

1706:61[sic].69 *Tower:* Thank you.

1706:50: *COLLISION:* KLM on takeoff run collides with PAA on ground.

is still the most destructive accident in aviation history.

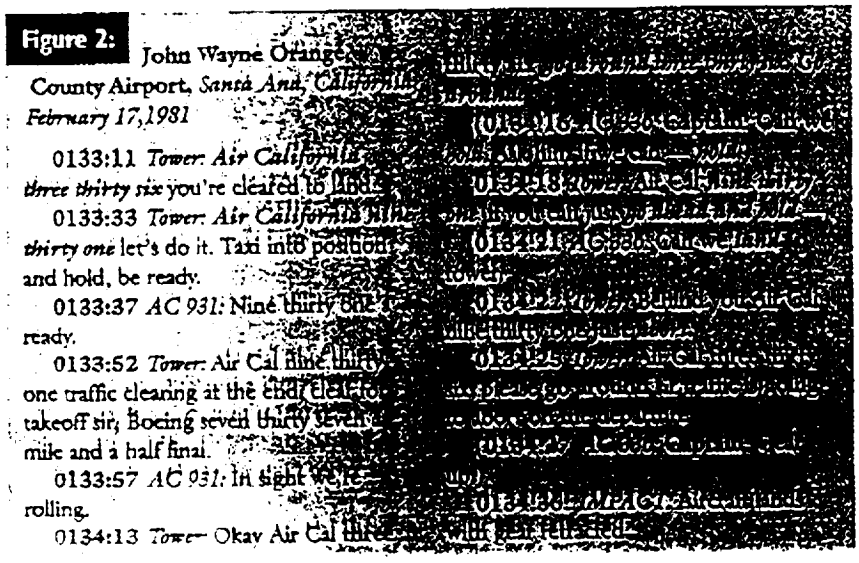
The KLM pilot's otherwise perplexing use of the very nonstandard phrase *at take-off,* rather than the more standard phrase *taking off,* can be explained as a subtle form of what linguists refer to as code-switching. Careful studies of bilingual and multilingual speakers have shown that, for reasons that are not well understood, they habitually switch back and forth from one of their languages to another in the course of a conversation. In the KLM pilot's case, the present progressive tense of a verb, which is expressed in English by the verb's *-ing* form, is expressed in Dutch by the equivalent of *at* plus the infinitive of the verb. For whatever reason, perhaps fatigue or stress, the Dutch pilot inadvertently switched into Dutch grammatical construction while keeping the English words. The Spanish-speaking controller had no clue that this was going on and so interpreted the *at* most naturally as a locative word indicating a place, the take-off point.

A different form of code-switching contributed to the accident that occurred at John Wayne Orange County Airport in Santa Ana, California, on February 17, 1981 (*Figure 2*). Air Cal 336 was cleared to land at the same time as Air Cal 931 was cleared to taxi into position for takeoff, but the controller decided that more time was needed between the two scheduled events and so told 336 to go around. For some reason, the 336 captain resisted this instruction by having his copilot radio for permission to continue landing, but he used the word *hold,* inadvertently switching from technical aviation jargon to ordinary English vernacular. (In aviation parlance, *hold* always means stop what you are now doing; in this case, that would mean the pilot would continue circling rather than attempt to land.) But in ordinary English *hold* can also mean to continue what you are now doing; in this case, to land. The controller's seemingly self-contradictory instruction to 931 to *go ahead and hold* at almost exactly the same time further exacerbated the situation, especially in view of the near-indistinguishability of the two aircrafts' identifying call signs and the consequent uncertainty as to just who was being addressed with that instruction. The resulting confusion led to thirty-four injuries, four of them serious, and the complete destruction of the aircraft when Air Cal 336 landed with its gear retracted, the pilot having finally decided to follow instructions to go around, with it too late actually to do so.

Uncertainty of reference, rather than of addressee, contributed substantially to an accident in the Florida Everglades on December 29, 1972 (*Figure 3*). The Eastern Airlines plane's pilot and crew had been preoccupied with a nose-gear problem, which they had told several controllers about during their trip. When the Miami International Airport approach controller noticed on radar that their elevation was declining, he radioed, "How are things comin' along up there?" and they responded, "OK." The crew was referring to the nose-gear problem, which, as it happens, they had just managed to fix, entirely unaware that there was any problem with elevation. However, the controller interpreted the *OK* as referring to the elevation problem, because that is what he had had in mind when he radioed the question. There were 101 deaths from the resulting crash.

In my book *Fatal Words: Communication Clashes and Aircraft Crashes* (University of Chicago Press, 1993), I discuss over 200 incidents, some of which, like these three, resulted in disastrous accidents — all of which easily could have

Figure 2: John Wayne Orange County Airport, *Santa Ana, California, February 17, 1981*



0133:11 *Tower: Air California nine three thirty six* you're cleared to land.

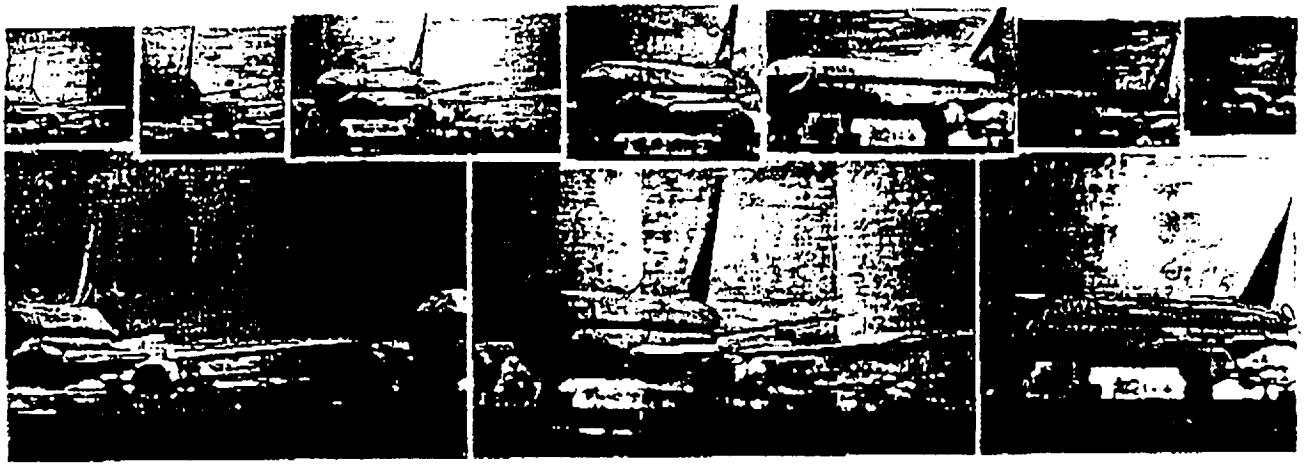
0133:33 *Tower: Air California nine thirty one* let's do it. Taxi into position and hold, be ready.

0133:37 *AC 931:* Nine thirty one ready.

0133:52 *Tower:* Air Cal nine thirty one traffic clearing at the end cleared for takeoff sir, Boeing seven thirty seven mile and a half final.

0133:57 *AC 931:* In sight we're rolling.

0134:13 *Tower:* Okay Air Cal three



been prevented if the communication circumstances in each case had been only slightly different. Some of these incidents were caused by mundane factors: distractions, fatigue, impatience, obstinacy, uncooperativeness, frivolousness, or crew conflict, and could have been prevented or ameliorated through better conditions, training, or discipline. But the more serious — and more interesting — communication problems are those that arise from inherent characteristics of language itself, from reference confusion, or from the inferences that are drawn in the course of linguistic communication.

Language is replete with ambiguity. The presence in a word or phrase of more than one possible meaning or interpretation, such as *at* in the Tenerife case or *hold* in the John Wayne case; and with *homophony*, different words that sound exactly or almost alike, such as *to* and *two*, which actually led to a fatal accident at a southeast Asian airport, or *left* and *west*; peculiarities of punctuation or intonation, such as *back on — the power* vs. *back — on the power*; and the complexity of speech acts, which correspond only in the most indirect ways to sentence or statement types — all these can wreak havoc in even the simplest of situations. For example, when a pilot misconstrued the phrase *traffic . . . level at 6000* to be an instruction for himself meaning [descend to and remain] *level at 6000* [because of traffic], rather than an assertion about his traffic meaning [the traffic is] *level at 6000*, as the controller intended.

Pronouns, such as *him* or *it*, or indefinite nouns such as *things* in the Everglades case, can have multiple references that are not easily distinguished in a conversation, and the use of a word like *anticipate* or of unfamiliar terminology can create expectations that have no factual basis. Extensive repetition of essentially the same instruction, such as *cleared to __ feet* or *expedite*, can lull a pilot into inattention. Similarly, overlapping number ranges that are shared by several aviation parameters (for example, *240* can be a flight level, a heading, or an air speed) inevitably breed confusions, requiring almost constant mutual or self-correction.

> But the more serious — and more interesting — communication problems are those that arise from inherent characteristics of language itself, from reference confusion, or from the inferences that are drawn in the course of linguistic communication.

Problems with radios, such as being tuned to the wrong frequency, can prevent an instruction from being heard even when the message itself is clear. A perfectly well-formed and meaningful message can still cause problems when, for some reason, it is not sent; is sent, but is not heard;

Figure 3: Miami International Airport, *Miami, Florida, December 29, 1972*

2334:05 *EAL 401:* Ah, tower this is Eastern, ah four zero one, it looks like we're gonna have to circle, ***we don't have a light on our nose gear yet.***

2334:14 *Tower:* Eastern four oh one heavy, roger, pull up, climb straight ahead to two thousand, go back to approach control, one twenty eight six.

2334:21 *EAL 401:* Okay, going up to two thousand, one twenty eight six.

2335:09 *EAL 401:* All right, ah, ***approach control.*** Eastern four zero one, we're right over the airport here and climbing to two thousand feet, in fact, we've just reached two thousand feet and ***we've got to get a green light on our nose gear.***

2336:27 *MIA App Con:* Eastern four oh one, turn left heading three zero zero.

2338:46 *EAL 401:* Eastern four oh one'll go ah, out west just a little further if we can here and, ah, see if we can get this light to come on here.

2341 *Second Officer within cockpit:* I can't see it, it's pitch dark and I throw the little light, I get, ah, nothing.

2341:40 *MIA App Con:* Eastern, ah, four oh one how are *things* comin' along out there?

2341:44 *EAL 401: Okay,* we'd like to turn around and come, come back in.

2341:47 *MIA App Con:* Eastern four oh one turn left heading one eight zero.

2342:12: *IMPACT:* Aircraft crashes into the Everglades.

is sent and heard, but still not understood; or is sent, heard, and understood, but not remembered by the listener.

One source of the problem is that the aviation protocol was not designed systematically, but is a hodgepodge that grew *ad hoc* as new inventions and innovations were introduced. However "re-engineering" the system; that is, redesigning it from scratch, would require closing the world down for several years as pilots and controllers try to forget what they have learned and get retrained in whatever new procedures and terminologies might be developed.

A more realistic approach would involve intensive efforts to teach pilots and controllers about the subtle nuances of language and communication and about how their own and other people's safety depend on their willingness to use language more mindfully. For example, the Aviation Safety Reporting System of NASA-Ames Research Center in Mountain View, California, the center that funded the study reported in *Fatal Words*, issues alerts on threats to aviation safety that it finds to be particularly prevalent. Some of them involve issues of language and communication. And the Centre de Linguistique Appliquee of the Universite de Franche-Comté in Besançon, France, develops linguistically sophisticated training materials for pilots and controllers and sponsors a triennial International Aviation English Forum, at which I presented some of the results reported in *Fatal Words* in 1991.

However, much more needs to be done in this area, especially in the United States, where English is taken for granted as a language that everyone is expected to speak in a standard way. In Europe, by contrast, where there are multiple languages, people have to take linguistic issues more seriously.

Another path that needs to be pursued is the development of appropriate communication tools. There are no sure fixes for emergency situations, which require split-second decisions by human beings, but technology can be used to reduce the number of emergency situations that arise. A close-to-ideal solution to at least some of these sorts of problems would be the development of an intelligent voice interface for aviation communication. Such a device would monitor communications and filter out potential linguistic confusions, checking with the speaker for clarification before conveying messages, and monitoring the aircraft's state, providing needed callouts automatically. Such a system would be valuable on-line as a safety device in real-time, but would be useful also as a training device, an aid to developing an awareness in both pilots and controllers of the kinds of linguistic constructions they ought to avoid, while conditioning them, to some extent, to do so.

Developing such a system would require extensive further research to solve many still open questions of scientific linguistics, such as the problem of speech recognition (how to extract a meaningful signal from an acoustic wave). This problem has become tractable technologically for individual words but still resists solution for more extended utterances.

There are also many unsolved problems of what linguists call *pragmatics*, or the ways in which *context* can effect the meaning of an utterance. For example, the sentence *I have some free time* means one thing during a discussion about one's work schedule, but means something quite different when driving up to a parking meter. With very little effort, people routinely distinguish such meanings in real conversations but exactly how they do that and how a device could duplicate this process remains to be discovered. The only certainty is that a workable intelligent voice interface is not likely to be developed for this or the next generation of aviation.

In the meantime, and in parallel with that research, it may be more fruitful to develop limited systems, in which a visual interface for processing a more restricted English-like language is used. A prototype version of such a system, the Aviation Interface Research (AIR) System, has been developed under my supervision by some of my graduate students at Boston University and is described in *Fatal Words*.

AIR uses a system of nested menus to send messages back and forth between two Macintosh computers, which simulate pilot and controller interfaces. When a message is entered from one of the user interfaces, a program called a *parser* checks that it is correctly formed with respect to the restricted English-like language that is used by the system. If it is acceptable, it is transmitted to the other interface, where it appears at the top of the screen; if necessary, an error message is returned to the sender instead. Menu screens are invoked by selecting icons, and messages are constructed by selecting buttons that contain actual words or phrases that are echoed at the bottom of the sending screen. As the system is currently set up, the selections are made by mouse. But they could just as well be made by touch-screen.

As it now stands, AIR serves mainly to illustrate the concept and demonstrate the feasibility of an error-resistant visual message-sending-and-receiving system for two-way air-ground pilot-controller communication. Work has begun on a second version that is envisioned as having further features that will improve on the current system in several ways. For example, it will be possible to provide bilingual screens, in English and in the user's own language, to enable the crew or controller to check the correctness of messages they want to send or to test their understanding of messages they receive. It will also be possible to have the system choose randomly from a set of synonymous alternative formulations of an instruction in order to preempt the semi-hypnotic boredom that is induced by repeatedly receiving instructions in exactly the same form.

Steven Cushing is an associate professor of computer science at Boston University. He was a summer faculty research fellow in human factors at NASA-Ames Research Center in Mountain View, California, in 1987 and 1988 and in flight management at NASA-Langley Research Center in Hampton, Virginia, in 1989.