



# **Eucalyptus: Integrating Natural Language Input with a Graphical User Interface**

KENNETH WAUCHOPE

*Navy Center for Applied Research in Artificial Intelligence  
Information Technology Division*

February 25, 1994



REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE February 25, 1994	3. REPORT TYPE AND DATES COVERED	
4. TITLE AND SUBTITLE Eucalyptus: Integrating Natural Language Input with a Graphical User Interface		5. FUNDING NUMBERS PE - 62234N TA - RS34-C74-000 WU - DN2573	
6. AUTHOR(S) Kenneth Wauchope			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Research Laboratory Washington, DC 20375-5320		8. PERFORMING ORGANIZATION REPORT NUMBER NRL/FR/5510--94-9711	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research Arlington, VA 22217-5660		10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.		12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words)  This report describes Eucalyptus, a natural language (NL) interface that has been integrated with the graphical user interface of the KOALAS Test Planning Tool, a simulated Naval air combat command system. The multimodal, multimedia interface handles both imperative commands and database queries (either typed or spoken into a microphone) while still allowing full use of the original graphical interface. In this way the precision and consistency of direct manipulation is balanced and augmented by the descriptive power and reduced redundancy of NL. The two input media used together yield such powerful interaction techniques as deixis (simultaneous speech and pointing) and the ability to use mouse clicks and verbal referring expressions interchangeably. Finally, the system's discourse handling capability allows abbreviated NL follow-ups (anaphora and ellipsis) to receive full interpretations based on the prior interaction context, whether verbal or graphical.			
14. SUBJECT TERMS Natural language processing Human-computer interface Speech recognition		15. NUMBER OF PAGES 44	16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL



## CONTENTS

INTRODUCTION .....	1
KOALAS AEW TEST PLANNING TOOL .....	2
WORKING ASSUMPTIONS .....	4
NL INTERFACE: BASIC GRAPHICAL EQUIVALENTS .....	6
Fighter Management .....	6
Threat Assessment .....	9
Check Boxes .....	11
Cycle Choice Items .....	12
Buttons .....	13
Radar Screen Mouse Operations .....	13
NL INTERFACE: BEYOND GRAPHICS .....	14
Reference .....	14
Query Facility .....	16
Discourse Capabilities .....	17
MULTIMEDIA INTEGRATION .....	20
NL Input to Dialogue Windows .....	20
Dialogue Windows for NL Command Completion .....	21
Discourse History for Graphical Operations .....	21
Deictic Reference .....	22
Threat Reference .....	24
EUCALYPTUS ARCHITECTURE .....	25
Speech Recognition Component .....	25
Natural Language Processor .....	26
Application-Specific Translator .....	30
IMPLEMENTATION .....	31
USER INTERACTION STUDY .....	31
RELATED WORK .....	33
CONCLUSIONS .....	34

**ACKNOWLEDGMENTS** ..... 35

**REFERENCES** ..... 36

**APPENDIX – Data Processing Examples** ..... 41

## EUCALYPTUS: INTEGRATING NATURAL LANGUAGE INPUT WITH A GRAPHICAL USER INTERFACE

### INTRODUCTION

As interactive software tools of the future become more and more complex and knowledge-intensive, the more the so-called "human-computer dialogue" may come to resemble true human-human dialogue in a number of ways. Foremost among these is discourse processing, the human ability to track and maintain continuity of topic, reference, and reasoning in extended sequences of natural language (NL) utterances, whether monologue or dialogue. At its simplest level, discourse processing allows speakers or writers to use a number of convenient shorthand devices to eliminate excessive repetition and wordiness, with the discourse context serving to fill in gaps and resolve underspecifics. At a higher level of complexity, discourse structure reflects the logical progression of thoughts in a discussion and signals shifts from one topic to another; while at the highest levels, it transitions to the less empirically explicable arts of rhetorical and compositional style.

In the early days of artificial intelligence research, many believed that once the technology had sufficiently advanced, NL would become the human-computer interface medium of choice. The success of the graphical user interface (GUI) in the intervening years now suggests that each of these interface media has relative advantages and disadvantages [1]. Most GUIs currently in use are based on the highly natural paradigm of direct manipulation, in which the user operates directly on visual analogs of domain and control entities (represented as menus, dials, iconic objects, etc.) using manual pointing, seizing, and selection gestures. Such interfaces can become tiring and repetitious in use, however, because of their inability to abstract over previous interactions and the necessity to select command arguments individually rather than denotationally. The NL paradigm, conversely, is based on the mutual reasoning ability and underlying contextual knowledge of the participating parties, so it provides many natural mechanisms for abstraction, such as logical connectives, quantification, and generalized description. But by the same token, NL runs the risk of being underspecific, sometimes resulting in vagueness, ambiguity, and misunderstanding.

Since discourse understanding depends heavily on contextual knowledge, inferencing ability, and implicit understanding of the principles of effective communication, a human-computer interface having these capabilities falls into a category that in recent years has come to be known as the Intelligent User Interface, or IUI [2]. The role of the IUI is to serve as an intelligent intermediary between user and application, translating cognitively high-level user inputs into lower-level operations that the application can execute, and customizing the form and modality of system output for maximum user comprehensibility. With the addition of discursive context tracking, each transaction between user and system can be minimally specific, deriving its full interpretation from the dialogue partners' implicit understanding of the current topic of the interchange. A recent trend in IUI technology is toward integrated interfaces in which both the graphical and NL paradigms play complementary roles, the strengths of the one supporting the weaknesses of the other [3,4], and it has also been suggested that discourse modeling techniques might be applicable to graphical interactions as well [5].

This report describes a natural language interface with limited discourse capability that the author has built and integrated into the GUI of a Navy system design and training demonstration, the KOALAS (Knowledgeable, Observation Analysis-Linked Advisory System) Test Planning Tool, as part of an exploratory development effort demonstrating intelligent multimedia interface technology applied to Navy needs. The interface, named Eucalyptus ("natural input to koalas"), handles both imperative commands and database queries, either spoken through a microphone or typed at the keyboard, while still allowing full use of the original GUI as an alternative input medium. The NL and graphical interfaces do not simply coexist but are mutually integrated, interacting in a number of ways that are discussed in detail.\*

Integrating NL input into an existing GUI (rather than developing a fully integrated multimedia interface from scratch) facilitated the development of a demonstration for the Navy of IUI technology applied to one of their own tools, and also provided an avenue for exploring which graphical interaction techniques are more compatible with NL and which less. Eucalyptus also builds on earlier work in NL interfaces by our group [7], giving us an opportunity to further exercise the natural language processing system we have been developing over the past several years, with a particular view toward extending its discourse handling capability.

The report is organized as follows. It begins with an introduction to the KOALAS demonstration tool and its graphical user interface. This is followed by a statement of the basic working concepts that were adopted for this project. The report then describes the Eucalyptus integrated NL interface, beginning with the basic NL equivalents for each graphical interaction technique, followed by a discussion of the special capabilities that NL offers, and concluding with a description of how NL and graphics interact in the integrated interface. It then describes the system architecture and details of the implementation, presents some confirmatory user data, gives an overview of related research, and concludes with a discussion of findings and observations.

## KOALAS AEW TEST PLANNING TOOL

KOALAS [8] is a process architecture for the design of a new generation of intelligent control systems in which the inductive intelligence of a human operator is optimally joined with the deductive capabilities of machine intelligence. The KOALAS AEW (Airborne Early Warning) Test Planning Tool [9] is an application built by the Los Alamos National Laboratory for the Naval Air Systems Command to serve as a concept demonstration for this design philosophy. The Test Planning Tool illustrates how a simulation-based software environment built around the KOALAS concept of intelligent control can be used in the design of complex military systems, in particular the planning of operational and design tests; the tool's authors also suggest other possible applications, notably tactical combat training.

The particular military system this tool simulates is a hypothetical command and control ( $C^2$ ) system for a Navy E2 AEW aircraft. The tool consists of two components, a scenario generation program and a simulation program, of which we are only concerned with the second. The simulation program runs on Sun SPARCstations and has a graphical interface written using the SunView toolkit. Central to the graphical display (Fig. 1) is a synthetic radar screen showing symbolic blips for each friendly aircraft and actual or suspected incoming raid in the simulated radar field. This screen is accompanied by four scrollable text display windows that are continuously updated to show current fighter and threat status, communications transactions, and sensor readings. The system also contains a rule-based tactical advice generator that monitors the state of the simulation and proposes courses of action in the form of simple imperative English sentences. The operator can either let the simulation run autonomously by putting the advice generator in "auto accept" mode (causing the system to automatically execute each internal tactical proposal), or inter-

\*An overview of an earlier version of the interface can be found in Ref. 6.

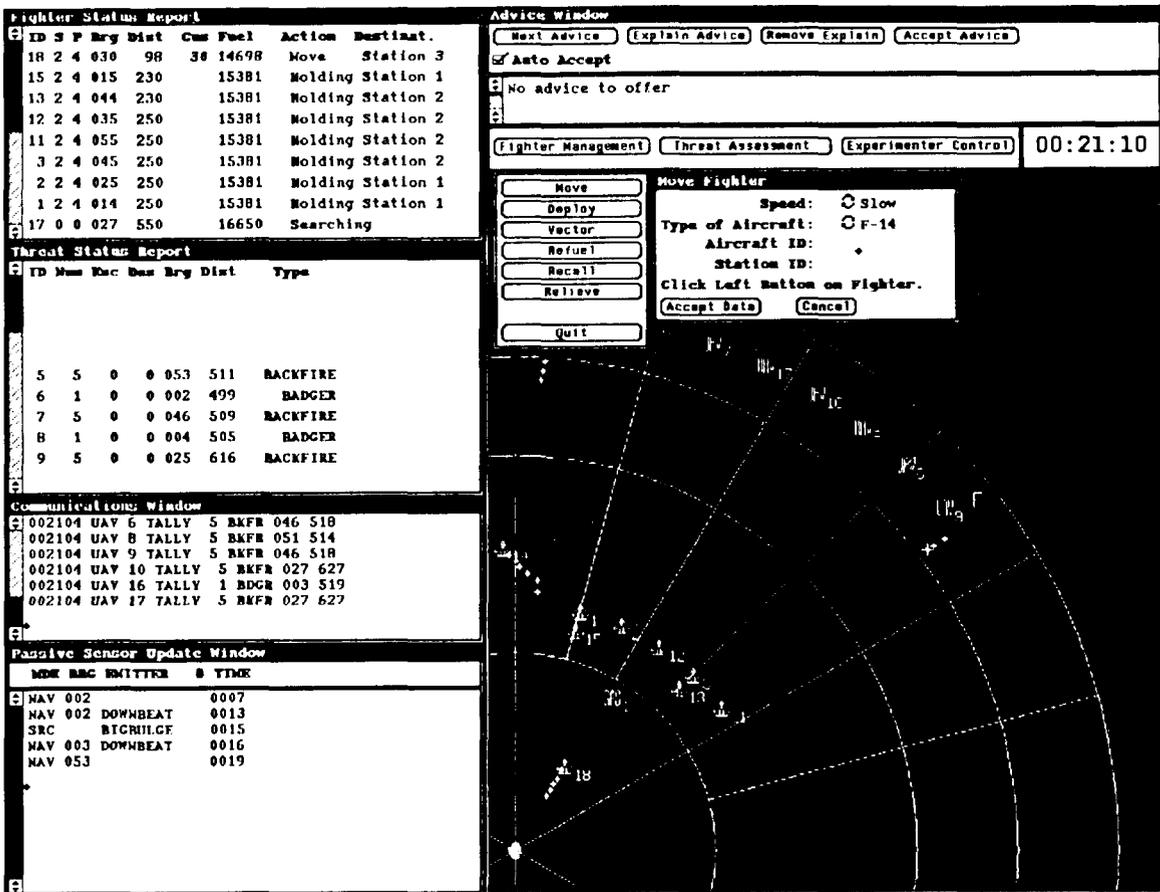


Fig. 1 – KOALAS graphical user interface

vene in the simulation at any time by accepting individual pieces of advice, manually issuing orders to fighters, or inserting hypotheses about the location and identity of possible incoming raids.

Operator intervention takes place through a number of different graphical interaction techniques. The operator selects, views, and accepts system advice using buttons in an Advice Window, and generates fighter orders and threat hypotheses by selecting the desired command from a menu and then entering data into a popup dialogue window that appears on the screen. This window prompts the operator to use the mouse to left-click on entities in the radar display, each click automatically updating the appropriate selection items and text fields in the window. A right click undoes the most recent data entry, clearing the corresponding text fields and backing up to the previous prompt. When all data have been entered, a final left click anywhere on the screen (or a click on the window's **Accept Data** button) grants final acceptance, executing the command; the window also provides a **Cancel** button to abort the proposed operation.

The operator can also modify various aspects of the simulation and graphical display. A middle mouse click on the radar screen re-centers it at those coordinates. Holding the left mouse button down over an aircraft or threat icon displays a popup window giving such pertinent information as ID, aircraft type, status, bearing, course, and range, and a right click on a UAV (Unmanned Airborne Vehicle, a drone surveillance aircraft) or hypothetical threat displays icons representing its associated threat sightings. Finally, an Experimenter Control panel provides selection items, buttons, and check boxes for such operations as

changing the simulation runtime speed, zooming the radar screen, generating visible trails behind aircraft and radar blips, suspending the simulation, and exiting the program.

The variety of graphical interaction techniques – buttons, menus, dialogue windows, on/off switches, choice items, text fields, and mousing on icons or screen positions – suggested that this interface might be a fruitful testbed for exploring the integration of NL and graphical input. At the same time, however, it was evident that the software had been designed as a concept demonstration rather than a production quality tool, and that its developers were well aware that the graphical interface they had provided was cursory at best, describing its design themselves as “superficial” [9]. This meant that care would have to be taken not to misinterpret specific design flaws in the KOALAS interface as indicative of graphical interfaces in general, while still correctly identifying those that do tend to exist in one form or another even in the majority of current state-of-the-art windowing systems.

The tool’s GUI also proved interesting from an NL standpoint in another way: the graphical interaction technique used for fighter management and threat assessment bears an intriguing resemblance to NL syntax and reference. The item labels on the command selection menus are all simple verbs, and in most cases the fields of the dialogue windows correspond to the verb’s linguistic arguments (verb complements). For example, to order a fighter to a new assignment station, the operator first selects the **Move** item from the Fighter Management menu, then clicks on the fighter, and then on the destination – a sequence of operations closely resembling the imperative sentence *Move this fighter here*. This use of the mouse to denote or identify an object in a command context is not direct manipulation as such but more properly direct graphical reference, and reference is a key ingredient of NL or NL-like interactions. A simulation-based system like this one, embodying as it does a domain-constrained world of dynamic entities engaged in various actions and possessing various attributes, provides a good arena for investigating reference-related issues.

This work is not intended to suggest that an actual military C<sup>2</sup> system such as the one the KOALAS tool simulates is an appropriate target for spoken NL input, particularly when new and experimental techniques like discourse tracking are being investigated. The problems of speaker independence, speech recognition accuracy, habitable coverage, and NL ambiguity have not yet been solved sufficiently to make truly flexible NL interfaces to mission-critical military systems feasible at this time. Since the KOALAS tool is not a prototype of such a system but rather an interactive tool that illustrates how such prototypes might be designed and tested, it closely resembles a decision aid or decision support system: the user makes decisions based on information obtained from the system and informs the system of those decisions so it can update its internal situational representation. As such noncritical desktop systems become more and more knowledge-intensive, the more useful a flexible and intelligent interface with spoken NL input capability like Eucalyptus can be.

## WORKING ASSUMPTIONS

The development and design of Eucalyptus was grounded in a number of initial working assumptions and decisions.

**Commercial Speech Recognition.** Eucalyptus uses a self-contained, commercial speech recognition front end and so does not involve any research in the speech recognition process itself.

**Natural Language Only.** The Eucalyptus user is to interact with the system using normative English utterances only. These can include elliptical and fragmentary forms, but only those that people might normally use with each other under similar circumstances. Eucalyptus is thus a spoken NL input system rather than simply a voice input system. This decision arises from our group’s history of taking a

strongly linguistic approach to NL processing, one of our current research goals being to study how human-human discourse and dialogue properties might be carried over to human-computer interactions. Our interests are thus not with interface modality or human-computer interaction techniques as such, but with the role natural language and discourse processing might have in more intelligent interfaces of the future.

**Input Only.** Aside from generating brief but helpful English sentence fragments in response to user queries, Eucalyptus ignores NL output issues such as generation or speech synthesis. The addition of spoken NL output to the KOALAS interface and the Eucalyptus query facility is currently under investigation by other researchers in the Interactive Systems group at the Naval Research Laboratory (NRL).

**Graphical Interface Unchanged.** In Eucalyptus the tool's original interface is left unchanged. The objective was not to improve the interaction techniques and displays of the GUI or to extend its functionality in any way, but simply to integrate NL whenever possible with those graphical constructs that were already provided and learn from the experience.

**No New Domain Functionality.** Eucalyptus does not give the NL interface any special domain-specific functionality not already available in some way via the graphical interface. This approach assumes that the existing GUI represents a model of all the operational functionality the system's designers intended for it to provide and support, and that the NL interface should adhere to the same (implicit) I/O specification. For example, although an NL interface to the tactical advice generator (i.e., a query-based explanation facility) might be a desirable addition to the system, it would require extending the underlying artificial intelligence capabilities of the KOALAS application itself, not just making the NL interface more intelligent.

**Datalink, not Voice Comms.** Since the KOALAS tool simulates a command and control station that would replace the E2's current radar screen and light-pen interface to the E2-F14 datalink, one might expect the role of NL input to be simulation of the voice communications channel between E2 controller and fighter pilot. Since Eucalyptus treats the tool as a decision support system rather than as a prototype C<sup>2</sup> system, however, it uses NL to interact not with the simulated aircraft (using standard Comm Brevity vocabulary and syntax) but with the computer system itself (i.e., the datalink) just as the graphical interface does.

**User-Initiated NL Interactions Only.** The original KOALAS graphical interface generates three types of NL-like textual outputs: fighter status entries ("Vector track 1"), tactical advice ("Deploy fighter 18 to station 3"), and advice explanations (fairly lengthy English-like rule traces). These are formatted ASCII strings generated by template filling, and not having NL semantic representations, Eucalyptus does not consider them part of the NL dialogue and so does not interact with them as such. Similarly, events initiated by the system (such as the proposal of a new hypothetical threat or the automatic execution of tactical advice) are not construed as implicit "utterances" to be included in the NL discourse context, although treating them as such is a possibility worth investigating.

**No Graphical Metalanguage.** Eucalyptus treats the graphical and NL interfaces as parallel and semantically equivalent, each accessing the same set of domain-functional operations but using different lexical/syntactic mechanisms to do so. For example, a GUI "sentence" to initiate aircraft trail display might consist of two "phrases": clicking the **Experimenter Control** button to bring up the control panel, and then clicking the panel's **Aircraft Trails** switch. The NL sentence with the same semantics is simply *Show aircraft trails*, not "Open the Experimenter Control panel and push the Aircraft Trails switch." The latter is an example of metalanguage, or language talking about language – in this case, using NL to refer explicitly to the vocabulary items (control panel, switch) and syntax (order of graphical operations) of the graphical

interface language. By disallowing such expressions, direct manipulation input devices are honored as just that, entities to be manipulated directly by the user, not indirectly by being talked about.

Similarly, a domain object represented by an icon on the KOALAS radar screen can be referenced in terms of the entity the icon denotes (e.g., *this fighter*) but not metalinguistically (“this icon”). Finally, a Eucalyptus command or query that results in output to the graphical display must be phrased in terms of the displayed entity rather than the display device – for example *Show the advice explanation*, not “Open the Advice Explanation window.” While the latter is something a computer expert might be inclined to say, the average user would probably not resort to that extra level of indirection.

The role of the NL interface in Eucalyptus is thus not to verbally operate the graphical interface controls, but to directly interact with the underlying application program just as the graphical interface does. An analogy would be a voice-activated home lighting system that can also be accessed by wall switches bearing labels like “Front Door Lights”. One can either flip the switch bearing that label or say “Turn on the front door lights,” which is a reference not to the switch itself (or to its label) but to the same entity controlled by the switch (and denoted by its label). This approach would disallow the utterance “Flip the front door light switch” since the switch is for direct manipulation only: we don’t need the physical interaction device to change state, just the entity it controls.

## NL INTERFACE: BASIC GRAPHICAL EQUIVALENTS

As described earlier, the KOALAS tool graphical interface provides a number of different interaction techniques for user input: menus, dialogue windows, check boxes, cycle items, buttons, and mouse clicks on the radar screen. Each poses different problems for the design of equivalent NL utterances that interact appropriately with the graphical devices in the integrated interface. We will look at each of these techniques in turn, beginning with the menu/dialogue window combination used for fighter management and threat assessment.

### Fighter Management

Six order types are available from the Fighter Management menu in the GUI: **Move** (order a deployed aircraft to a new assignment station), **Deploy** (order an aircraft from the carrier to an assignment station), **Vector** (order a fighter out to a hypothetical threat’s coordinates), **Refuel** (order an aircraft to a tanker for refueling), **Recall** (order an aircraft back to the carrier), and **Relieve** (order a backup fighter to relieve a forward fighter for refueling).

Selecting an order type brings up a dialogue window prompting the user to click on one or more objects on the radar screen to fill in the window’s data fields. Two order types take only a single argument: **Deploy** prompts for a destination (the system itself selects the fighter) and **Recall** prompts for an aircraft. Three order types are binary – **Move** (*aircraft, destination*), **Vector** (*fighter, threat*) and **Refuel** (*aircraft, tanker*) – and the remaining order type, **Relieve**, is ternary (*forward fighter, backup fighter, tanker*). The **Move** and **Deploy** windows also include a cycle item for choosing the aircraft speed, either **Slow** or **Fast**. After filling in all requested data, a final click on the window’s “Accept Data” button (or anywhere on the radar screen) executes the order and a click on the “Cancel” button aborts it.

The resemblance of the syntax of most of these operations to that of NL has already been noted. In particular, sentences of the type illustrated in (1) are natural equivalents to the first five graphical order types, with proper names constructed using the arguments’ ID numbers serving as the linguistic analog of direct graphical references.

- (1) *Deploy a fighter to station 5 [fast].*  
*Recall fighter 14.*  
*Move fighter 14 to station 5 [slow].*  
*Vector fighter 12 to track 3.*  
*Refuel fighter 14 at tanker 1.*

Eucalyptus also accepts the simple syntactic variant illustrated in (2). The grammar also permits fighters to be addressed directly (3), but this is discouraged since it runs contrary to the implicit syntax of the graphical interface, where aircraft are the subjects of orders rather than their direct recipients.

- (2) *Have a fighter deploy to station 5.*  
*Have fighter 14 return to the carrier.*
- (3) *Fighter 14, return to the carrier.*  
*Vector to track 3, fighter 12.*  
*Refuel at tanker 1.*

For each of these verbal inputs, after a second or two of processing time, Eucalyptus generates a dialogue window just like the one used for the corresponding graphical operation but with the verbal arguments and modifiers already filled in, allowing the user to check the input data for accuracy before having the order executed.\* Final acceptance or rejection can be issued either verbally (*Okay, That's fine, Never mind, Cancel that*) or graphically with a single mouse click.

Note in the first example of (1) that the appropriate NP in a verbal **Deploy** operation is the indefinite *a fighter*, acknowledging that the user does not care which one and allows the system to choose, as it does in the corresponding graphical operation (since in the KOALAS GUI fighter icons only become visible, and thus selectable, once they are airborne). Alternatively, in Eucalyptus the user can name a specific fighter known to be on the carrier (*Deploy fighter 18 to station 5*). If the named fighter is not on the carrier but has already been deployed, Eucalyptus ignores the order by producing no dialogue window for it.

Along the same lines, a fighter that has crashed continues to exist in the simulation (an entry for it remains in the Fighter Status window) but its icon disappears from the graphical display with the appropriate result that it can no longer be selected graphically as an argument to a fighter order. If its ID number is typed into the dialogue window's argument field, however, the system accepts the input but then simply disregards the order, so Eucalyptus behaves in just the same way when an order is verbally issued to a downed fighter.

Users of the KOALAS tool's graphical interface have complained about the number of low-level mechanical operations it takes to carry out an aircraft order. For example, as many as six mouse operations are needed to issue a fighter command: one to bring up the menu of order types, another to select the order, from one to three to specify the arguments and/or speed parameter to the dialogue window, and one more to confirm. While this could certainly be considered a design shortcoming in the KOALAS graphical interface that could be remedied by some less mouse-intensive graphical interaction technique, most users of commonly available GUI-based applications can probably think of similar situations they have experienced at one time or another where the number of graphical inputs demanded by the system became tiring and mechanical. Users of an NL interface must provide all the same parameters to complex interactions as in the graphical interface, but the ease of spoken NL certainly seems to provide a pleasant alternative to graphical input (and vice versa: one may tire of thinking and interacting verbally and choose to switch back to the hand-eye mode of graphical interaction).

---

\*The motivations behind this design decision are discussed later in the section "Multimedia Integration."

Another problem that arises occasionally in window-based graphical interfaces is visibility obscuration. For example, during a **Vector** operation in the KOALAS GUI, the popup menu and dialogue window can obscure the destination threat's icon, making it impossible to select with the mouse. (Indeed, the graphics obscure that particular region of the radar display so much of the time that adversaries playing scenarios against each other will sometimes have their enemy raids attack from that bearing so that they can penetrate their opponent's defenses unnoticed!) While again these should properly be considered design shortcomings in the KOALAS interface, the fundamental point remains that in any window-based graphical interface where much or all of the visual "real estate" of the screen is already in use, newly created windows will necessarily obscure some component or another of the GUI and possibly require resizing, moving, hiding, closing, etc. if the obscured portion of the display needs to be accessed. Since arguments are provided verbally in a NL interface, however, visibility is never an issue (e.g., one can refer to a threat even if its icon is temporarily obscured). The down side to verbal reference is that greater demands are made on the user's memory, requiring that s/he be able to name or describe the intended object, which is not the case with direct manipulation.

### *Relieve*

Unlike the five order types just discussed, **Relieve's** three participants do not correspond in number or canonical syntactic order to the thematic role arguments of the corresponding English verb, which is strictly binary (X relieves Y). This order type can thus be thought of as a composite of two simpler actions, **Relieve** and **Refuel**: X relieves Y so that Y can refuel at Z. The only way this can be expressed compactly in English is by using an embedded infinitival clause whose zeroed subject is interpreted as coreferent with *relieve's* direct object (4).

(4) *Have fighter 14 relieve fighter 12 to refuel at tanker 1.*

Sentence (4) illustrates how the verb *relieve's* thematic grid gives primary attention to the backup (*fighter 14*), assigning it the syntactic role of subject. In the graphical interface, however, the forward fighter (the one being relieved) is referenced first and the backup fighter second. This is undoubtedly because the forward fighter, since he needs refueling, is considered the primary focus of interest, with the backup playing a supporting role and the tanker (the third argument) the most passive role of all. A more dubious English phrasing (5) is required to capture this ordering.

(5) *Relieve fighter 12 with fighter 14 to refuel at tanker 1.*

Finally, although Eucalyptus can handle both (4) and (5), to handle more complex forms such as (6) would require the addition of ad hoc techniques for coalescing two simple sentences into a single KOALAS command.

(6) *Have fighter 14 relieve fighter 12 so he can refuel at tanker 1.  
Have fighter 12 refuel at tanker 1 and have fighter 14 relieve him.*

Since the **Relieve** order conflicts with the verb *relieve* in argument structure, better integration between graphics and NL would be achieved by changing the graphical **Relieve** construct to a binary operator and use it in combination with the existing **Refuel** operator to achieve **Relieve's** current operational definition. A completed **Relieve** order would then cause a **Refuel** dialogue window to automatically come up with the relieved fighter already filled in. This approach would allow Eucalyptus to handle all of (4-6) with no ad hoc fixes at all.

Should the order of arguments in the graphical **Relieve** operation then be reversed to match the more canonical English usage of (4)? That would probably be of little value since the less common usage

of (5) is also valid. It is worth noting, however, that in the graphical interaction the user must take time to stop and interpret the dialogue window slot labels (“Forward Fighter” and “Backup Fighter”) to determine which one represents the fighter needing refueling and which the relief. This is not the case with verbalizations like (4), where our inherent knowledge about the verb *relieve* automatically informs us which argument is which. While it might be possible to revise the slot labels to increase their recognizability, the fact remains that NL syntax and semantics provide that role information automatically whereas the order of arguments in a dialogue window does not.

### **Underspecific inputs**

Arguments can be omitted from Eucalyptus NL inputs just so long as the resulting utterance is still grammatical. Example (7) shows some acceptable Fighter Management orders. In each case a dialogue window comes up with only the specified data filled in, prompting the user to enter the next piece of missing information using the mouse. The examples in (8), however, are ungrammatical (the stigma “\*” denotes a syntactically ill-formed phrase) and are rejected by the NL parser with the message “Sorry, don’t understand.”

- (7) *Relieve fighter 14.*  
*Have fighter 14 refuel.*  
*Deploy a fighter.*
- (8) \**Recall to the aircraft carrier.*  
\**Have fighter 1 relieve.*

There are a number of reasons why users might issue underspecific orders such as (7) (see Ref. 10 for a related discussion). First, they might have forgotten that an order requires a particular argument, such as the **Deploy** order requiring that a destination sector be specified. Second, they might prefer to initiate the order verbally and then fill in certain arguments graphically. For example, it is more convenient for the user to simply say *Deploy a fighter* and then specify the destination with a quick mouse click on the radar screen than to have to mentally recall the sector ID number associated with a particular screen location and include it in the verbal input (*Deploy a fighter to sector 5*). Finally, the user might assume that the system will automatically fill in certain missing arguments: for example, if there is only one fuel tanker in the simulation scenario, the user might assume that *Refuel fighter 12* is sufficient input and that the system will naturally fill in the ID of the only available tanker. Since the original graphical interface does not do this, however, neither does Eucalyptus, to preserve semantic equivalence between the two interface media.

Note that the user cannot bring up an empty dialogue window by simply uttering the verb *Deploy*, because the NL understanding component interprets that utterance as an elliptical imperative telling the addressee (in this case the KOALAS tool itself) to leave the carrier, which violates the system’s semantics. If the user had previously been addressing a fighter, however, that fighter becomes the implicit subject of the imperative and a dialogue window would come up with the fighter’s ID filled in. This illustrates how Eucalyptus is not a simple speech input system where one can just utter the names of widget labels (i.e., the menu item label **Deploy**), but a spoken NL system that obeys at least some of the rules of dialogue discourse.

### **Threat Assessment**

The Threat Assessment menu provides four action types: **Make** (create a hypothetical threat), **Delete** (delete a threat), **Alter** (change a threat’s attributes), and **Lock on Track** (lock a threat onto a radar blip). As with the Fighter Management operations, selecting an action type brings up a dialogue window whose text fields can be filled automatically by clicking on the radar screen in response to system prompts.

**Delete** and **Lock on Track** are unary operators, prompting only for the threat to be deleted or locked onto a radar blip (the blip must be at the same coordinates as the threat and so is not needed as a second argument). **Make** is also unary, prompting for the radar screen location where the new threat is to be positioned. **Alter** is binary, prompting for a threat and a new position. Except for *alter*, the corresponding English verbs have the same argument structure as the graphical operations, giving us the basic NL equivalents of (9).

- (9) *Make a threat here.*  
*Delete threat 1.*  
*Lock threat 1 on track.*

As with Fighter Management orders, these fully specific utterances bring up completely data-filled dialogue windows ready for final confirmation, whereas the underspecific utterance *Make a threat* brings up a dialogue window prompting for the required position argument.

### **Alter**

Although **Alter** has the same argument structure as the **Move** fighter management operation, *Move fighter 1 here* is grammatical but *\*Alter fighter 1 here* is not. The reason the **ALTER** operation is so named is that the dialogue window can be used to change other threat attributes besides position. When the user selects a threat, the threat's current attributes (type of threat, number of aircraft, bearing, distance, speed and course) are filled in automatically as defaults. A new aircraft type and speed can then be manually chosen from cycle items in the window, and the **Number of Aircraft** text field can be edited with the keyboard. If the user responds to the "Click Left Button for Position" prompt, the threat's **Bearing** and **Distance** fields are automatically updated to reflect the new position, and its **Course** field is updated to reflect the new default heading (towards the carrier). A new course can then be assigned by clicking the right mouse button, which causes the prompt "Click Left Button for Course" to appear, and the position clicked on the radar screen then defines the threat's new course.

Like the **Relieve** fighter management order, more complex phrasings such as subordinate clauses and genitival constructions are required to verbalize both the basic **Alter** operation (10) as well as the other ways that a hypothetical threat's characteristics can be modified (11).<sup>\*</sup> These can also be combined in various ways as illustrated in (12).

- (10) *Alter threat 1 to have this position.*  
*Alter threat 1 to be over here.*  
*Alter threat 1 to be located here.*  
*Alter threat 1's position to here.*
- (11) *Alter threat 1 to a Badger.*  
*Alter threat 1 to be 5 Badgers.*  
*Alter threat 1's bearing/distance/course to 300.*  
*Alter threat 1 to have this heading.*  
*Alter threat 1 to be heading here.*  
*Alter threat 1 to be moving fast.*
- (12) *Alter threat 1 to five Badgers located here moving here fast.*  
*Alter threat 1 to be heading here from this position.*

---

<sup>\*</sup>The syntax of the graphical operation does not allow the user to select a new heading for a threat without first (re)selecting the threat's position as well. Since the NL syntax does not impose this constraint, this suggests that the graphical operation's syntax should be revised accordingly.

All the inputs above are considered fully specific and advance the dialogue window to its final "Click Left Button to Accept" prompt, since the threat's current position remains the default if no new position is specified. The underspecific (13) and (14), however, leave the window in the intermediate "Click Left Button for Position" and "Click Left Button for Course" states, respectively.

(13) *Alter threat 1.*

*Alter threat 1's position/bearing/distance.*

(14) *Alter threat 1's heading/course.*

Finally, since the **Make** command dialogue window's structure and behavior is virtually identical to **Alter**, the various NL capabilities outlined above transfer easily to the verb *make* as well (15).

(15) *Make 2 Badgers located here heading here fast.*

*Make a pathfinder with a course of 300 at this position.*

## Check Boxes

Check boxes (graphical on/off switches) toggle between on and off with each mouse click, displaying a check mark when in the on state. The KOALAS tool GUI has seven check boxes, labeled **Real Threats**, **Ground Truth**, **Aircraft Trails**, **Radar Trails**, **Diagnostics**, **Data Collection**, and **Auto Accept**. The first four control the display of information on the radar screen; the fifth adds rule-diagnostic information to the system's tactical advice explanations; the sixth activates on-line data collection; and the last switches the system into automatic advice acceptance mode, defined earlier.

Natural language equivalents for these were determined by first selecting a verb to represent the underlying functionality of each switch. For the first five switches (all labeled with just noun phrases), the implicit operational verb chosen is *show* or *display*. The operational verb for the sixth is just the denominalization *collect* of the switch's label, and the label of the seventh switch is already a verb (albeit fabricated), *auto accept*.

All these verbs are strictly transitive, so by adding the label's noun phrase (or, in the case of **Auto Accept**, the noun *advice*) as direct object we get the simplest NL equivalents, (16). Nominalizations of each of these, introduced by the auxiliary *do* (17), are also accepted, and an explicit *don't* auxiliary transforms each of (16-17) into its complementary operation (18). Alternatively, each of (16-17) can be made progressive and introduced by the verbs *start* or *stop* to switch on or off, respectively (19). Finally, the switch label can simply be prefixed by the generic verb *turn on/off* (20). In each case, the NL command both changes the internal state of the KOALAS process and generates the appropriate visual feedback in the graphical interface (i.e., adds or subtracts the check mark).

(16) *Show real threats (ground truth...).*

*Collect data.*

*Auto accept advice.*

(17) *Do real threat display.*

*Do data collection.*

*Do auto acceptance.*

(18) *Don't show real threats.*

*Don't collect data.*

*Don't auto accept advice.*

*Don't do real threat display (data collection auto acceptance)*

- (19) *Start showing real threats.*  
*Stop collecting data.*  
*Start auto accepting advice.*  
*Stop doing real threat display (data collection, auto acceptance).*
- (20) *Turn on real threats.*  
*Turn off data collection.*  
*Turn on auto accept.*

Whereas a mouse click just toggles the current state of the switch to its complement, each NL command expressly declares the goal state of the operation. If the switch is already in that state, Eucalyptus informs the user of the fact with the messages “Already on” or “Already off” rather than simply perform no operation. *Start*, *stop*, *turn on*, and *turn off* are modeled as presuppositional, that is, both they and their negations are interpreted as implying the truth of their argument. Thus, if data collection is currently off, Eucalyptus will respond “Already off” in all the cases illustrated in (21).

- (21) *Turn off data collection.*  
*Don't turn off data collection.*  
*Stop doing data collection.*  
*Don't stop doing data collection.*

### Cycle Choice Items

The **Experimenter Control** panel contains two cycle choice items, **Zoom** and **Speed**, which respectively alter the radar screen magnification and the ratio of simulation time to real time. A range of verbs were chosen to model these choice items: *zoom (in/out)*, *make*, *set*, *slow down/speed up*, *change*, and *increase/decrease*, illustrated in (22). As with the check boxes, the NL command both changes the state of the corresponding KOALAS variable and updates the visual display to reflect the new value.

- (22) *Zoom out.*  
*Zoom in to a magnification of 4.*  
*Make the screen magnification 3/2.*  
*Set the magnification to 2/3.*  
*Slow down the simulation.*  
*Speed the simulation up to 2 times real time.*  
*Make the simulation speed real time.*  
*{ Change | Increase | Decrease } the speed to 20.*  
*{ Change | Increase | Decrease } the magnification.*

In the graphical interface, clicking repeatedly on a cycle item increments its value upward to the maximum value and then cycles back to the lowest value, so goal-less verbal inputs like *Zoom*, *Change the screen magnification*, and *Change the simulation speed* do the same. Most other phrasings, however, have presuppositions that must be satisfied before execution. For example, *Zoom in* presupposes that the current magnification is less than the maximum value, and *Zoom in to 1* presupposes that it is currently less than 1. If such presuppositions are not met, Eucalyptus responds “Less/Greater than that now,” “At that value now,” or “At maximum/minimum value now.” The graphical interface does not protest if the user resets a parameter to its current value,\* however, so the verbs *zoom* and *make* are not treated as presuppositional, making sentences like *Zoom to 4* and *Make the simulation speed 2* acceptable regardless of the parameters’ current values.

\*Resetting the screen magnification to its current value is one way of redrawing the screen to clear away graphical glitches.

## Buttons

The six domain-functional buttons in the interface are labeled **Next Advice**, **Explain Advice**, **Remove Explain**, **Accept Advice**, **Suspend/Resume**, and **Exit Simulation**.<sup>\*</sup> The labels of all of these but the first (which Eucalyptus assigns the implicit verb *show*) include a verb representing the underlying command. As with the check box commands, Eucalyptus requires that all the button-equivalent verbs be used transitively, as in (23).

- (23) *Show the next advice.*  
*Explain the [current] advice.*  
*Remove the [advice] explanation.*  
*Accept the [current] advice.*  
*{ Suspend / Resume } the simulation.*  
*Exit [from] the simulation.*

In addition to these six buttons, each fighter management/threat assessment dialogue window provides two buttons labeled **Accept Data** and **Cancel** for confirming or cancelling the currently proposed order. As before, Eucalyptus accepts transitive sentences (*Accept the data*, *Cancel the operation*) for these two operations. In addition, since the dialogue window interaction takes the form of a “subdialogue” embedded in the top-level user-machine dialogue, formulaic elliptical inputs like *Okay* and *Never mind* are also accepted.

## Radar Screen Mouse Operations

As mentioned earlier, mouse clicks on the radar screen have predefined functionality in the KOALAS graphical user interface: a left click on an aircraft icon displays its position information, a middle click recenters the radar screen, and a right click on an aircraft shows any threat sightings (tallies) associated with it. We will consider each of these in turn.

The description of mouse-left in the user’s manual is “Display information about an aircraft’s position.” A verbal input along those lines seems too wordy to make NL an acceptable alternative compared to the simplicity of a single mouse click, so Eucalyptus instead interprets mouse-left simply as the verbs *describe* or *tell*, accepting sentences such as (24).

- (24) *Describe that fighter.*  
*Tell me about threat 1.*

Mouse-generated aircraft position popup windows automatically vanish once the mouse button is released, a form of direct manipulation in which the user in effect “holds up” the display window with the mouse for viewing and then “drops” it when through examining it, giving mouse-left-down the semantics “describe” and mouse-left-up the semantics “remove description.” Correspondingly, a window generated by a verbal *describe* remains on the screen until explicitly dismissed (*Okay*, *Remove the description*). This parallels an undocumented “pushpin” behavior already present in the graphical interface: moving the mouse over the window before releasing the button leaves the window stuck to the screen until dismissed by a second left click on an empty portion of the screen (if more than one position window is pinned, only the most recent one is dismissed – the others must be killed from their titlebar menus). As a result, Eucalyptus allows verbally generated windows to be dismissed by a second left click as well.

---

<sup>\*</sup>Four other buttons in the graphical interface (**Fighter Management**, **Threat Assessment**, **Experimenter Control**, and **Quit**) serve only to bring up or dismiss other graphical entities like menus, popup windows, or control panels. As discussed earlier, Eucalyptus does not accept NL input for such “metalinguistic” graphical operations.

Mouse-middle simply translates in Eucalyptus to commands of the form *Center the (radar) screen here*, where “here” will be discussed later in the section “Deictic Reference.” Mouse-right behaves in somewhat similar fashion to mouse-left, the icons representing threat sightings remaining visible on the screen until dismissed by a second right mouse click at an empty screen position, but in this case the accumulated displays of successive tallies are all erased at once. Example (25) illustrates the corresponding NL commands.

(25) *Show the [threat] ( tally / sightings ) for UAV 16.*  
*Remove the [threat] ( tallies / sightings ).*

## NL INTERFACE: BEYOND GRAPHICS

So far in this report we have looked at simple NL sentences that represent direct translations of individual GUI commands, thus constituting only a change of interface medium. The goal in Eucalyptus was not to simply produce a multimedia interface, but to exploit the additional expressive power that NL provides above and beyond what is possible in the graphical interface, while still keeping the two fully integrated (multimedia integration is discussed in the next major section). Three features of NL that most distinguish it from direct manipulation are its capability for paraphrase (synonymy and variant syntax), set-theoretic and logical operations (particularly denotative reference), and context maintenance (discourse tracking). Since our group had already explored the first of these in the InterFIS project [7], the remaining two were chosen for investigation in Eucalyptus and are discussed in the sections that follow.

### Reference

NL reference allows a user to specify cognitively significant sets of objects descriptively through the use of attributive and relational predicates, quantifiers, and logical operators. This is considerably different from the direct manipulation paradigm, in which operations are applied to individually selected objects. Descriptive denotation also makes possible more efficient information access in the form of NL queries, the user asking about the existence or identity of cognitively interesting sets of objects or the values of their attributes; the Eucalyptus query facility will be described shortly.

NL referring expressions in Eucalyptus can be pronouns (*it, he, them*), headless quantifiers (*one, any*), locative adverbs (*here, there*), proper names (*fighter 1*), or noun phrases: indefinite (*an F14, any of the searching UAVs, no fighter*), definite (*that tanker, the fighters moving to station 2*), or universal (*all the fighters on the carrier, every UAV*). NP premodifiers can be nominal (*radar trails*), adjectival (*hypothetical threat*), verbal (*deployed fighter, moving F14*), or genitival (*fighter 1's missiles*); postmodifiers are either prepositional phrases (*an aircraft on the carrier, the bearing of fighter 1*) or relative clauses (*a threat being vectored to, the fighters [that are] low on fuel*). NPs can be conjoined conjunctively (*the tanker and all the fighters, fighter 14 and tanker 1*) or disjunctively (*either an F14 or one of the UAVs, UAVs 3, 4, or 6*).<sup>\*</sup> Indefinite NPs are interpreted intensionally only in the context of the threat management commands **Make** and **Alter** (*Create a Badger here, Change threat 1 to a Badger*) and are interpreted as extensional references in all other cases.

Nominal and adjectival NP premodifiers are interpreted as object specifiers, e.g., *radar trails* and *aircraft trails* refer to two different domain entities that together could be referenced as *all the trails*. All other premodifiers and postmodifiers are interpreted as relational constraints to be checked against the KOALAS internal database when computing the referent set, e.g., *the moving fighters* is the subset of fight-

---

<sup>\*</sup>Due to constraints in the logical form generator (described later), conjoinings that combine conjunction and disjunction (*either all the fighters or none of the UAVs*) are not supported.

ers that are moving somewhere, equivalent to *the fighters moving to a tanker or a station*. Postmodifiers in particular are fully iterative and recursive (26).

(26) *a threat located here consisting of 5 Badgers  
every fighter moving to the station that fighter 2 is holding*

Genitives like *fighter 1's missiles* are interpreted, like *the missiles of fighter 1* or *the missiles fighter 1 has*, as references to the values of aircraft attributes like bearing, distance, course, speed, pounds of fuel, and number of missiles. Such NPs are not dereferenced extensionally but are treated as intensional formulae and evaluated as function calls during execution of the logical form. For example, *Does fighter 1 have a bearing of 30?* generates a logical form along the lines of

```
(TELLIF (EQUAL :THEME
            (ATTRIBUTE-VALUE :PATIENT 'FIGHTER-1 :THEME 'BEARING)
            :GOAL 30))
```

where the ATTRIBUTE-VALUE function call returns the current value of fighter 1's bearing (obtained by consulting the KOALAS tool's internal state), which is then tested against the goal value.

The domain objects that serve as linguistic referents are generated in three phases. Scenario-independent entities (KOALAS, the operator, the aircraft carrier, fighter assignment stations, display/simulation control parameters, and generic aircraft attributes) are defined in an external Eucalyptus knowledge base loaded at system startup. Next, Eucalyptus creates all the scenario-dependent objects (the friendly aircraft and actual threats) at the beginning of the run using information contained in the internal data structures that KOALAS generates from the input scenario. Finally, Eucalyptus represents ephemeral objects (the hypothetical threats) by temporary objects created by consulting the KOALAS internal database on an as-referenced basis, since the KOALAS entities those objects represent are created and destroyed dynamically throughout the simulation run. Eucalyptus detects and reports reference resolution failures (specifier and number mismatches) with the messages "{No | Only one | More than one} such <object type> exists." Processing of the command or query ceases at that point unless other parses are available, in which case the system looks for an alternative interpretation that is free of reference errors.

It was noted earlier that one feature of the KOALAS graphical interface is to only permit fighter management and threat assessment operations to be performed one at a time on individual arguments, and that no such operation can be initiated while another is still pending. To parallel this behavior in Eucalyptus, verbal fighter or threat commands containing plural arguments (e.g., *Lock threats 1 and 2 on track*) bring up each popup confirmation window one at a time, waiting for the current one to be dismissed before displaying the next one.\* Since the user may wish to cancel the entire series of dialogue windows that can result from such an utterance, verbal cancellation expressions like *Never mind*, *Forget it*, and *Cancel that* cancel the entire verbal order, not just the current individual window. Eucalyptus does not detect application-specific pragmatic errors involving plural arguments, as in *Move fighter 1 to stations 4 and 5* (a fighter can only move to one station at a time), but simply generates two dialogue windows in sequence, the latter (if confirmed) overturning the effect of the former (if confirmed).

The variety of referring expressions available in Eucalyptus serves two main functions. One is anaphoric reference, to be discussed in a following section. The other is descriptive reference, which allows the user to denote one or a set of objects by their attributes and relationships, relegating the work of identifying the particular individuals to the system itself. Example (27) is a command that a KOALAS user might have in mind and could execute directly as NL input to Eucalyptus, letting the system choose a fighter from the appropriate set and identify the particular threat. Using the graphical interface, the operator

\*The same behavior occurs in conjoined imperatives, e.g., *Create a threat here and lock it on track*.

would have to first interact with the system in various ways to obtain the ID numbers of the entities in question, arbitrarily choose one of the fighters, find the selected fighter and threat on the radar screen, and finally click on each of them with the mouse.

(27) *I want to vector one of the fighters holding station 1 out to the same threat fighter 2 is vectoring to.*

## Query Facility

So far we have only discussed imperative NL sentences for issuing commands to the system. Eucalyptus also includes a question-answering facility for querying the internal state of the simulation. The query component handles yes/no questions (*Are any fighters moving?*), Wh-questions (*Who is moving?*) and Wh-Noun questions (*Which fighters are moving?*). Wh-questions are restricted to asking *who*, *what*, *where*, and *how many (much)* since neither KOALAS nor the NL processor represents time history (when) or metaknowledge (why, how).

In keeping with the decision to treat the existing graphical interface as embodying an implicit specification of the system's intended I/O accessibility, Eucalyptus only allows the user to query for that system information also available via the graphical displays. The graphical interface presents simulation state information in three ways. First, fighter and threat status is continuously displayed in tabular form in text output windows which the user can scroll through in search of desired information. For each fighter in the simulation, the Fighter Status window displays its **ID**, number of **Sidewinder** missiles, number of **Phoenix** missiles, **Bearing**, **Distance**, **Course**, pounds of **Fuel**, **Action**, and **Destination**. For each suspected enemy raid that is currently hypothesized, the Threat Status window shows its **ID**, **Number** of aircraft, number **Escaped**, number **Destroyed**, **Bearing**, **Distance**, and **Type**.

Second, holding the left mouse down on a fighter or threat icon on the radar screen brings up a so-called "blip popup" window displaying aircraft **ID**, **Type**, **Bearing**, **Range**, **Status** (fighters only), and **Number** and **Course** (threats only). **Status** is the same as **Action + Destination** in the Fighter Status window, consisting of simple text strings like "Vector Track 3," "On Carrier," and "Holding Station 1." Finally, fighter icons change color to indicate certain state changes: normally colored cyan, the icons change to yellow when vectoring to a threat and green when out of weapons.

Eucalyptus uses a traditional database query system approach, translating each question into a performative-marked quantified logic expression which, when evaluated, accesses the KOALAS internal data structures and retrieves a set of conforming domain objects. The performative then uses this result to compose a brief but informative answer appropriate to the way the question was formulated, and displays this answer as a text string. For example, a yes/no question involving a universal NP (*Are all the threats being vectored to?*) just answers "Yes" if no exceptions exist, but does not simply answer "No" otherwise – it goes on to enumerate either the positive or negative instances (depending on which set is shorter), e.g., "No, all but Badger #1 and Pathfinder #2," or "None" if there are only exceptions.

Eucalyptus composes elliptical rather than complete sentences as responses simply because NL generation is not an issue being addressed here. Elliptical responses result in a less mechanical and more conversational style of dialogue and also suggest the permissibility of elliptical NL inputs by the user [11], discussed in the next section.\*

---

\*At the same time, however, elliptical responses may not provide sufficient feedback to reassure the user that the system correctly understood the query.

As mentioned earlier, aircraft attributes are not modeled as extensional objects but as intensional formulae whose values are accessed from KOALAS itself. Eucalyptus prefixes each returned attribute value with the aircraft's ID and, if necessary, the attribute type. For example, the answer to the query *What bearing and distance do the moving fighters have?* prefixes both the aircraft ID and the attribute type to the result values:

F14 #1: Bearing: 200 & Distance: 300 and F14 #2: Bearing: 300 & Distance: 400.

Eucalyptus does not do any syntactic sugaring to the string values that KOALAS returns to attribute queries, but reports them in just the same format as they appear in the graphical display.

The advantage of a NL input query facility over conventional graphical access and data display techniques (as well as more advanced graphical query systems [12]) is similar to the advantage of descriptive referring expressions: the user can *describe* the desired information and let the system perform the search operations itself. In certain cases, however, it is undoubtedly more convenient to visually search the tabular and slot-filler forms of the KOALAS graphical display, such as quickly scanning all the information relative to any one individual aircraft. Thus an integrated interface like Eucalyptus gives the user the option of either mode of access, depending on the complexity of the information being sought.

### Discourse Capabilities

The goal of adding discourse tracking to the Eucalyptus human-computer dialogue is to allow the user to employ vague, ambiguous, or abbreviated inputs which the system can then resolve in terms of the current dialogue context, reducing the amount of repetition in user input and increasing the naturalness of the human-machine dialogue. Eucalyptus attempts to resolve two forms of abbreviated NL input, anaphora and ellipsis. Broadly defined, anaphors are abbreviated references to entities already mentioned earlier in a discourse. They are one of the most commonly used NL mechanisms for avoiding excessive wordiness and keeping a discourse focused on new rather than old information. Ellipsis is an elision of lexical material that leaves a sentence or phrase syntactically incomplete and interpretable only in the context of prior utterances. Although naive users often believe that computers with NL capability actually prefer abbreviated (or so-called "telegraphic") inputs of this sort, it is really just the opposite: the less complete the syntax and semantics of the input, the more inferential work the machine generally has to do.

### Anaphora

The anaphoric referring expressions handled by Eucalyptus are pronouns (*it, he, they*), headless quantifiers (*one, any, each*), demonstratives (*that, those, there*), and reduced definite NPs (*the Badger*). Since definite NPs are not necessarily anaphoric but could be references to entities assumed to be already known to the user, Eucalyptus takes a standard approach [13] of trying the anaphoric reading first and then, if that fails, the non-anaphoric. For example, *the threat* has a non-anaphoric interpretation so long as only one threat exists in the simulation, but has only an anaphoric interpretation once additional threats appear. The reference resolution mechanism is discussed in more detail in a later section, but in a nutshell is a single-context-space, focus-based approach that searches ordered lists of data structures called Discourse Entities that are generated for each referring expression in the user's input.

There is little opportunity in the KOALAS domain to use anaphora in paired imperative clauses, because for the most part the application task only requires the KOALAS user to issue an order to an aircraft or modify a hypothetical threat and then make no further reference to the aircraft or threat until considerably later in the simulation run (after the first order has been successfully carried out). The one exception is the pair of threat assessment operations **Make** and **Lock on Track**, which are commonly issued in sequence and can be conveniently verbalized in Eucalyptus as *Make a new threat here and lock it*

*on track*. Eucalyptus responds to this input by creating a **Make** dialogue window and then (after it has been completed and dismissed by the user) a **Lock on Track** window with the ID of the newly created threat automatically filled in. The other way anaphora might occur in KOALAS would be if the user chose to issue a single command or query using more than one sentence, as in (28), but merging multiple sentences into a single interface command or query is beyond Eucalyptus's current capabilities.

(28) *That fighter moving to station 1? Have him refuel.  
Relieve fighter 1 to refuel. Have fighter 12 relieve him. He should refuel at tanker 1.  
Make a threat here. It should consist of 5 Badgers. Give it a course of 300.*

Where anaphoric reference becomes more useful is in connection with the query facility, where it can enhance the sense of coherent dialogue between human and machine, for example:

*Are any fighters vectoring to threats?  
Yes, F14 #1.  
Which one is he vectoring to?  
Badger #5.  
Are there any fighters still on the carrier?  
Yes, F14 #18 and F14 #19.  
Deploy one of them and vector him out to that threat.*

The user can refer anaphorically to entities reported by the query answering system (e.g., *he*, *them*, and *that threat* refer to F14 #1, F14s #18-19, and Badger #5, respectively) because Eucalyptus creates a Discourse Entity for each noun phrase generated by the query system and enters it into the discourse history, just as it does for each referring expression uttered by the user.

It was mentioned earlier that Eucalyptus does not consider KOALAS-initiated events such as the hypothesizing of a new threat or the proposal or execution of tactical advice to be contributions to the user-machine discourse. As a result, the newly appearing threat icon or the entities involved in the advised fighter order cannot be referenced anaphorically. For example, if the user introduces a new hypothetical threat and the system subsequently introduces one of its own, a followup anaphoric reference to *that threat* co-refers to the user- rather than machine-created one.

Finally, since the speech recognition component cannot accommodate relative clauses due to limitations on the size and complexity of grammar that it will accept, a relativized imperative like *Have all the fighters vectoring to a threat that are low on fuel return to the carrier* must either be typed in or else rephrased as a sequence of queries and final imperative employing anaphora:

*Which fighters are vectoring to a threat?  
F14 #1, F14 #3 and F14 #6.  
Are any of them low on fuel?  
Yes, F14 #1.  
Have him return to the carrier.*

### ***Ellipsis***

Eucalyptus handles elliptical followups (either queries or imperatives) by using the prior utterance's prelogical form (a case frame instantiation) as a template to be re-evaluated after possibly replacing one of its arguments with a new argument from the followup. Interpreting ellipticals in terms of the previous sentence of the current discourse segment is widely accepted [14], but how it is done in any given NLP system depends on the system's particular representational formalisms. The Eucalyptus prelogical form was a suitable representation in two ways: the case frame incorporates the semantic type constraints of its

arguments, and the form also retains some syntactic information, in particular identifying the syntactic subject.

Eucalyptus handles three basic classes of elliptical sentences. The first consists of simple repetition formulae like *Again*, which simply cause the prior utterance's prelogical form to be re-evaluated:

*Increase the simulation speed.*

*Again.*

[Increases it a second time]

The second, VP-gapped sentences, are interpreted by making a copy of the prior utterance's prelogical form to serve as the elided clause and replacing its subject operand by the subject of the elliptical:

*Which F14s are moving to station 1?*

*Which UAVs are?*

[Which UAVs are moving to station 1?]

*Is fighter 1 moving to station 1?*

*Is fighter 2?*

[Is fighter 2 moving to station 1?]

*Is fighter 1 moving to station 1?*

*Recall a UAV that is.*

[Recall a UAV that is moving to station 1.]

In the third class, the subject of the elliptical replaces one of the arguments (not necessarily the subject) of the prior utterance's logical form. Eucalyptus searches the prelogical form of the prior utterance for an argument slot that will semantically accept the subject of the elliptical, and performs the substitution:

*Are any F14s moving to station 1?*

*Any UAVs?*

[Are any UAVs moving to station 1?]

*What about station 5?*

[Are any UAVs moving to station 5?]

*I meant station 4.*

[Are any UAVs moving to station 4?]

*Move fighter 1 to station 5.*

*Fighter 2 also.*

[Move fighter 2 to station 5.]

*Make that station 4.*

[Move fighter 2 to station 4.]

Elliptical inputs of this type can also be used to correct speech recognition errors:\*

*Is fighter 4 moving to the tanker?*

[System heard "fighter 14", "that tanker"]

*I said fighter 4.*

[Move fighter 4 to that tanker.]

*The tanker.*

[Move fighter 4 to the tanker.]

Embedded clauses are included in the search for the argument to replace. If more than one substitution is possible at the same clause level, the system reports the ambiguity to the user. When two possible substitutions occur at different clause levels, however, Eucalyptus does not report an error but just replaces the highest level one, as in the final example:

*Which fighters moving to station 5 are low on fuel?*

*What about station 4?*

[Which fighters moving to station 4 are low on fuel?]

*Have fighter 1 relieve fighter 2.*

*Same with fighter 3.*

[System: "Sorry, don't understand."]

*Is an F14 moving to a tanker that is refueling a UAV?*

*What about an EA6B?*

[Is an EA6B moving to a tanker that is refueling a UAV?]

This last rule is really a preference heuristic (i.e., "Is an F14 moving to a tanker that is refueling an EA6B?" is a valid but less likely interpretation), and so should probably be reported as an ambiguity until

\*Thus corrected NPs are not removed from the discourse history but are "masked" by the more recent correcting NPs.

Eucalyptus is provided with more fully developed NL output capability to query the user for the intended meaning.

## MULTIMEDIA INTEGRATION

A primary objective in Eucalyptus is not to demonstrate that NL is a superior interface medium to graphics or vice versa, but that the two have fundamentally different and complementary strengths that should both be available to the user simultaneously. This section explores integrated, mixed-media transactions in which graphics and NL combine and interact.

### NL Input to Dialogue Windows

As we've seen, the KOALAS GUI uses popup dialogue windows to prompt the user for arguments and parameters needed to complete a proposed fighter order or threat assessment operation. Each window contains one or more labelled text fields (for numeric information such as ID, bearing, and course) and may also contain one or two labelled cycle items for choosing from closed sets of parameter values such as speed (**Slow, Fast, Stationary**) and aircraft type. These windows do not just passively accept user input to their slots, but have a high-level input behavior that prompts for certain arguments in a particular order. When the window appears, the cursor is positioned in the first text field and the window displays a message prompting the operator to select an object or position on the radar screen with the mouse. Once s/he does so, the system automatically fills in one or more text fields in the window (sometimes setting a choice in a cycle item as well), the cursor moves to the next empty text field, and a new prompt is displayed. When no empty fields remain, the system prompts the operator to click once more for final confirmation. The right mouse button can be used as an "undo" operator at any time, retracting the previous moused data entry operation by clearing the relevant text fields, resetting the cursor, and displaying the previous prompt. Although the user can also interact directly with the cycle items or text fields using the mouse and keyboard, this input mode is not interactive with the high-level input mode, so the user cannot type certain fields and then use the mouse to fill others. However, the keyboard can be used to edit or change values that have already been entered by mouse input.

Although the low-level operation of manually typing data into a labelled text field constitutes a "dialogue" technique from the standpoint of basic human-computer interaction, the higher level input mode of the KOALAS dialogue windows more closely resembles a true NL dialogue: the window prompts the user for information with a sentence like "Click Left Button on Fighter" and the user responds with a referential gesture by graphically selecting an object or location on the radar screen. While Eucalyptus was originally intended to only accept elliptical inputs as followups, given the dialogue-like nature of the KOALAS popup window interactions, it was decided that the user should be allowed the option of speaking singular referring expressions (proper names and noun phrases) in response to the window prompts, analogous to the clicks on the radar screen (graphical references) already supported by the GUI.

Each referring expression directed to a dialogue window undergoes parsing, semantic interpretation and domain-model dereferencing just like those contained in sentential inputs to the system. If the resulting singleton referent is of the proper semantic type to fill the current text field, its ID number is entered into the field, the cursor is advanced to the next empty slot, and the system prompt updated, exactly paralleling the system's feedback to graphical input. The entered data responds to the "undo" (right-click) graphical operator just the same as graphically entered data. The resulting interaction is analogous to a natural language dialogue such as the following:

User:       <Move>  
System:     Select a fighter.

User: *One of the F14s holding station 2.*  
System: That's fighter 1. Select a station.  
User: *The same station fighter 11 is moving to.*  
System: That's station 5. Okay?  
User: *Okay.*

Eucalyptus does not permit the user to select and speak to a text field other than the one currently being prompted for, since that would be equivalent to bypassing the window's high-level input behavior by using the keyboard for raw numeric data entry. This is consistent with the initial working decision that Eucalyptus be a spoken NL system, not merely a voice input system: the role of speech input in Eucalyptus is not to serve as a "vocal keyboard" for entering raw data, but to take part in a higher-level NL dialogue about conceptual entities (i.e., NL references), which is why Eucalyptus treats NPs as semantically equivalent to the GUI's high-level graphical denotation of objects.

As mentioned earlier, Eucalyptus permits corrections, queries, advice interactions, and system control commands as "asides" during dialogue window data entry. Since the system assumes that isolated noun phrases are subdialogue interactions with the window, however, such asides cannot include elliptical followups consisting of bare noun phrases. This is a case where a discourse component capable of managing more than a single context space [15] might be useful: if the user could verbally cue the return from an aside to the window, this restriction against using bare NPs would no longer apply.

### Dialogue Windows for NL Command Completion

As seen earlier, verbal fighter control and threat management commands always bring up the same dialogue windows used for data entry in the graphical interface, but with the verbalized arguments already filled in. This serves four purposes: to provide visual confirmation that the system understood and interpreted the input correctly; to allow the user to alter those arguments that have been entered so far; to prompt the user to fill in any remaining unspecified information; and to give the user final accept/reject opportunity. These are the same four roles that the dialogue window plays in the original graphical interface, so it was felt that using the same feedback device for NL input would be an integrated and consistent use of system resources that would avoid complicating the interface with additional output modalities like NL (e.g., "Having fighter 2 relieve fighter 12 to refuel at tanker 1, okay?"), and that providing the same unambiguous, canonical feedback technique to the user in all cases might be less confusing and demanding on the user's attention (although whether it actually does in fact reduce the user's cognitive load would of course require experimental testing).

One feature of the KOALAS graphical interface is to prohibit the user from issuing another Fighter Management, Threat Assessment, or Experimenter Control operation while a dialogue window is still on the screen, beeping and flashing the window to signal that it must be completed and dismissed first. Eucalyptus parallels this behavior by beeping and flashing the window if the user issues a NL command of those sorts while a dialogue window is still pending. Any other command type or query is acceptable, as is also true in the graphical interface (e.g., the user can interact with the current advice or scroll through the display windows during a dialogue window interaction).

### Discourse History for Graphical Operations

Whenever a domain-functional operation is performed using the KOALAS graphical interface, Eucalyptus generates an equivalent prelogical form similar to those that result from verbal input and enters Discourse Entities for each of the operation's implicit or explicit arguments into the discourse context. This

allows two kinds of interactions between graphics and NL. First, the graphical operation and its arguments become available as context for interpreting an elliptical follow-up command:

**<Refuel>** **<fighter 1>** **<tanker 1>**  
*Same with fighter 5.* [= refuel fighter 5 at tanker 1]

Second, its arguments become available as antecedents for anaphoric reference:

**<Move>** **<fighter 14>** **<station 1>**  
*What bearing does he have?* [*he* = fighter 14]

In addition to Fighter Management operations, Threat Assessment, Experimenter Control, and Advice Window operations also generate context history, allowing interactions such as the following:

**<Make Threat>**  
*Vector fighter 1 there.* [*there* = to new threat]  
**<Explain Advice>**  
*Accept it.* [*it* = current advice]  
**<Show Real Threats>**  
*Turn them off.* [*them* = real threats]

### Deictic Reference

Deixis is a form of reference in which pointing gestures accompany NL referring expressions. It is a powerful and useful mechanism in human communication because each component of the reference can be minimally specific, but in combination can constrain or disambiguate each other sufficiently to yield an unambiguous denotation.

A deictic reference in Eucalyptus consists either of the word *here* or a (possibly headless) NP having the demonstrative *this* or *these* as determiner (e.g., *this*, *these*, *these hypothetical Badgers*, *this fighter vectoring to a threat*), accompanied by one or more middle mouse clicks on the radar screen. As each mouse event occurs, its x-y coordinates are stored in a list for later access by the NLP component. During NL processing, these mouse events are correlated sequentially with the deictic NPs in the sentence. The referent for each NP is then determined by finding the subset of KOALAS entities located at or nearest to the mouse event's x-y coordinates that conforms to the NP's semantics (including the semantic presuppositions of its host predicate). "Nearest to" is defined as the closest aircraft within a distance of 10 pixels from the mouse coordinates, which is how it is defined for direct reference in the original graphical interface.

For example, a click on a tanker aircraft could be a reference either to the aircraft itself, to the radar screen sector (duty station) at those coordinates, or to the position indicated by the coordinates themselves. Accompanied by NPs like *this aircraft*, *this friendly aircraft*, *this support aircraft*, *this tanker*, or *this E2C*, the aircraft is chosen as referent. Similarly, *this sector* or *this station* selects the duty station and *this position* selects the screen coordinates. For an NP like *this enemy aircraft*, *this fighter*, or *this F14*, however, the intersection is empty and Eucalyptus responds "No such (aircraft, fighter, F14) exists."

The expression *here* is maximally vague and in the same example could be a reference either to the tanker, the screen sector, or the coordinates. In that case, Eucalyptus uses the semantic presuppositions of the expression's host predicate to try to resolve the reference. For example, in the context of the imperative utterance *Move fighter 1 here*, the verb's predicate P-MOVE-TO-STATION expects a duty station as its argument and so chooses the sector as referent. In a query, however, *move* can mean either P-MOVE-TO-STATION or P-MOVE-TO-TANKER, so *Is fighter 1 moving here?* receives both meanings: the system answers "Yes" if fighter 1 is moving either to that station or to the tanker. In the context of *Refuel fighter 1 here*, the predicate P-REFUEL's semantic expectations cause just the tanker to be chosen as referent, and

with the input *Create a hypothetical Badger here*, the system chooses the radar screen coordinates. Finally, given an utterance like *Vector fighter 1 here* (where P-VECTOR presupposes an enemy target as argument), the intersection set is empty and the reference component responds “No such threat exists.” Similar sorts of presuppositional behavior can be observed by clicking on the radar screen and asking questions such as *Who is this*, *What is this*, *What aircraft is this*, etc.

The deictic reference mechanism used in Eucalyptus is very similar to the direct-reference mechanism for moused object denotation in the KOALAS graphical interface, where the dialogue window slot being prompted for expects an argument of a certain sort (friendly, fighter, threat, station, or position) and the interface selects the object of that sort closest to the mouse coordinates. Thus, if a fighter, tanker, and hypothetical threat were in such close proximity that their icons overlapped on the radar screen, the KOALAS graphical interface could still select the object satisfying the type expectation, just as Eucalyptus can using deictic reference (e.g., *this F14*, *this EA6*, *this Badger*). In the more conventional direct-manipulation approach where each icon would be associated with a mouse-sensitive active region, only the uppermost of the overlapping icons would be visible to the mouse and could be selected, requiring additional expose/hide operations to access the hidden objects.

Two simplifying assumptions are made in Eucalyptus’s treatment of deixis. First, only one of the deictic expressions in a sentence may be plural. Second, the restrictive choice of deictic vocabulary (*this*, *these*, and *here*) ensures that no ambiguity exists between deictics and discourse anaphors, which must use the distinct set of demonstratives *that*, *those*, and *there*. In general English this distinction is not made – for example, the phrase “this distinction” was just used anaphorically, and one can refer deictically to *this book* and *that one* to indicate proximity to the speaker. Since deictic references in Eucalyptus all involve figuratively “touching” referenced entities by clicking on them with the mouse, however, the strictly proximate terms are more appropriate,\* and a user interaction study to be discussed later suggests that KOALAS users may tend to verbalize deictics and anaphors using these same vocabulary distinctions.

These two constraints allow the system to perform a simple left-to-right assignment of mouse events to NPs without requiring any time correlations. For example, a sentence with multiple plural deixis like *Have these F14s and these UAVs refuel* would require time correlations to assign a partition of the mouse events to each deictic expression. Similarly, if a single referring expression could be either anaphoric or deictic, then a sentence like *Have that tanker refuel that fighter* accompanied by a single mouse click would require time correlation to determine which NP should be interpreted as anaphoric (i.e., that tanker or fighter mentioned earlier) and which as deictic.

The use of timestamps is avoided because while it is easy to timestamp mouse events as they occur, timestamping deictic NPs in Eucalyptus is more problematic. This is because the initial phonetic processing takes place in hardware independent of the host computer, so although the transcribed speech data does include the overall duration of the utterance as well as the relative time position of each word in it, the system cannot know what time the utterance actually began. Since it does know the arrival time of the decoded signal at the host computer, however, if it were to assume a small fairly constant or linear decoding time then it would be possible to extrapolate with reasonable accuracy the time at which each deictic NP began (that is, when each word *this*, *these*, or *here* occurred). Eucalyptus does in fact timestamp deictic NPs in just this way, in the event they become needed for future extensions of the system.

In the original graphical interface, the left mouse button is used to select the operands to fighter orders and threat management commands, so ideally the same button should have been used for deictic reference as well. But since in Eucalyptus the host computer does not know when speech input is occurring

\*Such might not be the case if eyetracking (gaze direction detection) were used instead of deliberate pointing gestures.

until after the press-to-talk switch has been released at the end of the utterance, the system would not be able to distinguish deictic from ordinary graphical input if the same button were used for both. Besides, when no dialogue window is pending, the left mouse takes on a secondary functionality (bringing up a status information window for an aircraft) that also could not be distinguished from deixis. The other two mouse buttons already have functional definitions in the original interface and so are unavailable for deictic reference: middle click centers the radar display at the mouse coordinates, and right click displays the threat sightings for a UAV or hypothetical threat. Adding the Control key to left-click for deictic reference would have been awkward since the user's non-mouse hand should be left free to control the press-to-talk switch. Therefore, it was decided to reassign the original mouse-middle functionality (radar screen recenter) to Control-mouse-middle, freeing up mouse-middle for deictic reference.

An alternative would have been to require no mouse click at all during deictic reference, but to just take the x-y coordinates of the mouse at the time the deictic NP was uttered. Since the NP's timestamp cannot be extrapolated and assigned until after the complete utterance has been decoded, however, it would have been necessary to keep an ongoing history of mouse locations at all times during the simulation for possible correlation later with deictic NPs. If demonstrative NPs were allowed to be either deictic or anaphoric (as in the example *Have that tanker refuel that fighter* discussed earlier), the problem of correlating possible deictic references with mouse coordinates would be compounded: the mouse just might happen to be positioned over a fighter icon when the speaker uttered what was intended to be an anaphoric reference, resulting in a misinterpretation. Requiring that mouse clicks accompany deictic references avoided these problems.

When the left mouse button is used to graphically denote an argument to a dialogue window, the user gets immediate visual feedback of the success of the operation – the argument's ID appears in the current window slot. However, the user does not find out whether a deictic reference was successful until the accompanying speech input has been fully processed and the dialogue window is displayed (or in the case of a question, until the query answering facility responds). To reassure the user that the deictic mouse click was effective, Eucalyptus provides visual feedback by blinking the icons of all the potential referents at or near the mouse coordinates when the click occurs.

### Threat Reference

The ability to view the actual threats during a KOALAS simulation is treated as a special "God's eye view" that is normally not invoked, since the KOALAS philosophy is that the operator only create and interact with hypotheses about what threats might really be out there. As a result, the Threat Status window displays status information only for hypothetical threats unless the **Real Threats** switch is turned on, in which case real threats receive status entries in the window and their blips are added to the radar screen display (at which point they can be referenced graphically, i.e., moused on). Since one Eucalyptus design objective was for the NL and graphical interfaces to have exactly parallel semantic behavior, Eucalyptus does not make real threats available for NL reference unless they are graphically visible. For example, *What is threat 1's bearing?* asked with the **Real Threats** switch off accesses hypothetical threat 1 (if such exists), otherwise responding "No such threat exists" even if a real threat with that ID exists. With the switch on, if both a real and hypothetical threat 1 exist, the system responds "More than one such threat exists," but otherwise reports on the one that does. Under the same circumstances, however, the input *Vector fighter 14 to threat 1* (where *vector* semantically presupposes a hypothetical threat as argument, since one cannot order fighters to vector to real threats) does not result in a reference ambiguity. Finally, the user can always qualify the threat reference with the appropriate adjective (*real threat 1, the hypothetical threats*) if more specificity is needed.

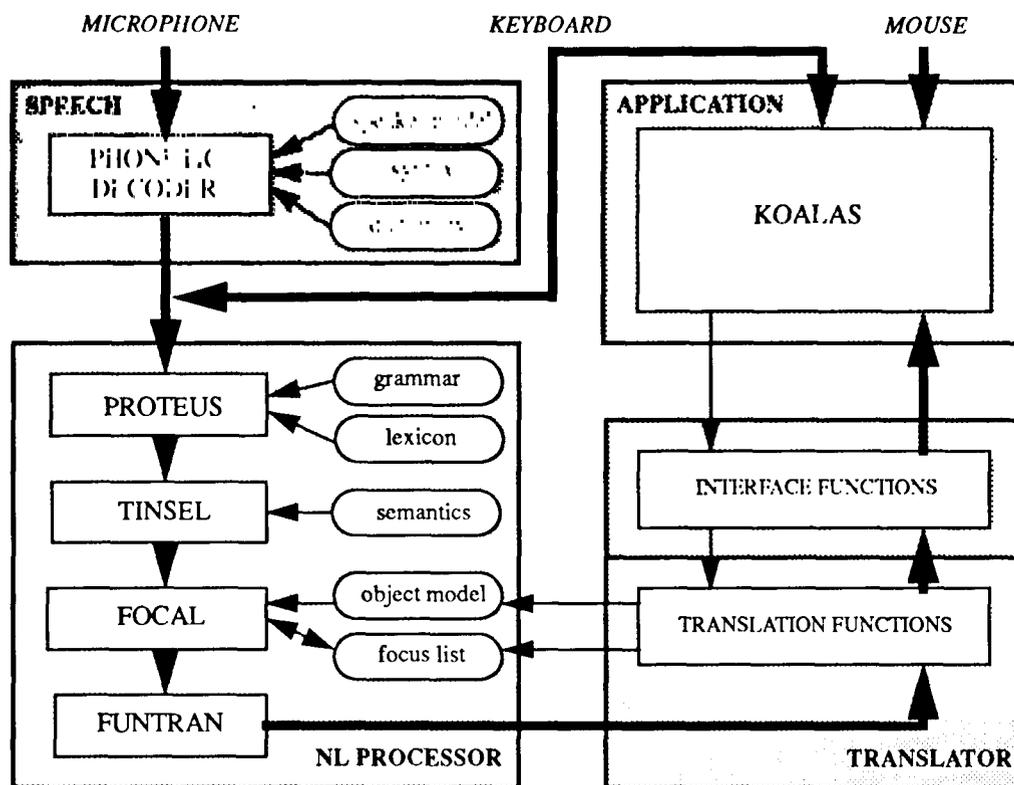


Fig. 2 - Eucalyptus architecture

## EUCALYPTUS ARCHITECTURE

Figure 2 shows the architecture of Eucalyptus. The three main modules of the system are (counter-clockwise from upper left) the speech recognition component, natural language processor, and application-specific translator, interfacing to the target system in the upper right. These components are discussed in more detail in the sections that follow.

### Speech Recognition Component

Eucalyptus uses the Speech Systems Inc. (SSI) Phonetic Engine 200 continuous speech recognizer. The SSI system requires a finite state grammar of all acceptable input utterances (compiled from a data file written in context-free notation) which it uses to find the closest match to the phoneme sequence it has derived from the input speech signal. The speech grammar for Eucalyptus consists of about 250 noniterative, nonrecursive productions capable of recognizing about 115 million different spoken utterances. Since the representational formalism only supports context-free notation, syntactic and semantic constraints are encoded in the grammar by proliferating nonterminal symbols, resulting in a so-called semantic grammar as opposed to a phrase structure grammar.

The subset phonetic dictionary for this application contains just under 260 words. Twenty-five of these are integers, seven are alphabetic characters (for speaking acronyms like *UAV* and *EA6B*), 34 are apostrophized plurals (for speaking genitives like *the fighter's* and *fighter I's*), and the remainder are ordinary English words. Of these, eight (*badgers*, *koalas*, *pathfinder*, *pathfinders*, *phoenixes*, *reposition*, *vector*, and *vectoring*) were not present in the SSI standard phonetic dictionary and had to be added using

their dictionary update facility. Eucalyptus allows the user to pronounce aircraft ID numbers either as single words (*nineteen, twenty*) or as their individual digits (*one nine, one niner, two zero, two oh*), the latter often providing improved recognition accuracy.

Although recursive phrase structures such as compounds, nested prepositional phrases, and relative clauses present no problem to the natural language processing (NLP) component described in the next section, adding such structures to the speech recognition grammar increases its size and reduces its recognition accuracy to such a degree that it is not profitable to do so. In particular, attempting to add relative clauses to the speech grammar actually renders it too large to compile. This is unfortunate since relativization is one of the most powerful linguistic mechanisms for constructing the descriptive denotational expressions that make natural language such a desirable adjunct to graphical interfacing. At present, then, sentences that use recursion must be entered by keyboard rather than spoken.

The SSI system provides generic (speaker-independent) male and female speaker models that work quite well within the constrained syntax and semantics of the Eucalyptus domain. The most recognition errors are encountered in two context-insensitive word classes: determiners (*a, the, this, that*) and ID numbers (*thirteen, fourteen*). Very short sentences (*Okay, Cancel that, Threat 2, Exit*) tend to yield the most serious misrecognitions since they provide the least amount of phonetic context for the speech decoder to identify the intended predicate.

### **Natural Language Window**

Eucalyptus adds one window, the **Natural Language Window** (Fig. 3), to the KOALAS tool's graphical display. The ASCII transcription of the spoken utterance is echoed in a text field and then passed automatically to the rest of the system for NL processing and execution unless the **Confirm Speech** check switch is turned on, in which case the **Parse** button must be used to continue processing. This allows the user the opportunity to check the accuracy of the text and perhaps make minor edits to it with mouse and keyboard before sending it on for further processing. With **Confirm Speech** off, the system can be operated completely hands-off using voice only. Alternatively, the user can clear the text field with the **Clear** button and enter an entire sentence via the keyboard followed by pressing the **Parse** button. Finally, below the text input field is a write-only text output field where Eucalyptus progress messages and final responses appear.

### **Natural Language Processor**

After a sentence string has been transcribed by the speech component (or typed at the keyboard), it passes to the natural language understanding component for linguistic analysis and conversion to logical form. The NLP component consists of four modules: the PROTEUS parser developed at New York University's Courant Institute [16], and the TINSEL case-frame interpreter, FOCAL reference resolution module, and FUNTRAN quantified-expression builder, all three developed in-house. The Appendix illustrates

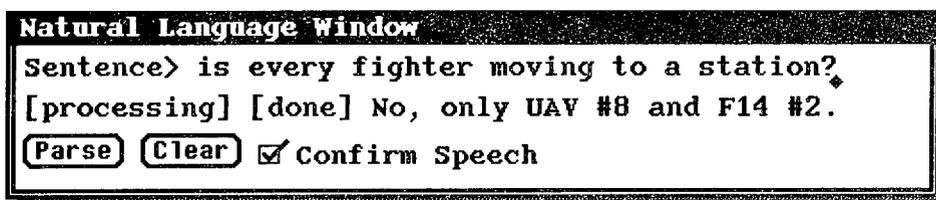


Fig. 3 – Eucalyptus Natural Language Window

the various stages of data processing for a typical input sentence, discussed in more detail in the subsections that follow.

### *Syntactic Analysis*

The PROTEUS syntactic grammar used in Eucalyptus is based on a streamlined NL interface grammar developed for the InterFIS project [17], and consists of approximately 180 context-free phrase structure rules and 50 procedural syntactic restriction rules. Besides covering the same input data as the speech recognition grammar, it also covers compounding (on imperatives, questions, and noun phrases), iteration of NP modifiers, and recursive embedding within NP postmodifiers (relative clauses and prepositional phrases). The PROTEUS lexicon contains about 425 words, nearly twice as large as the speech recognition lexicon. Most of the extra words are unused morphological variants (verb aspects and noun plurals) generated automatically from root forms by the PROTEUS lexical macros.

The parser's input is the ASCII string contained in the Natural Language Window text input field, either typed from the keyboard or transcribed from speech (Appendix, example 1). The parser's output is a regularized form suitable for semantic interpretation called the Intermediate Syntactic Representation (ISR) [18]. Among the regularizations that occur are verb phrase depassivization, tokenization of idioms and stock phrases (*auto accept, as well*), translation of numeric phrases (*twenty, three quarters*) into numbers (20, 3/4), reduction of multi-token pronunciations (*one niner, e two c*) into individual tokens (19, E2C), and extraction of the negation operator NOT from verb contractions (*aren't, didn't*).

### *Semantic Interpretation*

The predicate-argument interpreter TINSEL [19] runs in tandem with the parser, pattern-matching each ISR against a hierarchy of about 160 semantic frames (typed slot-filler structures) and a roughly equal number of syntactic-semantic mapping patterns, inserting semantic class and thematic role information into the ISR (Appendix, example 2) to yield a semantically-marked version (Appendix, example 3). During interpretation, TINSEL applies predicate domain constraints (selection restrictions) to prune anomalous interpretations and thus help guide the parse. TINSEL performs five primary interpretive functions:

- Many-to-one mapping of syntactic to semantic roles (thematic resolution). For example, in each sentence of (29), *fighter 2* maps to the `:instrument` role even though it is the subject of the first and prepositionally marked *with* in the second.

(29) *Fighter 2, relieve fighter 1.*  
*Relieve fighter 1 with fighter 2.*

- Many-to-one mapping of lexical to semantic classes (synonymy), for example mapping the verb in each sentence of (30) to the predicate `P-RECALL`.

(30) *Fighter 1, return to the carrier.*  
*Recall fighter 1.*  
*Have fighter 1 come back to the carrier.*

- One-to-many mapping of syntactic to semantic roles (thematic ambiguity). For example, in the first sentence of (31), the subject *that aircraft* maps to the `:patient` role (the entity affected by the verb), whereas in the second it maps to the `:instrument`.

(31) *Is that aircraft refueling?*  
*Is that aircraft refueling anyone?*

- One-to-many mapping of lexical to semantic classes (polysemy), for example mapping the noun *fuel* (*How much fuel does fighter 1 have?*) to the sort (0-ary) predicate P-FUEL, and the verb *fuel* (*Fuel fighter 1*) to the relational predicate P-REFUEL.
- Semantic analysis of NP and VP adjuncts as explicit roles or predicates, such as interpreting *the fighter on the carrier* as “the fighter which is located on the carrier” and *Move fighter 1 fast* as “move fighter 1 at high speed.”

The TINSEL semantic classes are organized into an ISA hierarchy to facilitate encoding of domain type constraints, so subsuming noun classes automatically become available for use in reference: both fighters and threats can be referred to as *aircraft*, agentive entities can be referenced as *anyone*, etc. Assuming that the users of a tool that simulates a military air control system will be comfortable adhering to a constrained vocabulary and regular syntax, paraphrase in Eucalyptus is limited to slight syntactic variations and some minor synonymy of verbs and prepositions. Wide variety of usage is permitted, however, in the referring expressions (pronouns and NPs), since discourse coherence expresses itself so strongly there. The reference resolution module is the next major component to be discussed.

### *Preferences and Transformations*

Two smaller modules not shown in Fig. 2 operate on the TINSEL output before passing it along for reference resolution and further processing. The first module weeds out semantically acceptable but inferior parses by applying heuristic semantic preference rules, such as preferring parses as complete sentences rather than as sentence fragments. For example, *Is a fighter moving to station 5?*, in addition to the expected sentential parse, also receives a less preferable analysis as the elliptical “Is a fighter which is moving to station 5 [VP]?” The second module transforms certain TINSEL representations to make them more suitable for conversion to logical form, for example modifying the semantic structure “Which fighters have a bearing of 300?” into “For which fighters does their bearing equal 300?” The transformation module is also responsible for generating complete semantic representations for elliptical inputs.

### *Reference Resolution*

Focus-based reference resolution is performed by the FOCAL (FOCUS ALgorithm) module, developed by ONR fellow Gina-Anne Levow of the Massachusetts Institute of Technology. For each referring expression in the TINSEL prelogical form, FOCAL generates a data packet called a Discourse Entity containing the phrase’s unique ID number, semantic class, number (singular/plural), the case role it played in the utterance, and extension. The extension is a set of pointers to one or more symbols, each representing a referenceable entity in the domain. The focusing algorithm is based largely on Refs. 20 and 21 and uses number, semantic class [22], and recency to select the most likely antecedent for each anaphoric reference. The discourse entities are stored in data structures (current focus, actor focus, alternative focus list, and focus stack) searched in that order by the focusing routine when trying to resolve possible anaphora (Appendix, example 4). As mentioned earlier, definite references first seek resolution as anaphors, and then as references to the appropriate number of objects from the domain context. Indefinite extensional references (*a fighter, one of the threats*) are assumed to represent “don’t care” specifications which, in the context of an imperative, give the system itself permission to choose any member of the extension set.

### *Quantification*

The FUNTRAN (FUNCTIONAL TRANslator) component transforms each TINSEL case-frame representation into an expression in first-order predicate calculus (FOPC) with restricted quantification (RQ). RQ provides a set-formation operator that allows complex noun phrases to be represented in the logical

form as self-contained expressions, unlike conventional (e.g., Ref. 23) FOPC representations where the predicates representing the NP are simply connected inline with those representing its parent clause.

FUNTRAN first establishes a scoping order over the NPs in the expression. In cases of scoping ambiguity, FUNTRAN always gives universal quantifiers wide scope and negation operators narrow scope. Hence *Is a fighter vectoring to every threat?* is always interpreted as asking whether each threat has some (possibly different\*) fighter vectoring to it. This is justified by the semantics of the domain, since individual fighter actions in KOALAS apply only to singular goals (no fighter can simultaneously vector to more than one threat). Similarly, *Is a fighter not vectoring to threat 1?* is assumed to ask whether there exists at least one fighter that is not vectoring to that threat, which is the generally preferred (syntactic) scoping.

The scoped NPs are then converted to restricted quantifier expressions: universal and plural definite NPs are mapped to FORALL quantifiers, indefinite NPs to EXISTS, and singular definite NPs to EXISTS! (exists-unique). The result is the logical form of the underlying proposition of the sentence (Appendix, example 5). The appropriate performative is then chosen based on the syntactic type of the utterance: imperatives map to the COMMAND performative, Wh-questions to TELL, and yes/no questions to one of the TELLIF family (TELLIF, TELLIFANY, TELLIFALL, and TELLIFNONE) depending on the identity of the outermost quantifier (EXISTS!, EXISTS, FORALL, and NOT EXISTS, respectively).† In the case of all queries but TELLIF, the outermost quantifier is then converted to the set-formation operator SETOF to allow the performative to return more informative answers than just a flat “yes” or “no.” The performative-marked quantified expression (Appendix, example 6) is then ready for evaluation.

### Operator and Performative Evaluation

The EXISTS, EXISTS!, and FORALL operators are defined as Lisp macros, each with the syntax (operator variable set test), that expand into lambda expressions when evaluated. EXISTS succeeds if any element of the set passes the test, EXISTS! succeeds if one and only one element passes, while FORALL requires that every element pass. The SETOF macro has the syntax (SETOF variable class-or-set &optional test) and returns the subset of elements belonging to the class or set that pass the (optional) test. A test can be either another logical operator expression, or a call to a Translation Function (such as P-MOVE and P-HOLD in example 6 of the Appendix). The evaluation of Translation Functions is discussed in the section that follows.

In the case of queries, evaluation of the quantified expression computes a return value, either a set of conforming instances or a Boolean TRUE/FALSE. This value is then passed to the enclosing performative operator, which reports the result back to the user in some appropriate fashion. The performatives TELL, TELLIFALL, TELLIFANY, and TELLIFNONE compare the set of conforming instances <I> to the set of exceptions <E> to produce a short but informative response, as shown in Table 1. For example, in the evaluation of the expression in example 6 of the Appendix, the set of conforming instances X1 (all F14s moving to a station fighter 1 is holding) is compared to the set of exceptions N1 – X1 (where N1 is the set of all F14s). If the basic answer to the query is “No” but some conforming instances do exist, the system goes on to list the shorter of the two sets (e.g., “No, only F14 #1 and F14 #3”). The performative TELLIF just responds “Yes” or “No” based on the value of its Boolean argument.

---

\*In the KOALAS domain, each fighter vectoring to a threat *must* be different, but that is a restriction the FOPC quantifier does not capture.

†Performatives and quantifiers for answering questions about quantity (*how many, how much, more than, less than, etc.*) are also provided.

Table 1 – Performative Behavior

Performative	Example	No Instances	No Exceptions	Fewer Instances	Fewer Exceptions
TELLIFALL	<i>Are all moving?</i>	No, none.	Yes.	No, only <I>.	No, not <E>.
TELLIFANY	<i>Are any moving?</i>	No.	Yes, all.	Yes, <I>.	Yes, all but <E>.
TELLIFNONE	<i>Are none moving?</i>	Yes, none.	No, all.	Only <I>.	All but <E>.
TELL	<i>Which are moving?</i>	None.	All.	<I>.	All but <E>.
TELLIF	<i>Is he moving?</i>	No.	Yes.		

### Application-Specific Translator

The backend translator module provides the function definitions by which NL predicates like P-MOVE and P-HOLD query the KOALAS internal state or issue commands for execution by the application. The module consists of two intercommunicating sets of code, one written in Lisp and linked in with the natural language processor, the other written in C and linked in with KOALAS.

On the Lisp side, Eucalyptus provides each TINSEL verb class with a function definition called a Translation Function (TF). When evaluated, these expand into one or more calls to Interface Functions (IFs), which are primitives for interacting with the KOALAS system and have their code definitions on the C side. When a TF is called in the context of a query or NP modifier, it translates into a call to the Interface Function CONFIRM-FACT, a predicate (returning either TRUE or FALSE) that consults KOALAS to see whether the relation predicated by the TF is true of its arguments. When called in the context of an imperative command, on the other hand, the TF generally acts as a procedure, expanding into a sequence of IF calls that cause KOALAS state changes to take place. For example, in the imperative *Move fighter 1 to station 5*, the P-MOVE TF call expands into the following sequence of IF calls:

```
(MAKE-MOVE-FIGHTER-POPUP)
(SET-TYPE-ITEM F14)
(SET-ID-ITEM "1")
(SET-STATION-ITEM "5")
(SET-MESSAGES)
(SHOW-CURRENT-FRAME)
```

As with CONFIRM-FACT, each of these Lisp calls maps directly to a corresponding C function linked into KOALAS. Respectively, they create a "Move Fighter" dialogue window, set its fighter type to the KOALAS constant F14, set its fighter ID to 1 and station ID to 5, update its user prompt message, and finally make it visible on the screen.

The IFs represent copies of portions of already existing KOALAS interface code factored out for convenient invocation by the Translation Functions. In most cases, this was done because the original code was interleaved with GUI-related data structures such as mouse events and needed to be called with a different parameter list. In no case was it necessary to create any new data structures or write truly novel code for the IFs, which are just copies and adapted portions of the original source.

A number of KOALAS command verbs have no corresponding data query capability. For example, it is possible to order *Have fighter 1 relieve fighter 2*, but since the KOALAS database only retains a

record that fighter 1 is now moving to fighter 2's former station, it cannot then answer the query *Is fighter 1 relieving fighter 2?* Since such a query is semantically well-formed but pragmatically outside the application's functional domain, the corresponding Translation Function answers with the message "I don't know," in effect a simple kind of metaknowledge (something the system might be expected to know, but knows that it doesn't). A similar approach is taken with questions that involve other than simple present tense (*Was fighter 1 moving? Will he be moving?*), which are automatically covered by the NLP grammar and lexicon but lie outside the application's ability to answer.

## IMPLEMENTATION

The NL processor (PROTEUS, TINSEL, FOCAL, FUNTRAN, and the semantic preference and transformation modules) consists of about 6860 lines of Common Lisp code. The Eucalyptus translator module contains about 2940 lines of Lisp code (Translation Functions) and 2480 lines of C code (Interface Functions). The KOALAS application itself is about 16150 lines of C code. The declarative data files (speech grammar, syntactic grammar and lexicon, semantic model, and scenario-independent referent model) total about 5750 lines of text. The KOALAS application requires 3100K runtime memory, Lisp 6068K, and the Eucalyptus code and data 4684K, for a total of 13852K, or roughly 45% of the available memory on a 32M Sun SPARCstation 2.

In the current version of Eucalyptus, all the C object code (KOALAS itself, the speech decoding routines, and the Interface Functions) is linked into the Lisp process in which the NL processor runs. The Interface Functions between the two sets of code are invoked as foreign function calls from Lisp to C routines and vice versa. Although the simulation and NL processor run synchronously, significant pauses in the simulation are uncommon because the simulation display update rate is once per second and most parses take no longer than that. The only change necessary to KOALAS itself was the modification of the main simulation loop to poll for speech input and send it off to the speech decoding routines.

In another experimental version, all the C object code was linked into one process separate from the NLP (Lisp) process. In this version, the Interface Functions on each side encode and send messages (passed over standard I/O channels) to the other side for decoding and execution. This approach has the advantage that the NL processor and simulation run asynchronously (possibly on different machines), but the message passing traffic in the prototype slowed processing times down considerably. For example, a query like *Which fighters holding a station that no fighters are moving to are low on fuel?* generates on the order of two hundred Interface Function calls, translating into double that number of individual interprocess messages, resulting in an unacceptably long processing time. More efficient message passing, however, might render this approach feasible.

## USER INTERACTION STUDY

A series of data collection studies were conducted by Lisa Achille at NRL's Human-Computer Interaction Laboratory, Information Technology Division, after Eucalyptus had already been developed. In the data collection procedure, a user familiar with the KOALAS tool's graphical interface issued verbal instructions and mouse pointing gestures to a second person, the tool's operator, who then executed the desired actions using the graphical interface. The user's verbalizations, heavily primed by his familiarity with the graphical interface, illustrate a number of issues relevant to the Eucalyptus approach and capabilities and confirm several of the design decisions that were made.

**NL Coverage.** In addition to issuing simple NL equivalents to individual graphical inputs (32), the user also employed compounding (33), quantification (34), anaphora (35), and deixis (36), all representing NL features covered by Eucalyptus.

- (32) *Refuel 5.*  
*Make a threat.*  
*Put a lock on 4.*  
*Vector 5 to track 4 also.*
- (33) *Deploy fighters 9, 8, 7, and 3.*  
*Vector 4 to 12 and vector 1 to 11.*  
*Vector fighters 3 and 9 to threat 5.*  
*Recall 11 and 12.*
- (34) *Deploy two more fighters, one in each sector.*  
*Have all the green fighters return to carrier.*
- (35) *Alter 4 and lock it.*  
*Make a threat with its radar return.*  
*Vector fighter 1 to that threat.*
- (36) *Move 1 to this sector.*  
*Make a threat here.*  
*Delete these threats along in here.*  
*Alter 3 to go here.*

**Anaphora and deixis.** The user employed *that* for anaphoric reference and *this*, *these*, and *here* for deixis, matching the approach taken in Eucalyptus.

**Quantification.** With its strictly FOPC quantification, Eucalyptus cannot handle the complex quantifier *one in each sector* in the sense intended in (33), i.e., deploy each of two fighters in a different sector (implicitly, sectors 1 and 2).

**Overspecificity.** In the first sentence of (32), the user consulted the Fighter Status window to determine the ID numbers of the fighters still on the carrier, and ordered them all deployed by naming them individually in the order they appeared in the window. This reflects his prior knowledge that this is how the graphical interface behaves, deploying one individually identified fighter at a time. In Eucalyptus, the user is free to say simply *Deploy all the fighters on the carrier*, a direct expression of what s/he intended in the first place.

**Verb Synonymy.** In (37), the user erroneously issued a **Vector** order using the verb *move* instead of *vector*, but because the destination is a threat, the operator understood and executed the intended order regardless. The synonymy and type-checking facilities of the TINSEL semantic interpreter perform the same function in Eucalyptus.

- (37) *Move fighter 5 to track 2.*

**Ungrammaticality.** On several occasions, the user ordered that a threat be repositioned using ungrammatical sentences like (38), clearly primed to do so by the graphical interface vocabulary and syntax. The incompatibility between this graphical operator and its NL verb structure was noted earlier.

- (38) *Alter threat 11 to here.*  
*Alter threat 6 over here to my pointer.*  
*Alter threat 1 to the position of fighter 4.*

**Ellipsis.** The user often omitted the head noun when referring to fighters and threats (*Vector 4 to 12*), which Eucalyptus could accommodate with a simple addition to the grammar. The only occurrences of

verb ellipsis were in reference to Advice Window button labels (39) and one case of elided verb in a fighter command (40). In the first case, it appears that the user considers the issuing of advice by the system to constitute a subdialogue, commanding focus (that is, a newly appearing piece of advice becomes the "current topic") and thus capable of taking elliptical responses. In the second case, Eucalyptus does not currently handle ellipticals containing prepositional phrases (*to 3*) but otherwise would recognize the phrase as a command to vector fighter 15, since it occurred in the context of several prior **Vector** operations.

(39) *Next.*  
*Accept.*

(40) *15 to 3.*

**Metaknowledge.** At one point, the user said (41) wishing to have the screen redrawn to clear a graphical glitch. While there is no "refresh" command, the knowledgeable operator can refresh the entire screen (not just a particular point) either by recentering or zooming it. In Eucalyptus, verbs like *redraw* or *refresh* could easily be added and mapped to the appropriate Interface Function.

(41) *Refresh this point.*

**Metalanguage.** The user made a few explicit references to the graphical interface itself (42). This is understandable since the user knew that the operator was another person viewing the screen, whereas in Eucalyptus the implicit agent the user is speaking to is the KOALAS system itself, which cannot "see" the mouse pointer icon. The system might be expected to know which fighter icons are colored green, however, and it is quite simple in Eucalyptus to just add *green* as having the same semantics as *out of weapons*, which is what that color denotes in the display.

(42) *Alter threat 6 over here to my pointer.*  
*Have all the green fighters return to the carrier.*

## RELATED WORK

Several other systems have been developed in recent years that provide integrated NL/graphical interfaces with deictic and other anaphoric capabilities: XTRA [24-27], Shoptalk [4, 28], CHORIS [29], CUBRICON [30-34], and the HITS user interfaces [35]. Of these, CHORIS and CUBRICON are interfaces to map-oriented applications and so resemble Eucalyptus in that regard. Other related systems of interest, including earlier seminal work in NL/graphical integration, can be found in Refs. 36 through 42.

XTRA allows deictic reference to the contents of fields in a tax computation form interfacing to an expert system, and provides selectable granularity of pointing gesture (exact, standard, vague, and encircling).

Shoptalk is an interface to a factory simulation that accepts deictic reference to mouse-selected icons representing pieces of product and equipment. As in Eucalyptus, NL phrases can be used to fill the fields of NL-like dialogue windows called Natural Language Form. Shoptalk supports use of anaphors in followup questions and also takes an innovative approach to the graphical identification of discourse contexts.

CHORIS is an intelligent multimedia interface that has been adapted to a geographical information system. It uses the same NL processor as Shoptalk and so supports deixis and graphical discourse context control much the same way.

CUBRICON is another geographical interface that, like Eucalyptus, allows deictic reference to icons and geometric points on a map; it also accepts deictic reference to windows and table entries, which Eucalyptus does not. In addition, CUBRICON allows so-called "multimodal NPs," where a pick gesture can take the place of a spoken NP in an utterance (unlike deixis, where the two must occur together). Eucalyptus disallows multimodal NPs because they can result in grammatically ill-formed utterances, which runs contrary to Eucalyptus's more highly NL-oriented stance.

The HITS (Human Interface Tool Suite) interface design uses a distinctive three-tier discourse representation for multimedia user dialogues. Unlike Eucalyptus and the other above-mentioned systems, it does not disambiguate graphical references using accompanying verbal descriptions, but only accepts one or the other. HITS provides a cognitively based, flexible scaffolding for resolving anaphora and handling incomplete inputs.

Many of these systems manage multimedia inputs by inserting textual representations of graphical selections into the NL string being parsed. Since Eucalyptus only allows graphical inputs to accompany NL inputs in deictic reference, it does not require a multimedia parser but just goes ahead and produces a (possibly ambiguous) semantic interpretation of the verbal input and waits until reference resolution time to disambiguate it against the (also possibly ambiguous) graphical references, much as it does in resolving ordinary anaphora.

## CONCLUSIONS

The objective in Eucalyptus was to demonstrate the thesis that graphical and NL interfaces have complementary strengths and that an integrated interface combining the two provides additional, natural interaction techniques. The KOALAS tool's original interface illustrates the GUI's strengths of transparency, lack of ambiguity, and ability to easily denote spatial information (in this case, radar screen positions). The NL interface illustrates the strengths of flexible reference, context-sensitive abbreviation, and freedom from visibility constraints. Finally, deictic reference combines the naturalness of graphical gesture with that of verbal description to provide a powerful interaction technique particularly useful in map-based applications.

The KOALAS tool's NL-like verb-argument mechanism for issuing fighter and threat commands was particularly suitable for integration with NL input. Indeed, it is likely the tool's designers chose to use that particular graphical interaction technique as a way of simulating the underlying interactive task of issuing orders to fighters, which would normally be done verbally were speech input available. Since NL is by its very nature communicative, it is least appropriate as an alternative input modality to graphical interfaces where direct manipulation is the most natural interaction technique (e.g., a paint program), becoming more relevant as the graphical interaction style becomes more high-level and command-oriented, the user telling the computer what to do rather than having the sensation of doing it him/herself.

The KOALAS tool's approach to graphical selection of radar screen objects (locating that object closest to the mouse coordinates that satisfies the semantic constraints of the expected argument) proved to be highly adaptable for use in deictic reference. The more common direct-manipulation approach of making the aircraft icons selectable objects by associating them with mutually exclusive mouse-sensitive regions would not have been suitable.

One strength of graphical interfaces is that a new user can explore the space of permissible functional inputs to an application just by traversing the interface's graphical presentations. NL interfaces, by contrast, are typically opaque to the user: what am I permitted to say to the system, what can we talk about? In Eucalyptus, the command menu labels suggest an acceptable vocabulary of imperative verbs, the

dialogue window argument structure suggests an acceptable syntax of verb arguments, and the nouns that can be used in those arguments (*fighter, F14, threat, Badger*, etc.) are also to be found in various labels, displays, and prompts throughout the graphical interface. The GUI in effect suggests to the user what sorts of things can be said to the system and how they can be said, an approach one might call WYSIWYCS – What You See Is What You Can Say, pronounced “wizzy-wicks” – that ameliorates the opacity problem (see Ref. 40 for early work in a similar vein).

Two commands, **Relieve** and **Alter**, were found to violate the WYSIWYCS approach by not adhering strictly to the syntax of the corresponding English verbs, and should be redesigned to yield a fully symmetric integrated interface. Similarly, the various widget types (check boxes, buttons, and choice items) in the graphical interface should probably have labels uniformly consisting of a verb and a noun phrase (when transitive), to suggest basic skeletal clauses for verbal input (e.g., **show aircraft trails, collect data, accept advice**). In the case of choice items, the displayed current value would suggest the optional second argument (*zoom screen to 2, change speed to 20*).

This approach also keeps the vocabulary and language model to a size that can be processed with acceptably low error rates by current speech recognition technology. The number of referenceable domain objects (typically 100 or fewer) and action types (about 80) in the KOALAS tool is also well matched to the technology. The introduction of flexible reference, however, greatly increases the input space and the chances of speech recognition errors through the confusion of similar sounding pronouns and determiners (e.g., *a, the, that, them, this, these, those*). If the speech recognition grammar could have accommodated relative clauses, recognition errors would have increased even further, but perhaps not to an uncomfortable level. Providing for several restricted types of elliptical input also pushed the speech recognition requirements considerably, but the overall system is still sufficiently accurate for comfortable use.

The ellipsis and anaphora handling capability of the discourse component increases the naturalness and ease of use of the NL interface considerably. Since the graphical interface generally prohibits more than one subdialogue from occurring at a time, the corresponding NL interface can get by for the most part with a simple discourse model employing only a single context space. In graphical interfaces where focus can shift from one window to another, however, mechanisms for directing NL input to particular subdialogues (windows) will be required.

Eucalyptus's requirement that all verbal exchanges with the system resemble an interpersonal NL dialogue is too constraining: the machine is a tool, not a person, and users may want to be quite terse with it whenever possible and only resort to full NL when they feel the need to explain or clarify something. They will thus use a much wider range of telegraphic, fragmentary, and ill-formed utterances with the machine than commonly occurs in interpersonal discourse [43]. This suggests that a combination of ordinary speech input (e.g., speaking menu items) and Eucalyptus-style spoken NL is desirable in multimodal interfaces, which may require cutting back on the amount of allowable elliptical input.

Finally, Eucalyptus's modular design proved its worth when the interface was successfully ported to an XView-based version of the KOALAS tool in less than one working day. The key to this success was the Interface Function module, representing KOALAS code needed by Eucalyptus but factored out from the specifics of the GUI's graphics routines.

## ACKNOWLEDGMENTS

The author thanks Manuel A. Perez, Stephanie S. Everett, and Helen M. Gigley for their helpful comments in reviewing this report, and acknowledges the contributions of James A. Ballas and Kay Schulze, as well as the participation of Midshipmen Matthew Feeny and David Montgomery and reserv-

ists LTCDR Douglas Dreyer and LT Victor Picarro, in the user interaction studies conducted by Lisa Achille at NRL's Human-Computer Interaction Laboratory.

## REFERENCES

1. P.J. Hayes, "Steps Towards Integrating Natural Language and Graphical Interaction for Knowledge-Based Systems," Proceedings 7th European Conference on Artificial Intelligence (ECAI-86), B. du Boulay and D. Hogg, eds., July 20-25, 1986, Brighton, U.K., North-Holland, pp. 543-552.
2. J.W. Sullivan and S. W. Tyler, eds., *Intelligent User Interfaces* (ACM Press, New York, 1991).
3. P.R. Cohen, J.W. Sullivan, M. Darymple, R.A. Gargan, D.B. Moran, J.L. Schlossberg, F.C.N. Pereira, and S.W. Tyler, "Synergistic Use of Direct Manipulation and Natural Language," Proceedings of CHI 89, K. Bire and C. Lewis, eds., April 30-May 4, 1989, ACM, Austin, TX, Addison-Wesley, pp. 227-233.
4. P.R. Cohen, "The Role of Natural Language in a Multimodal Interface," SRI International Technical Note 514, November 1991.
5. E. Marsh, "Towards Friendlier User Interfaces for Expert Systems," Proceedings of the 1991 World Congress on Expert Systems, Vol. 2, J. Liebowitz, ed., Dec. 16-19, 1991, Orlando, FL, Pergamon Press, pp. 1068-1076.
6. K. Wauchope, "Eucalyptus: An Integrated Spoken Language/Graphical Interface for Human-Computer Dialogue," Proceedings of the Voice I/O Systems Applications Conference (AVIOS '92), Sept. 22-24, 1992, American Voice I/O Society, Minneapolis, MN, pp. 71-76.
7. D. Perzanowski and B. Potter, "InterFIS: A Natural Language Interface to the Fault Isolation Shell," NRL Report 9299, December 1990.
8. C. Barrett and M. Donnell, "Real-time Expert Systems: Considerations and Imperatives," *Information and Decision Technologies* **16** (1990).
9. C. Barrett and C. Aldrich, "Final Report, KOALAS Test Planning Tool Concept Demonstration: Users Manual," Sept. 28, 1990, Los Alamos National Laboratory, Los Alamos, NM.
10. J. Gurney, E. Marsh, and K. Wauchope, "Focus of Attention in Decision Support Systems," Proceedings of the 10th Annual Conference on Command and Control Decision Aids, June 28 - July 1, 1993, National Defense University, Washington, DC.
11. S. E. Brennan, "Conversation as Direct Manipulation: An Iconoclastic View," in *The Art of Human-Computer Interface Design*, B. Laurel, ed. (Addison-Wesley, Reading, MA, 1990) pp. 393-404.
12. B. Shneiderman, C. Williamson, and C. Ahlberg, "Dynamic Queries: Database Searching by Direct Manipulation," Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 92), P. Bauersfeld, J. Bennett, and G. Lynch, eds., May 3-7, 1992, ACM, Monterey, CA, Addison-Wesley, pp. 669-670.
13. R. Grishman, *Computational Linguistics: An Introduction* (Cambridge University Press, Cambridge, 1986).

14. J. Allen, *Natural Language Understanding* (Benjamin Cummings, Menlo Park, CA, 1987).
15. R. Reichman-Adar, "Extended Person-Machine Interface," *Artificial Intelligence* 22(2), 157-218 (1984).
16. R. Grishman, "PROTEUS Parser Reference Manual," PROTEUS Project Memorandum #4, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, July 1986.
17. S.S. Everett, K. Wauchope, and D. Perzanowski, "Talking to InterFIS: Adding Speech Input to a Natural Language Interface," NRL/FR/5510-92-9515, Sept. 1992.
18. J.M. Gawron, "Syntactic Regularization in PROTEUS," PROTEUS Project Memorandum #5, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, March 1986.
19. K. Wauchope, "A Tandem Semantic Interpreter for Incremental Parse Selection," NRL Report 9288, Sept. 1990.
20. B.J. Grosz, "The Representation and Use of Focus in a System for Understanding Dialogs," Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77), Vol. 1, Aug. 22-25, 1977, Cambridge, MA, Carnegie Mellon Computer Science Department, Pittsburgh, pp. 67-76.
21. C. Sidner, "Focusing in the Comprehension of Definite Anaphora," in *Computational Models of Discourse*, M. Brady and R.C. Berwick, eds. (MIT Press, Cambridge, MA, 1983) pp. 267-330.
22. J. Hobbs, "Resolving Pronoun References," *Lingua* 44, 311-338 (1978).
23. R. Montague, "The Proper Treatment of Quantification in Ordinary English," in *Formal Philosophy: Selected Papers of Richard Montague*, R.H. Thomason, ed. (Yale University Press, New Haven, CT, 1974) pp. 188-221.
24. A. Kobsa, J. Allgayer, C. Reddig, N. Reithinger, D. Schmauks, K. Harbusch, and W. Wahlster, "Combining Deictic Gestures and Natural Language for Referent Identification," Proceedings of the 11th International Conference on Computational Linguistics (Coling '86), Aug. 25-29, 1986, Bonn, FR Germany, ACL Publishers, pp. 356-361.
25. J. Allgayer, K. Harbusch, A. Kobsa, C. Reddig, N. Reithinger, and D. Schmauks, "XTRA: A Natural-Language Access System to Expert Systems," *International Journal of Man-Machine Studies* 31, 161-195 (1989).
26. J. Allgayer, R. Janssen-Winkeln, C. Reddig, and N. Reithinger, "Bidirectional Use of Knowledge in the Multi-Modal NL Access System XTRA," Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89), Vol. 2, N.S. Sridharan, ed., Aug. 20-25, 1989, Detroit, Morgan Kaufmann, pp. 1492-1497.
27. W. Wahlster, "User and Discourse Models for Multimodal Communication," in *Intelligent User Interfaces*, J.W. Sullivan and S.W. Tyler, eds. (ACM Press, New York, 1991) pp. 45-67.

28. P.R. Cohen, "Adaptive Interfaces," Technical Report RADC-TR-91-27, Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Base, NY, 1991.
29. S.W. Tyler, "Adaptive Interfaces," Technical Report RADC-TR-90-276, Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Base, NY, 1990.
30. J.G. Neal, K.E. Bettinger, J.S. Byoun, Z. Dobes, and C.Y. Thielman, "An Intelligent Multi-Media Human-Computer Dialog System," Proceedings of the Second Annual Workshop on Space Operations, Automation, and Robotics (SOAR-88), S. Griffin, ed., July 20-23, 1988, Wright State University, Dayton, National Technical Information Service, pp. 245-251.
31. J.G. Neal, Z. Dobes, K.E. Bettinger, and J.S. Byoun, "Multi-Modal References in Human-Computer Dialogue," Proceedings of the AAAI Seventh National Conference (AAAI-88), Vol. 2, Aug. 21-26, 1988, St. Paul, MN, Morgan Kaufmann, pp. 819-823.
32. J.G. Neal, C.Y. Thielman, Z. Dobes, S.M. Haller, and S.C. Shapiro, "Natural Language with Integrated Deictic and Graphic Gestures," Proceedings of the DARPA Speech and Natural Language Workshop, Oct. 15-18, 1989, Cape Cod, MA, Morgan Kaufmann, pp. 410-423.
33. J.G. Neal, J.M. Lammens, Z. Dobes, S.C. Shapiro, D.J. Funke, S. Glanowski, C.Y. Thielman, J.S. Byoun, M.S. Summers, J.R. Gucwa, and R. Paul, "Intelligent Multi-Media Integrated Interface Project," Technical Report RADC-TR-90-128, Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Base, NY, 1990.
34. J. G. Neal and S.C. Shapiro, "Intelligent Multi-Media Interface Technology," in *Intelligent User Interfaces*, J.W. Sullivan and S.W. Tyler, eds. (ACM Press, New York, 1991) pp. 11-43.
35. S. Luperfoy, "The Representation of Multimodal User Interface Dialogues using Discourse Pegs," Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics, Newark, NJ, ACL Publishers, June 1992, pp. 192-201.
36. D.C. Brown, S.C. Kwasny, B. Chandrasekaran, and N.K. Sondheimer, "An Experimental Graphics System with Natural-Language Input," *Computer and Graphics* 4, 13-22 (1979).
37. R.A. Bolt, "Put-That-There: Voice and Gesture at the Graphics Interface," *Computer Graphics* 14, 262-270 (1980).
38. J.G. Carbonell, W.M. Boggs, M.L. Mauldin, and P.G. Anick, "First Steps Toward an Integrated Natural Language Interface," in *Applications in Artificial Intelligence*, S.J. Andriole, ed. (Petrocelli Books, Princeton, NJ, 1985), pp. 227-245.
39. B. Press, "The U.S. Air Force TEMPLAR Project Status and Outlook," Western Conference on Knowledge-Based Engineering and Expert Systems (WESTEX), Anaheim, CA, June 24-26, 1986, IEEE Computer Society Press, pp. 42-48.
40. C. Thompson, "Building Menu-Based Natural Language Interfaces," *Texas Engineering Journal* 3, 140-150 (1986).

41. E. Hinrichs and L. Polanyi, "Pointing the Way: A Unified Treatment of Referential Gesture in Interactive Discourse," Papers from the Parasession on Pragmatics and Grammatical Theory at the 22nd Regional Meeting, Chicago Linguistic Society, 1987, Chicago, pp. 298-314.
42. R.P. Wetzel, K.H. Hanne, and J.P. Hoepelmann, "DIS\_QUE: Deictic Interaction System-Query Environment," LOKI Report KR-GR 5.3/KR-NL 5, 1987, Fraunhofer Gesellschaft, IAO, Stuttgart, Germany.
43. N. Dahlbäck, A. Jönsson, and L. Ahrenberg, "Wizard of Oz Studies: Why and How," Proceedings of the 1993 International Workshop on Intelligent User Interfaces, W.D. Gray, W.E. Hefley, and D. Murray, eds., Jan. 4-7, 1993, Orlando, FL, ACM Press, pp. 193-200.



## Appendix

### DATA PROCESSING EXAMPLES

#### 1. SSI Phonetic Decoder output

are all the f fourteens moving to a station fighter 1 is holding

#### 2. PROTEUS intermediate syntactic representation

```
(REQUEST PRESENT PROG MOVE
  (ALL N1 F14 PLURAL)
  (TO (SOME N2 STATION SINGULAR
    (PRESENT PROG HOLD
      (NULL-DET N3 FIGHTER SINGULAR (IDNUM 1))
      VAR))))
```

#### 3. TINSEL semantic interpretation

```
(REQUEST PRESENT PROG V1 (:CLASS P-MOVE)
  (:PATIENT (ALL N1 (:CLASS P-F14 PLURAL)))
  (:TO-LOC (SOME N2 (:CLASS P-STATION) SINGULAR
    (PRESENT PROG V2 (:CLASS P-HOLD)
      (:PATIENT (NULL-DET N3 (:CLASS P-FIGHTER)
        SINGULAR (:ID 1)))
      (:AT-LOC N2))))))
```

#### 4. FOCAL alternate focus list

```
((N1 SINGULAR P-FIGHTER :PATIENT (FRIENDLY-1 FRIENDLY-2 ... FRIENDLY-9))
 (N2 SINGULAR P-STATION :TO-LOC (STATION-2))
 (N3 SINGULAR P-F14 :PATIENT (FRIENDLY-1)))
```

#### 5. FUNTRAN quantified expression

```
(FORALL X1 (SETOF N1 P-F14)
  (EXISTS X2 (SETOF N2 P-STATION
    (EXISTS! X3 (SETOF N3 P-FIGHTER (:ID 1))
      (P-HOLD :PATIENT X3 :AT-LOC N2)))
    (P-MOVE :PATIENT X1 :TO-LOC X2)))
```

#### 6. Quantified expression after performative insertion

```
(TELLIFALL
  (SETOF X1 (SETOF N1 P-F14)
    (EXISTS X2 (SETOF N2 P-STATION
      (EXISTS! X3 (SETOF N3 P-FIGHTER (:ID 1))
        (P-HOLD :PATIENT X3 :AT-LOC N2)))
      (P-MOVE :PATIENT X1 :TO-LOC X2)))
```

