CommandTalk: A Spoken-Language Interface for Battlefield Simulations

R. C. Moore, J. Dowding, H. Bratt, J. M. Gawron, Y. Gorfu, and A. Cheyer
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
{bmoore, dowding, harry, gawron, gorfu, cheyer}@ai.sri.com

Abstract

CommandTalk is a spoken-language interface to battlefield simulations that allows the use of ordinary spoken English to create forces and control measures, assign missions to forces, modify missions during execution, and control simulation system functions. CommandTalk combines a number of separate components integrated through the use of the Open Agent Architecture, including the Nuance speech recognition system, the Gemini natural-language parsing and interpretation system, a contextual-interpretation module, a "push-to-talk" agent, the ModSAF battlefield simulator, and "Start-It" (a graphical processing-spawning agent). CommandTalk is installed at a number of Government and contractor sites, including NRaD and the Marine Corps Air Ground Combat Center. It is currently being extended to provide exercise-time control of all simulated U.S. forces in DARPA's STOW 97 demonstration.

1 Overview

CommandTalk is a spoken-language interface to synthetic forces in entity-based battlefield simulations. The principal goal of CommandTalk is to let commanders interact with simulated forces by voice in a manner as similar as possible to the way they way they would command actual forces. CommandTalk currently interfaces to the ModSAF battlefield simulator and allows the use of ordinary English commands to

- Create forces and control measures (points and lines)
- Assign missions to forces
- Modify missions during execution
- Control ModSAF system functions, such as the map display

As an example, the following sequence of commands can be used to initialize a simple simulation in ModSAF and begin its execution:

Create an M1 platoon designated Charlie 4 5.

Put Checkpoint 1 at 937 965.

Create a point called Checkpoint 2 at 930 960.

Objective Alpha is 92 96.

Charlie 4 5, at my command, advance in a column to Checkpoint 1.

Next, proceed to Checkpoint 2.

Then assault Objective Alpha.

Charlie 4 5, move out.

With the simulation under way, the user can exercise direct control over the simulated forces by giving commands such as the following for immediate execution:

Charlie 4 5, speed up.

Change formation to echelon right.

Get in a line.

Withdraw to Checkpoint 2.

Examples of voice commands for controlling ModSAF system functions include the following:

Show contour lines.

Center on M1 platoon.

Zoom in closer.

Pan west 500 meters.

Center north of Checkpoint 2.

CommandTalk was initially developed for LeatherNet, a simulation and training system for the Marine Corps developed under direction of the Naval Command, Control and Ocean Surveillance Center, RDT&E Division (NRaD). In addition to CommandTalk, LeatherNet includes

- MCSF, a version of ModSAF customized for the Marine Corps
- CommandVu, a synthetic, data-enhanced environment with 3-D representation of MCSF behaviors and display of commander decision aids
- Terrain Evaluation Module (TEM), a system for line-of-sight and weapons coverage analysis

LeatherNet is intended to be used both as a training system for the Marine Corps and as the Marine Corps component of DARPA's Synthetic Theater of War (STOW) program. LeatherNet is currently installed at the Marine Corps Air Ground Combat Center (MCAGCC), at Twentynine Palms, California.

A single CommandTalk interacts directly with only one ModSAF process. ModSAF, however, creates distributed simulations that can include multiple graphical user interface (GUI) processes and multiple simulator processes, plus other applications such as CommandVu, communicating over a network through Distributed Interactive Simulation (DIS) and Persistent Object (PO) protocols. This architecture lets CommandTalk interact indirectly with all these components. Thus, a user can control a simulation using CommandTalk while viewing it in 3-D via CommandVu, without having to be aware of the ModSAF processes that mediate between the spoken commands and their results as seen in the 3-D display.

2 Architecture

CommandTalk combines a number of separate components, developed independently, some of which are implemented in C and others in Prolog. These components are integrated through the use of the Open Agent Architecture (OAA) (Cohen et al., 1994). OAA makes use of a facilitator agent that plans and coordinates interactions among agents during distributed computation. Other processes are encapsulated as agents that register with the facilitator the types of messages they can respond to. An agent posts a message in an Interagent Communication Language (ICL) to the facilitator, which dispatches the message to the agents that have registered their ability to handle messages of that type. This mediated communication makes it possible to "hot-swap" or restart individual agents without restarting the whole system. The ICL communications mechanism is built on top of TCP/IP, so an OAA-based system can be distributed across both local- and wide-area networks based on Internet technology. OAA also provides an agent library to simplify turning independent components into agents. The agent library supplies common functionality to agents in multiple languages for multiple platforms, managing network communication, ICL parsing, trigger and monitor handling, and distributed message primitives.

CommandTalk is implemented as a set of agents communicating as described above. The principal agents used in CommandTalk are

- Speech recognition
- Natural language
- Contextual interpretation
- Push to talk
- ModSAF
- Start-It

2.1 Speech Recognition

The speech recognition (SR) agent consists of a thin agent layer on top of the Nuance (formerly Corona) speech recognition system. Nuance is a commercial speech recognition product based on technology developed by SRI International. The recognizer listens on the audio port of the computer on which it is running, and produces its best hypothesis as to what string of words was spoken. The SR agent accepts messages that tell it to start and stop listening and to change grammars, and generates messages that it has stopped listening and messages containing the hypothesized word string.

The Nuance recognizer is customized in two ways for use in CommandTalk. First, we have replaced the narrow-band (8-bit, 8-kHz sampled) acoustic models included with the Nuance recognizer and designed for telephone applications, with wide-band (16-bit, 16-kHz sampled) acoustic models that take advantage of the higher-quality audio available on computer workstations. Second, any practical application of speech recognition technology requires a vocabulary and grammar tailored to the particular application, since for high accuracy the recognizer must be restricted as to what sequences of words it will consider. To produce the recognition vocabulary and grammar for CommandTalk, we have implemented an algorithm that extracts these from the vocabulary and grammar specifications for the natural-language component of CommandTalk. This eases development by automatically keeping the language that can be recognized and the language that can be parsed in sync; that is, it guarantees that every word string that can be parsed by the natural-language component is a potential recognition hypothesis, and vice versa. This module that generates the recognition grammar for CommandTalk is described in Section 3.

2.2 Natural Language

The natural-language (NL) agent consists of a thin agent layer on top of Gemini (Dowding et al., 1993, 1994), a natural-language parsing and semantic interpretation system based on unification grammar. "Unification grammar" means that grammatical categories incorporate features that can be assigned values; so that when grammatical category expressions are matched in the course of parsing or semantic interpretation, the information contained in the features is combined, and if the feature values are incompatible the match fails. Gemini applies a set of syntactic and semantic grammar rules to a word string using a bottom-up parser to generate a *logical form*, a structured representation of the context-independent meaning of the string. The NL agent accepts messages containing word strings to be parsed and interpreted, and generates messages containing logical forms or, if no meaning representation can be found, error messages to be displayed to the user.

Gemini is a research system that has been developed over several years, and includes an extensive grammar of general English. For CommandTalk, however, we have developed an application-specific grammar, which gives us a number of advantages. First, because it does not include rules for English expressions not relevant to the application, the grammar runs faster and finds few grammatical ambiguities. Second, because the semantic rules are tailored to the application, the logical forms they generate require less subsequent processing to produce commands to the application system. Finally, by restricting the form of the CommandTalk grammar, we are able to automatically extract the grammar that guides the speech recognizer.

The Nuance recognizer, like all other practical recognizers, requires a grammar that defines a finite-state language model. The Gemini grammar formalism, on the other hand, is able to define grammars of much greater computational complexity. For CommandTalk, extraction of the recognition grammar is made possible by restricting the Gemini syntactic rules to a finite-state backbone with finitely valued features. It should be noted that, although we are not using the full power of the Gemini grammar formalism, we still gain considerable benefit from Gemini because the feature constraints let us write the grammar much more compactly, Gemini's morphology component simplifies maintaining the vocabulary, and Gemini's unification-based semantic rules let us specify the translation from word strings into logical forms easily and systematically.

2.3 Contextual Interpretation

The contextual-interpretation (CI) agent accepts a logical form from the NL agent, and produces one or more commands to ModSAF. Since a logical form encodes only information that is directly expressed in the utterance, the CI agent often must apply contextual information to produce a complete interpretation. Sources of this information can include linguistic context, situational context, and defaults. Since ModSAF itself is the source of situational information about the simulation, the interaction between the CI agent and ModSAF is not a simple one-direction pipeline. Often, there will be a series of queries to ModSAF about the current state of the simulation before the ModSAF command or commands that represent the final interpretation of an utterance are produced.

Some of the problems which must be solved by the CI agent are

- Noun phrase resolution
- Predicate resolution
- Temporal resolution
- Vagueness resolution

2.3.1 Noun Phrase Resolution

A noun phrase denoting an object in the simulation must be resolved to the unique ModSAF identifier for that object. "M1 platoon," "tank platoon," or "Charlie 4 5" could all refer to the same entity in the simulation. To keep the CI informed about the objects in the simulation and their properties, the ModSAF agent notifies the CI agent whenever an object is created, modified, or destroyed. Since the CI agent is immediately notified whenever the user creates an object through the ModSAF GUI, the CI can note the salience of such objects, and make them available for pronominal reference (just as objects created by speech are), leading to smoother interoperation between speech and the GUI.

2.3.2 Predicate resolution

While users employ generic verbs like move, attack, and assault to give verbal commands, the corresponding ModSAF tasks often differ depending on the units involved. The ModSAF movement task for a tank platoon is different from the one for an infantry platoon or the one for a tank company. Similarly, the parameter value indicating a column formation for tanks is different from the one indicating a column formation for infantry, and the parameter that controls the speed of vehicles has a different name than the one that controls the speed of infantry. All these differences need to be taken into account when generating the ModSAF command for something like "Advance in a column to Checkpoint 1 at 10 kph," depending on what type of unit is being given the command.

2.3.3 Temporal resolution

The CI agent needs to determine when a command is given to a unit should be carried out. The command may be part of a mission to be carried out later, or it may be an order to be carried out immediately. If the latter, it may be a permanent change to the current mission, or merely a temporary interruption of the current task in the mission, which should be resumed when the interrupting task is completed. The CI agent decides these questions based on a combination of phrasing and context. Sometimes, explicit indicators may be given as to when the command is to be carried out, such as a specific time, or after a given duration of time has elapsed, or on the commander's order.

2.3.4 Vagueness resolution

Sometimes a verbal command does not include all the information required by the simulation. The CI agent attempts to fill in this missing information by using a combination of linguistic and situational context, plus defaults. For instance, if no unit is explicitly addressed by a command, it is assumed that the addressee is the unit to whom the last verbal command was given. The ModSAF "occupy position" and "attack by fire" tasks require that a line be given as a battle position, but users often give just a point location for the position of the unit. In such cases, the CI agent calls ModSAF to construct a line through the point, and uses that line for the battle position.

2.4 Push to Talk

The push-to-talk (PTT) agent manages the interactions with the user. It provides a long narrow window running across the top of the screen—the only visible indication that a ModSAF is CommandTalk-enabled. This window contains a microphone icon that indicates the state of CommandTalk (ready, listening, or busy), an area for the most recent recognized string to be printed, and an area for text messages from the system to appear (confirmation messages and error messages).

This agent provides two mechanisms for the user to initiate a spoken command. A push-to-talk button attached to the serial port of the computer can be pushed down to signal the computer to start listening and released to indicate that the utterance is finished (push-and-hold-to-talk). The second option is to click on the microphone icon with the left mouse button to signal the computer to start listening (click-to-talk). With click-to-talk, the system listens for speech until a sufficiently long pause is detected. The length of time to wait is a parameter that can be set in the recognizer. The push-and-hold method generally seems more satisfactory for a number of reasons: Push-and-hold leads to faster response because the system does not have to wait to hear whether the user is done speaking, click-to-talk tends to cut off users who pause in the middle of an utterance to figure out what to say next, and push-and-hold seems natural to military users because it works like a tactical field radio.

The PTT agent issues messages to the SR agent to start and stop listening. It accepts messages from the SR agent containing the words that were recognized, messages that the user has stopped speaking (for click-to-talk), and messages, from any agent, that contain confirmation or error messages to be displayed to the user.

2.5 ModSAF

The ModSAF agent consists of a thin layer on top of ModSAF. It sends messages that keep the CI agent informed of the current state of the simulation and executes commands that it receives from the CI agent. Generally, these commands access functions that are also available using the GUI, but not always. For example, it is possible with CommandTalk to tell ModSAF to center its map display on a point that is not currently visible. This cannot be done with the GUI, because there is no way to select a point that is not currently displayed on the map. The set of messages that the ModSAF agent responds to is defined by the ModSAF Agent Layer Language (MALL).

2.6 Start-It

Start-It is a graphical processing-spawning agent that helps control the large number of processes that make up the CommandTalk system. It provides a mouse-and-menu interface to configure and start other processes. While it is particularly useful for starting agent processes, it can also be used to start nonagent processes such as additional ModSAF simulators and interfaces, CommandVu, and the LeatherNet sound server. Features of Start-It include the following:

- It makes it easy to assign processes to machines distributed over a network.
- It reports process status (not running, initializing, running, or dead).
- It makes it easy to set command line arguments and maintain consistent command line arguments across processes.
- The Start-It configuration is data-driven, so it is easy to add processes and command line arguments, or change default values.
- An automatic restart feature keeps agents running in case of machine failure or process death.

3 Gemini-to-Nuance Grammar Compiler

The SR agent requires a grammar to tell the recognizer what sequences of words are possible in a particular application, and the NL agent requires a grammar to specify the translation of word strings into logical forms. For optimal performance, these two grammars should, as nearly as possible, accept exactly the same word sequences. In general, we would like the recognizer to accept all word sequences that can be interpreted, and any overgeneration by the recognition grammar increases the likelihood of recognition errors without providing any additional functionality. In order to keep these two grammars synchronized, we have implemented a compiler that derives the recognition grammar automatically from the NL grammar.

To derive a recognition grammar with coverage equivalent to the NL grammar, we must restrict the form of the NL grammar. Like virtually all practical speech recognizers, the Nuance recognizer requires a finite-state grammar, while the Gemini parser accepts grammars that have a context-free backbone, plus unification-based feature constraints that give Gemini grammars the power of an arbitrary Turing machine. To make it possible to derive an equivalent finite-state grammar, we restrict the Gemini grammars used as input to our Gemini-to-Nuance compiler as follows:

- All features in the Gemini grammar that are compiled into the recognition grammar must allow only a finite number of values. This means that no feature values are structures that can grow arbitrarily large.
- The Gemini grammar must not contain any indirect recursion. That is, no rule subsets are allowed with patterns such as $A \rightarrow BC$, $C \rightarrow AD$.
- Immediately recursive rules are allowed, but only if the recursive category is leftmost or rightmost in the list of daughters, so that there is no form of center embedding. That is, $A \rightarrow AB$ and $A \rightarrow CA$ are allowed (even simultaneously), but not $A \rightarrow CAB$.

There are many possible formats for specifying a finite-state grammar, and the one used by the Nuance recognition system specifies a single definition for each atomic nonterminal symbol as a regular expression over vocabulary words and other nonterminals, such that there is no direct or indirect recursion in the set of definitions. To transform a restricted Gemini grammar into this format, we first transform the Gemini rules over categories with feature constraints into rules over atomic symbols, and we then transform these rules into a set of definitions in terms of regular expressions.

3.1 Generating Atomic Categories

Given the restriction that all features must allow only a finite number of values, it would be trivial to transform all unification rules into rules over atomic categories by generating all possible full feature instantiations of every rule, and making up an atomic name for each combination of category and feature values that occur in these fully-instantiated rules. This would, however, increase the total number of rules to a size that would be too large to deal with. We therefore instantiate the rules in a more careful way that avoids unnecessarily instantiating features and prunes out useless rules.

The set of atomic categories is defined by considering, for each daughter category of each rule, all instantiations of just the subset of features on the daughter that are constrained by the rule. Thus, if there is a rule that does not constrain a feature on a particular daughter category, an atomic category will be created for that daughter that is underspecified for the value of that feature. A prime example of this in the CommandTalk grammar is the rule

```
coordinate_nums:[] →
   digit:[] digit:[] digit:[] digit:[]
```

which says that a set of coordinate numbers can be a sequence of four digits. In the CommandTalk grammar the digit category has features (singular vs. plural, zero vs. nonzero, etc.) that would generate at least 60 combinations if all instantiations were considered. So, if we naively generated all possible complete instantiations of this rule, we would get at least 60^4 rules. Even worse, we need other rules to permit up to eight digits to form a set of coordinate numbers, which would give rise to 60^8 rules. Since the original rule, however, puts no constraints on any of the features of the digit category, by generating an atomic category that is under-specified for all features, we only need a single rule in the derived grammar.

From the set of atomic categories defined in this way, we generate all rules consistent with the original Gemini rules, except that for daughters that have unconstrained features, we use only the corresponding under-specified categories. We then iteratively remove all rules that cannot participate in a complete parse of an utterance, either because they contain daughter categories that cannot be expanded into any sequence of words, given the particular lexicon we have, or because they have a mother category that cannot be reached from the top category of the grammar.

3.2 Compiling Rules to Regular Expressions

Once we have transformed the Gemini unification grammar into an equivalent grammar over atomic nonterminals, we then rewrite the grammar as a set of definitions of the nonterminals as regular expressions. For the nonterminals that have no recursive rules, we simply collect all the rules with the same left-hand side and create a single rule by forming the disjunction of all the right-hand sides. For example, if the only rules for the nonterminal A are

$A \rightarrow BC$
$A \rightarrow DE$

then the regular expression defining A would be $[(BC)(DE)]$. In the Nuance regular expression notation, "( )" indicates a sequence and "[ ]" indicates a set of disjunctive alternatives.

For nonterminals with recursive rules, we eliminate the recursion by introducing regular expressions using the Kleene star operator. For each recursive nonterminal A, we divide the rules defining A into right-recursive, left-recursive, and nonrecursive subsets. For the right-recursive subset, we form the disjunction of the expressions that occur to the *left* of A. That is, for the rules

$A \rightarrow BA$
$A \rightarrow CA$

we generate $[BC]$. Call this expression *LEFT-A*. For the left-recursive subset, we form the disjunction of the expressions that occur to the *right* of A, which we may call *RIGHT-A*. Finally, we form the disjunction of all the right-hand sides of the nonrecursive rules, which we may call *NON-REC-A*. The complete regular expression defining A is then

(**LEFT-A NON-REC-A *RIGHT-A*)

In the Nuance regular expression notation, the Kleene star operator "*" precedes the iterated expression, rather than following it as in most notations for regular expressions. Thus, $*X$ means that a sequence of zero or more instances of X may occur.

As an example, suppose the rules defining the nonterminal A are

$A \rightarrow BA$
$A \rightarrow CDA$
$A \rightarrow E$
$A \rightarrow FG$
$A \rightarrow AH$

The corresponding regular expression defining A would be

$(*[B(CD)]\ [E(FG)]\ *H)$

This completes the transformation of a Gemini grammar with finitely-valued categories and a finite-state backbone into a Nuance regular expression grammar. However, as one final optimization, we look for special cases where we can use the "Kleene plus" operator, which indicates one or more instances of an expression in sequence, and which is handled more efficiently by the Nuance recognizer than equivalent expressions using Kleene star. We simply look for sequences of the form $(*X\ X)$ or $(X\ *X)$, and replace them with $+X$.

4 Development History

Work on CommandTalk began with SRI's initial receipt of MCSF on February 16, 1995. The first demonstration of spoken commands to simulated forces in MCSF was given three weeks later on March 7; an initial version the CommandTalk prototype was installed at the Marine Corps Air Ground Combat Center (MCAGCC) on May 1; and a demonstration of CommandTalk was given to General Palm, the Commanding Officer of MCAGCC, on May 16.

Enhanced versions of the system were demonstrated at DARPA's Software and Intelligent Systems Symposium in August 1995, and evaluated in the STOW ED-1 milestone test in October 1995. In the evaluation of ED-1 performance, CommandTalk was given the highest grade of any Marine Corps portion of the exercise. In addition to these milestones, CommandTalk has been included in demonstrations of LeatherNet to numerous VIPs including General C. C. Krulak, Commandant of the Marine Corps; General J. H. Binford Peay, Commander in Chief US Central Command; Secretary of the Navy J. H. Dalton; and Secretary of Defense William Perry.

CommandTalk is currently being extended to provide exercise-time control of all simulated U.S. forces in DARPA's STOW 97 Advanced Concept Technology Demonstration.

5 Availability

CommandTalk executables for Sun SPARC/SunOS and SGI MIPS/IRIX platforms are available at no cost to US Government users under Restricted Rights. Contractors may obtain CommandTalk in executable form exclusively for use on US Government projects under license from SRI. Distribution of CommandTalk for Government purposes is handled by NRaD (POC: Brenda Gillcrist, bwgill@nosc.mil). Other inquiries about CommandTalk, including licensing, should be directed to SRI (POC: Robert Moore, bmoore@ai.sri.com).

6 Acknowledgements

This work was supported by the Defense Advanced Research Projects Agency under Contract N66001–94–C–6046 with the Naval Command, Control, and Ocean Surveillance Center. Approved for Public Release – Distribution Unlimited

References

Cohen, P. R., A. J. Cheyer, M. Wang, and S. C. Baeg (1994) "An Open Agent Architecture," in *Working Notes, AAAI Spring Symposium Series, Software Agents*, Stanford, California, pp. 1–8.

Dowding, J., J. M. Gawron, D. Appelt, J. Bear, L. Cherny, R. Moore, and D. Moran (1993) "Gemini: A Natural Language System for Spoken-Language Understanding," in *Proceedings 31st Annual Meeting of the Association for Computational Linguistics*, Columbus, Ohio, pp. 54–61.

Dowding, J., R. Moore, F. Andry, and D. Moran (1994) "Interleaving Syntax and Semantics in an Efficient Bottom-Up Parser," in *Proceedings 32nd Annual Meeting of the Association for Computational Linguistics*, Las Cruces, New Mexico, pp. 110–116.