TRAINS as an Embodied Natural Language Dialogue System

James F. Allen, George Ferguson, Brad Miller and Eric Ringger
Department of Computer Science
University of Rochester
Rochester, NY 14627
james@cs.rochester.edu

The TRAINS project is a long-term effort to build a system that can interact and collaborate with humans in problem solving tasks, combining situation assessment, interactive plan construction, evaluation and execution. TRAINS-95 is an instance of the general architecture for mixed-initiative specification of routes using a simple map. The system is robust and can be used by novice users to solve problems with essentially no training. Over 50 different people have used the system so far, with a large percentage of them successfully completing their assigned tasks. This paper considers the TRAINS system as an embodied natural language system.

While one naturally thinks of language when speaking of dialogue, we do not restrict ourselves to language only, just as humans do not restrict themselves to language, and use gesture, charts, pictures and pointing when communicating. The TRAINS-95 system supports a variety of communication modalities, including spoken and keyboard natural language and direct manipulation of a graphical user interface. To us, the crucial aspects of dialogue are those aspects that make it an efficient collaboration mechanism between intelligent agents, namely:

> the incremental development of ideas, goals, and solutions;
>
> the interpretation of current communicative act in the context of the previous acts;
>
> confirmation, correction and rejection mechanisms;
>
> using the interaction itself to manage any problems in the interaction, as in clarifications and restarts.

The TRAINS system is embodied in the task it performs. It is irrelevant for our purposes whether the external world is actual or simulated, as the interaction of the system with its "environment" is the same in each case. One key part of the external world remains constant - there is always a human manager with whom the system interacts. The communication with the manager and communication with the agents in the external world completely define the TRAINS environment.

The task/environment provides both the motivation for system utterances, and the environment for interpreting user utterances. One could not evaluate the system outside of the task it performs. Unlike in a QA application, one cannot define a correct response to many interactions - rather responses can be judge as to how appropriate they are given the task, and how well they further the task. I want to make two general points about how the environment relates to the system, and then give a brief description of the system as it is currently operational.

Issue 1: Why build the system?

The task provides the raison d'être for the system. It provides the motivation for why the system would participate in a dialogue at all - it has to in order to perform well at its task. Note that this connection between the system's behavior and its task need not be captured explicitly in the system's knowledge. For example, an insect does not need to know that certain behaviors have a survival advantage in order to act in an effective way. But our knowledge of the connection allows us to recognize and evaluate how certain behaviors help the insect survive. Without the goal of survival, we would have no way of evaluating whether the insects behavior was effective or not. TRAINS-95, likewise, has little explicit knowledge linking its behavior to its task, but the task provides the framework for discussing and evaluating the system's performance and driving further development.

Issue 2: How well does the system do?

This brings us to the second issue: evaluation. One of the major problems that has plagued research in natural language understanding has been in defining what the word "understanding" means. As a result, much work remains almost impossible to evaluate because the goal is not precisely defined. We don't require that TRAINS-95 "understand", all it need do is be effective at its planning task. This suggests some obvious performance-based metrics for evaluating the system, such as

- time needed to completely specify the plan,
- plan-based metrics, e.g., expected execution time, probably of success, amount of resources consumed;
- user satisfaction.

Rather than attempting to characterize what a good response to an utterance is in the abstract, and evaluating the system against this metric, we can use the performance measures over numbers of users to compare the effectiveness of different processing strategies.

For example, a common problem in the TRAINS-95 dialogues is the treatment of fragmented utterances

because of speech recognition errors and other complications such as out-of-vocabulary words. For instance, the system might received as input the words

xxx xxx to Bath

where "xxx" is some uninterpretable input. There is a wide range of things the manager might have said here, such as "how far is it to Bath?" "and go on to Bath", and "don't go to Bath". How should one deal with such input? Three general approaches might be considered: 1) we could do nothing and ignore the input, 2) we could always go back to the user and ask them to restate the utterance, or 3) we could ignore the unknown words and attempt to interpret the phrases present in the current context, taking the risk of misinterpreting the utterance significantly. Which approach is better? How would we tell? Our answer is to implement all three options and try them out on a range of users. If one approach is better than the others, then we should be able to detect this when looking at the performance metrics over a range of users. It's hard to even think of an alternate way to address this problem, as there is no such thing as a "right" answer that can be effectively argued for in the abstract. The issue is what approach is the most effective dialogue strategy, and we get to that issue via the task measurements.

An Overview of the TRAINS-95 System

The goal of TRAINS-95 was to use the simplest scenarios and tasks that would produce interesting dialogue behavior. Our aim was that TRAINS-95 be robust and natural enough that it can "run its own demos". Basically, a person should need no prior instruction about the system or guidance when using the system. We now have constructed a first version of such a system and have demonstrated it at several conferences and workshops (1995). We collected all the data from these sessions and are using this information both to improve the language understanding and to develop more effective collaborative problem solving strategies.

The task for TRAINS-95 is simple route planning problems using a simple map (e.g., see Figure 1). The user is given a randomly generated initial scenario and a goal by the supervisor module, which is considered external to the TRAINS system. Typically, the goal involves moving three trains to specific goal locations.

The current version of the TRAINS-95 system consists of the following components:

- the SPHINX-II speech system (Huang et al, 1993) trained on ATIS data, giving 1-best output;
- the TRUETALK speech synthesis system;
- a fairly comprehensive TRAINS grammar and lexicon built from the TRAINS dialogue corpus;
- a robust understanding system based on a bottom-up chart parser with monitors;
- a speech-act based dialogue model (used for both understanding and generation)

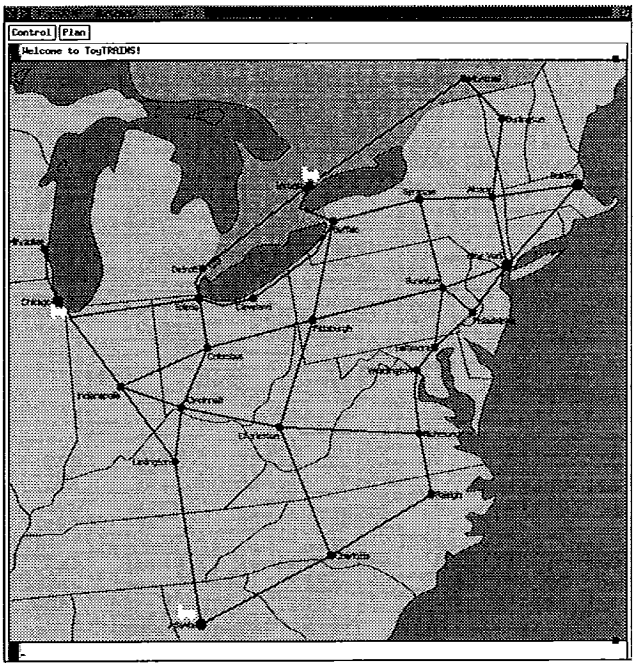

Figure 1: A Northeast Scenario for TRAINS-95

- an X-interface display with an abstract command language for realizing speech acts graphically;
- a deliberately weak route planner.

The route planner is deliberately weak in order to encourage interaction. Otherwise, as soon as the system identified the goal, it could provide the optimal solution. The planner is restricted so that it cannot plan routes between cities more than four hops apart, and it randomly generates a route from all possible routes (of length 4 or less) for cities that are close enough. Thus user has to incrementally develop their plan throughout the dialogue. A simple dialogue collected with the actual system is in Figure 2, where S is the system, and M is the manager. The supervisor picks the manager's goal to get trains to Cleveland, Syracuse and Montreal.

The dialogue shows that the system has a basic conversational capability: generating appropriate confirmations and participating in the give-and-take of the route development. It also demonstrates a natural integration of the language generation with the graphic display. We will point out a few examples below that show how the system can robustly operate even in the presence of significant speech recognition errors.

The remainder of the paper gives a brief overview of the system (for more details, see Allen et al, 1995).

Parsing and Speech Act Interpretation

The TRAINS parser must be both robust against speech recognition errors, yet produce a detailed semantic analysis of the input when possible. It does this by combining ideas from several different sources. For effective disambiguation, the grammar classifies each constituent both by syntactic and semantic categories.

Supervisor: *Welcome to TRAINS-95 version 1.2. Please wait a moment while I set up a scenario for you. ... The engines are currently at Chicago, Atlanta and Philadelphia, your goal is to get them to Cleveland, Syracuse and Montreal.*

S: *Okay. I think I'm ready to start.*

M: *Okay. I need to get to Cleveland, Syracuse and Montreal.*

S: Yes <<highlights the three cities>>

M: *Okay. let's send the train from Chicago to Cleveland.*

S: *OK, the terminal at Toledo is delaying traffic due to congestion. Trains will take an extra five hours to move through there.* <<displays route>>

M: *That's Okay. Now let's go from Atlanta up to Syracuse.*

S: *Tell me a route to get from Atlanta to Syracuse please.*

M: *Let's go via Cincinnati and Pittsburgh.*

S: *Yes. City Cincinnati is congested due to unusually heavy traffic. Trains will take an additional five hours to move through there.* <<displays route>>

M: *Okay. Let's go via Charleston and Buffalo instead.*

S: *Yes* <<displays revised route>>

M: *That's great.*

S: *I hope it was helpful.*

M: *Finally, I need to go from Philadelphia to Montreal.*

S: *OK* <<displays route to Montreal>>

M: *That's great. I'm done.*

S: *Please confirm that you are done using the dialogue box.* <<displays dialogue box>>

M: clicks OK button

S: *You Betcha. You'll be hearing from the scoring module shortly.*

Supervisor: *Okay. Wow. You made it to every city doing better than I would have done. You took three minutes and made nine interactions with the system. I expected you to take four minutes and make fourteen interactions for this level of problem. I rated this problem as difficult which gives you a score of 1322 with an average score of 100 - a stupendous score.*

Figure 2: Part of a dialogue with TRAINS-95

Thus, rules in the grammar mix syntactic and semantic constraints. While our grammar shares many of the advantages of semantic grammars, few of our semantic categories are domain specific. Rather, they deal with semantic constructs that underlie much of everyday language, including semantic concepts such as agents, events, actions, times, locations, units and measures, static and movable objects, plans and goals, and so on. The parser operates independently of the context of the dialogue, keeping its interpretations close to that indicated by the surface form.

The parser is a bottom-up chart parser, as described in (Allen, 1994). Because a single utterance may involve several speech acts, the chart output of the parser must be converted into a sequence of speech acts.

The robust parser pays particular attention to extracting information about speech acts. There are two basic mechanisms for recording information about speech acts. First, it uses rules that associate speech acts with the syntactic and semantic structure of the utterances. It also looks for lexical clues for the speech act intended. For example, an utterance that contains *please* would be flagged as a request, and an utterance that begins with *I want to ...* is flagged as a goal statement. If there is no full parse, the lexically-based pattern may still successfully identify the speech act. For example, in one session the manager actually said *Let's try going via Cleveland and Syracuse instead*, but the speech recognizer produced the output *Let's try going feed equivalent answer reduced instead*. While virtually none of this input is interpretable, the parser detected the use of the phrase *Let's try going* and the word *instead* at the end of the utterance, which suggests that the speech act was a rejection (or partial rejection) of the previous proposal. The system responded based on this interpretation by offering an alternate route, a good continuation given how little of the utterance was correctly recognized.

This demonstrates an important point about robustness. If the system can identify the intended speech act, it can often produce a reasonable continuation even if little of the content of the utterance is understood.

When faced with a fragment such as *to Bath*, the parser produces a generic TELL speech act. This allows the rest of the system to specialize the interpretation based on its knowledge of the state of the world and the discourse context.

The Dialogue Manager

Once past the parser, each speech act is analyzed with respect to the current task and discourse context. It tries to associate descriptions with specific objects in the database, or turns them into patterns that can be used for matching later. Paths that have been mentioned are turned into path description structures, which include specific constraints that the generated route must meet. It also adds or changes speech act interpretations, usually due to strategies that can better classify items because of its world knowledge or knowledge of the last utterance.

At this point, the set of speech acts is passed through a discourse manager, which attempts to track the state of the dialogue with respect to a simple stack-based segmentation scheme. This representation allows us to distinguish between the case where a segment is temporarily suspended and may be resumed later and the case where the segment is closed for good. Segments are especially important for handling corrections, as they identify information that typically is accepted, rejected or modified as a unit. For example, the simplest strategy for

handling a reject speech act such as *No*, is simply to remove all the information conveyed in the last segment.

The discourse manager, after updating it's dialogue state, passes on the production to the verbal reasoner. The verbal reasoner has a subsumption architecture to handle each speech act, which allows it to robustly deal with those that are only partially specified. From the information it is able to combine, it passes off a more specific request to the domain reasoner, such as to extend the route of a specific train (e.g. using the focus from prior generations), such that a path object is satisfied. The domain reasoner either chooses a route from a set that satisfies the path constraints, or returns an error as to why one could not be found. The verbal reasoner can then take this result, and generates a set of speech acts to communicate the effect back to the user. This is passed back to the dialogue manager (who again updates its dialogue state), and then on to the generator.

Speech Act realization: The Generator

The system typically produces speech acts in response to certain discourse situations, for example

- a confirm if the user's last utterance was understood and acceptable,
- a reject if there were problems, possibly with a clarification about the difficulty,
- an inform if the system is proposing a new route,
- a warning if unexpected problems might arise (e.g., routes cross, or cities are congested).

The generator basically handles each of these one at a time, but both scans forward for related information, and back for information it has already handled. It puts together a series of actions for the display which can include objects to be highlighted (and how), utterances to be made and/or routes to be drawn (or erased). It has databases of the current screen contents, as well as things it has previously informed the user about. Because most acts have a series of ways that can be used to convey the information to the user, the generator picks randomly between them for variety. It also uses the general "feel" of the set of speech acts to be handled to decide how confident it is in the result. A single confirm will be stated strongly (e.g. as "Certainly."), but one that is followed by an elaboration of an error will lower it's confidence, and cause it to insert dialogue to this effect (such as, if displaying a route, it will say "Well, is this all right?" or OK, but I don't understand ...").

Lastly, the display updates are sent to update the X interface and to generate the speech response.

The Graphics and Speech Interface

The display component of TRAINS-95 is a general purpose object-oriented system implemented in C and using the X Window System. A hierarchy of object types and their attributes are defined using C structures with inheritance. Most objects have a "shape" component used to render them on the screen and to detect when they are selected with the mouse. Communication to and from the display component is via UNIX standard input and output, providing separation between components to allow for rapid prototyping and reuse of the component(s). The display component understands a powerful command language using a context-free parser implemented using the Bison compiler-compiler. Commands are provided to create and destroy objects, modify their attributes, and perform display-related effects (e.g., highlighting).

The speech recognition component of TRAINS-95 is based on the Sphinx II recognizer from CMU running a language model trained for the ATIS task. The interface currently provides a "push-to-talk" metaphor and displays the results of the recognition as they are computed by Sphinx II. Words are output incrementally as they are recognized, with control tokens used when Sphinx "backs up" its hypothesis. As with the display component, communication with other components is via UNIX standard input and output.

THE FUTURE

The TRAINS-95 system first ran in late January, 1995. In the short term, we are continuing to make the system more robust by enhancing the dialogue interactions and extending the grammar and the interface. We plan extensive testing with this simple system, using the data collected to drive our research.

In the slightly longer term, we are building a new planning component that can reason about plans of greater complexity, reasoning about shipments of personnel and equipment using multiple modes of transportation.

References

Allen, J., Ferguson, G., Miller, B. and Ringger, E. Spoken Dialogue and Interactive Planning *Proc. ARPA Spoken Language System Workshop*, Jan. 1995. (Morgan Kaufmann)

Allen, J. et al, The TRAINS Project: a case study in building a conversational planning agent, Journal of Experimental and Theoretical AI, 1995.

Ferguson, G. Knowledge Representation and Reasoning for Mixed-Initiative Planning, Technical Report 562 and Ph.D. thesis, Dept. of Computer Science, University of Rochester, 1995.

Huang, X., Alleva, F., Hwang, M.Y. and Rosenfeld, R. An Overview of the SPHINX-II Speech Recognition System. In Proc. ARPA HLT Workshop, 1993.

Acknowledgments

This work was supported by ONR/ARPA research grant no. N00014-92-J-1512 and ONR research grant N00014-95-1-1088. Thanks for Alex Rudnicky and CMU for providing us with the Sphinx II system.