The TRAINS Project: A case study in building a conversational planning agent

James F. Allen, Lenhart K. Schubert, George Ferguson, Peter Heeman, Chung Hee Hwang, Tsuneaki Kato,[1] Marc Light, Nathaniel G. Martin, Bradford W. Miller, Massimo Poesio,[2] David R. Traum

The University of Rochester
Computer Science Department
Rochester, New York 14627

TRAINS Technical Note 94-3

September 1994

Abstract

The TRAINS project is an effort to build a conversationally proficient planning assistant. A key part of the project is the construction of the TRAINS system, which provides the research platform for a wide range of issues in natural language understanding, mixed-initiative planning systems, and representing and reasoning about time, actions and events. Four years have now passed since the beginning of the project. Each year we have produced a demonstration system that focused on a dialog that illustrates particular aspects of our research. The commitment to building complete integrated systems is a significant overhead on the research, but we feel it is essential to guarantee that the results constitute real progress in the field. This paper describes the goals of the project, and our experience with the effort so far.

This paper is to appear in the *Journal of Experimental and Theoretical AI*, 1995.

The TRAINS project has been funded in part by ONR/ARPA grant N00014-92-J-1512, U.S. Air Force–Rome Laboratory research contract F30602-91-C-0010, and NSF grant IRI-90-13160.

[1]Tsuneaki Kato's address is NTT Information and Communication Systems Lab, Kanagawa, JAPAN.

[2]Massimo Poesio's address is Centre for Cognitive Science, University of Edinburgh, 2 Buccleuch Place, Edinburgh, EH8 9LW Scotland, UK.

1 Introduction

The TRAINS project is a long-term effort to develop a system that can use English conversation to interact with and assist humans in problem solving tasks. Viewed in one direction, we are developing a natural language system that uses extensive common-sense knowledge and reasoning. Viewed in the other direction we are developing a plan reasoning system that can collaborate in a natural way with humans to solve problems. The particular application involves an interactive planning assistant that helps a user construct and monitor plans about a railroad freight system. We have completed the fourth year of the project, building a prototype system at the end of each year. The development of each system was driven by particular dialogues that were constructed by selecting and cleaning up excerpts of actual spoken dialogues between two people solving problems in the TRAINS domain. This paper describes our experience in the project so far, what problems we have faced and how the system has evolved. It also gives an overview of the most recent version, the TRAINS-93 system.

If the only goal were to construct a system that scheduled trains, then the work should be evaluated by how well the system actually performs the task. But absolute performance in a single domain is not our concern. Rather, the TRAINS system is a platform for supporting research into general language processing and plan reasoning capabilities, and we have not developed modules that use domain-specific techniques. Each component is based on general principles and is specialized to TRAINS only by adding the appropriate domain and task knowledge. As a result, most modules in the system handle a wider range of behavior than evidenced by examples used in the demonstrations.

From the natural language viewpoint, our goal is to develop the technology for embedded dialogue systems that perform a non-linguistic task involving significant reasoning. By studying dialogue as part of a collaborative problem solving task, we can study the effects of knowledge and reasoning on different dialogue phenomena. From the problem solving viewpoint, the goal is to develop technologies for collaborative plan reasoning system in which goals are discussed and plans are formed incrementally as the two agents interact. This mode of reasoning has attracted little attention so far in the problem solving literature, but is essential for effective human-computer collaboration. In fact, these two viewpoints cannot be separated. In a dialogue, the participants need to plan to make sure that information is communicated effectively. And in mixed-initiative planning, the agents require a communication medium with the power and flexibility of natural language.

Our research has been strongly motivated by studying actual human performance. We have collected over eight hours of human-human dialogue in the TRAINS domain, and have used this data to gain insight into both how language is actually used, and how humans collaborate to form plans. This type of analysis is crucial for identifying what problems actually arise, and helps us avoid working on problems that might seem interesting in the abstract, but do not occur in practice.

The paper describes the project in a top-down manner. Section 2 considers the issue of choosing a domain, the crucial first step of the work. Section 3 then provides an abstract characterization of the TRAINS system as a conversational agent. With this description in hand, we then present two examples in Section 4, which provide an overview of the system's activity as it collaboratively defines and executes plans. We then proceed to describe the TRAINS-93 system in more detail. Section 5 addresses global issues, namely the overall system architecture and the knowledge representations used to communicate between modules, and Section 6 gives a short description of each of the major components. Section 7 addresses some related issues, including the TRAINS world simulator and two projects that will contribute to future systems but do not presently play a role in the TRAINS-93 system. Finally, Section 8 discusses the accomplishments of the project so far, and describes some issues that are motivating our current work on the next TRAINS system.

2 Choosing the Task

Choosing an appropriate task and domain was one of the most crucial decisions early in the project. It had to balance several factors that appear contradictory at first glance. On one hand, the task had to be rich enough so that it would naturally lead to a wide range of dialogue behavior. In order to study discourse structure and dialogue phenomena, it was crucial that the task didn't limit the forms of interaction. On the other hand, the task had to be simple enough to allow the development of an effective reasoning system during the life of the project. If we cannot hope to perform the task, then the system can never be demonstrated, and the link between reasoning and dialogue cannot be systematically studied. In addition, the task must be a natural one for a person that does not require specialized training yet is not too simple. If the task requires significant training, then the person may prefer to use a specialized sublanguage to communicate with the system rather than natural language. If the task is too easy, then people will prefer to solve the problem themselves rather than collaborating with the system.

We explored two other tasks before choosing the final one. The first was an automated reference librarian [Allen *et al.*, 1989]. In this scenario, a person was given an essay topic and then was asked to interact with the "system" in order to find relevant books from the library catalog. We performed experiments and found that this task produced a rich range of dialogue and collaborative problem-solving behavior. The problem was that it was not feasible to construct an reasoning system for this task. If it had been restricted to simple information retrieval queries, then it might have been possible. But few of the dialogues were of this sort. Rather, the conversants discussed general characteristics of the essay content and developed sets of keywords for querying the catalogue based on their common-sense knowledge about the world coupled with heuristics based on what makes an effective keyword. The second task we explored was a robotics-control application, where a person communicated with the system to monitor and control a model train layout [Martin *et al.*, 1990]. This task supported interesting problem solving behaviors, but did not support a wide range of dialogue behavior. In particular, the tendency was for the person to issue a series of commands to the system. As such, it did not provide adequate support for the dialogue aspects of the project.

This led to the third task—the person and system interact to manage a railway transportation system. Neither directly executes the actions in this domain. Rather, the actions are performed by railroad engineers and factory and station supervisors. Dialogue is encouraged by giving the person and the system different responsibilities and knowledge. The

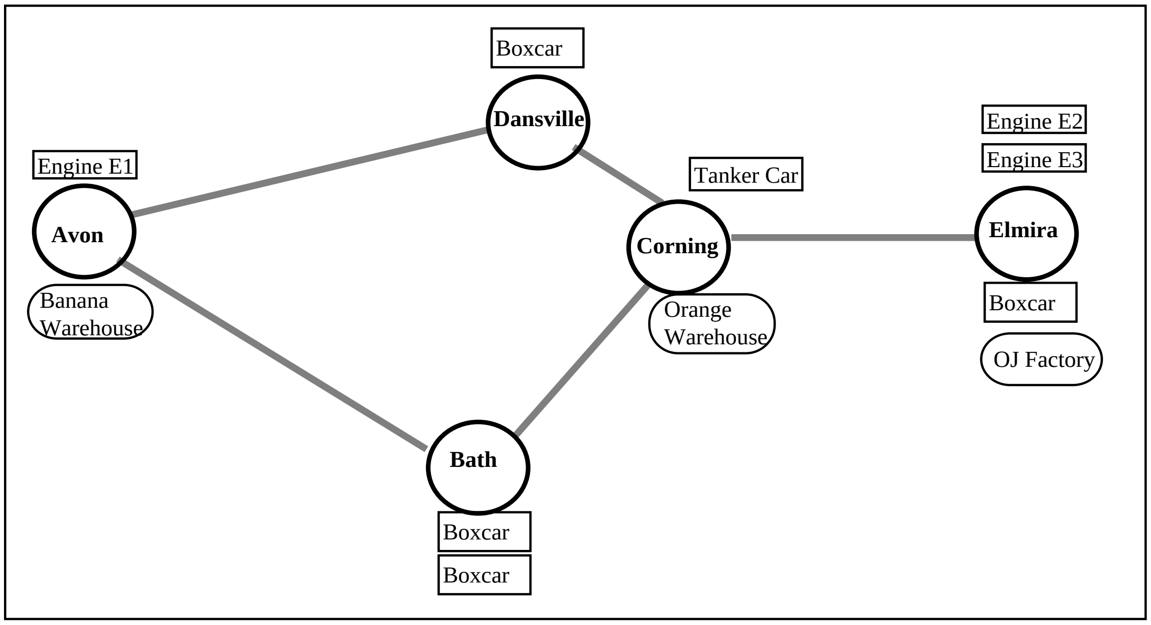

Figure 1: A simple TRAINS-world Map

person (henceforth called the manager) is given the goals that need to be achieved, but does not have direct access to the world or complete knowledge about the actions that can be performed. The system acts as the manager's assistant. It knows what actions are possible, and provides the communication link to the world. It interacts with the manager about all aspects of this task using the natural language interface. The only information that they initially share is a map of the current TRAINS scenario. Figure 1 shows a map of a small scenario that has been used for much of the dialogue collection and as the scenario for the TRAINS-93 system. The world consists of five cities, each of which contains a rail station that contains engines and rail cars as indicated. Some cities also contain warehouses containing goods (*e.g.*, there are bananas at Avon and oranges at Corning), and factories (*e.g.*, Elmira has an orange juice factory that converts oranges to orange juice). A very simple goal in this scenario might be to get one boxcar of oranges to Bath by 8 AM, starting at midnight. The plan might involve sending an engine from Elmira with a boxcar to Corning to get some oranges, and then continuing on to Bath.

The TRAINS task supports research on a wide range of issues of crucial importance for automated problem solving systems. Specifically, it involves extensive collaborative planning, requires a rich temporal model of action for planning and scheduling, and requires a sophisticated model of plan execution and monitoring using incomplete information. Specifically, the complete system will have to perform the following tasks:

- evaluating courses of action, calculating expected completion times, and detecting scheduling conflicts

- filling in details of the proposed plan that do not require the manager's attention
- suggesting ways to solve particular subproblems that arise
- presenting and describing the current plan and the state of the world
- dispatching the plan to the (simulated) agents in the world
- interpreting reports back from the agents in the world in order to monitor the progress of the plan and anticipate problems before they arise
- coordinating the correction and modification of plans with the manager, even as the plans are being executed.

On the face of it, it is not obvious that this task supports rich dialogue interactions. Our data collection, however, revealed rich dialogues even for simple goals. This is because much of the dialogue involves collaborative problem solving in general, such as specifying goals, discussing problem solving strategies, and confirming or clarifying suggestions. It turns out that the complexity of the underlying scenarios and the goals has little effect on the range of dialogue phenomena observed, except for the obvious characteristic of length. Clearly, the more complex the problem, the longer the dialogue required to solve it. Figure 2 contains an excerpt from one of the dialogues in the TRAINS-91 corpus [Gross *et al.*, 1993]. The utterance number indicates the turn and the utterance in the turn. Thus 1.1 is the first utterance of the first turn, 1.2 is the second utterance of the first turn, and so on. The two people spoke to each other over headsets, and could not see each other. The only information they shared was the map shown in Figure 1.

Consider a brief analysis of what happened in this dialogue. Utterance (1.1) and (1.2) set up the goal and describe the initial situation. This goal is accepted by the assistant in utterance (2.1). The manager then signals that she will start the planning process, by taking the turn with the utterance okay in (3.1). Utterance (3.2) is used to maintain the turn while the manager is thinking. Utterance (3.3) starts to describe relevant aspects of the current state of the world, but after a 4 second wait, the assistant decides to speak (utterance 4.1). But this is interrupted by the manager with (5.1), and a clarification subdialogue is initiated in (5.2) and (5.3). The assistant answers the question in (6.1), and the manager accepts the answer and resumes discussing the plan with utterance (7.1). Utterance (7.2) focuses attention on the subgoal of getting a boxcar to Corning. This sets the context for interpreting the next few utterances. Utterance (9.2) suggests a particular method of getting a boxcar to Corning, namely taking a boxcar from Bath. The assistant accepts this suggestion in (10.1), and then takes the initiative for the development of the plan in (12.3) by asking which engine should be used. Instead of answering directly, the manager starts a complicated clarification subdialogue about the distances of various routes. These utterances demonstrate the incremental nature typical of spoken dialogue, and contain a speech repair (Avon was mispronounced in (13.4) and corrected in (13.5)), and a restart (the utterance is abandoned at the end of (13.6) and utterance (13.7) restarts with a suggestion to send engine E2 from Elmira to Corning). The complexity of this dialogue for such a simple goal demonstrates that the domain handily meets all of our requirements.

```
Manager: We have to ship a boxcar of oranges to Bath by 8 AM          (1.1)
         and it is now midnight                                       (1.2)
Assistant: Okay                                                       (2.1)
Manager: Okay                                                         (3.1)
         Um ... all right                                             (3.2)
         So, there are two boxcars at Bath and one at Dansville and ... (3.3)
Assistant: and there's                                                (4.1)
Manager: Wait                                                         (5.1)
         I've forgotten where the oranges are                         (5.2)
         Where are the oranges                                        (5.3)
Assistant: The oranges are in the warehouse at Corning                (6.1)
Manager: Okay                                                         (7.1)
         So we need to get a boxcar to Corning                        (7.2)
Assistant: Right                                                      (8.1)
Manager: Alright                                                      (9.1)
         So, why don't we take one of the ones from Bath              (9.2)
Assistant: Okay                                                       (10.1)
Manager: So                                                           (11.1)
Assistant: We need                                                    (12.1)
           Okay                                                       (12.2)
           Which engine do you want to bring it                       (12.3)
Manager: Oh. Um. How about                                            (13.1)
         Well let's see                                               (13.2)
         what's shorter                                               (13.3)
         the distance between Avon and                                (13.4)
         Avon and um Bath or                                          (13.5)
         Elmira                                                       (13.6)
         It looks like it's shorter from Elmira to Corning            (13.7)
         so why don't you send E2                                     (13.8)
         to Corning                                                   (13.9)
Assistant: Okay
```

Figure 2: An excerpt from a dialogue (d91-7.1 in the TRAINS-91 corpus)

In fact, handling dialogue like this is well beyond the current state o f the art of discourse processing systems. Even with perfect speech recognition, no system yet can handle such incremental development, mixed-initiative interactions, speech repairs and corrections. And certainly, it is not yet possible to attain near-perfect speech recognition. So we had to introduce some simplifying assumptions in the project from the start. The principal decision was initially use keyboard input, thereby avoiding the speech recognition problem. We also decided use constructed dialogues that are motivated by the corpus but have some complexities, such as speech repairs, removed. We started some subprojects in parallel for dealing with problems such as speech repairs, but this work has not yet been integrated into the system. Some of this work will be described briefly in Section 7.

3 The TRAINS System as a Conversational Agent

An agent is an entity that acts in the world. The TRAINS system is an example of a conversational agent, all of whose actions involve communication. The system performs communicative acts when it generates sentences and sends messages to the agents in the TRAINS world. It also observes communicative acts by other agents when utterances are performed by the manager and reports are produced by the TRAINS world agents.

One of the insights of the early computational work on speech acts (*e.g.*, [Cohen and Perrault, 1979; Allen and Perrault, 1980]) was that traditional planning mechanisms could be used for communicative acts. TRAINS can be viewed as continuing in this tradition, although it must deal with many additional complications that arise because of the highly interactive nature of our dialogues. But its conversational behavior still results primarily from planning and plan execution. To see this, first consider a model of a non- communicating, deliberative agent that plans and executes physical actions shown in Figure 3. This has many similarities to the BDI (belief, desire and intention) architecture described by Bratman *et al.* [1988]. The agent has a set of beliefs about the state of the world. Using these beliefs and some predetermined set of desires, the agent selects goals to achieve. Once the agent selects a goal, it constructs an abstract plan that it believes will lead to the achievement of the goal. When this plan is sufficiently developed, the initial actions in the plan can be executed. We assume a division between actions used for reasoning about plans and a more detailed level of action that directs physical behavior. The agent takes an abstract plan and selects and executes the physical actions. It also generates expectations about what observations it might receive based on the action performed. As observations are made of the world, the agent uses its expectations and other general knowledge to draw conclusions about the current state of the world. This is typically called plan monitoring in the planning literature. Of course, when the agent's beliefs change, it may cease to believe that the current plan will work, and further deliberation (replanning or plan repair) will occur, revising the agent's plan.

A conversational agent operates in a more complex environment that involves other agents. The TRAINS system must both plan and execute communicative actions and understand the communicative actions performed by other agents. We use essentially the same model as shown in Figure 3 but specialized to discourse actions. Figure 4 shows the model. Using its beliefs about the domain, including beliefs about what is shared knowledge and

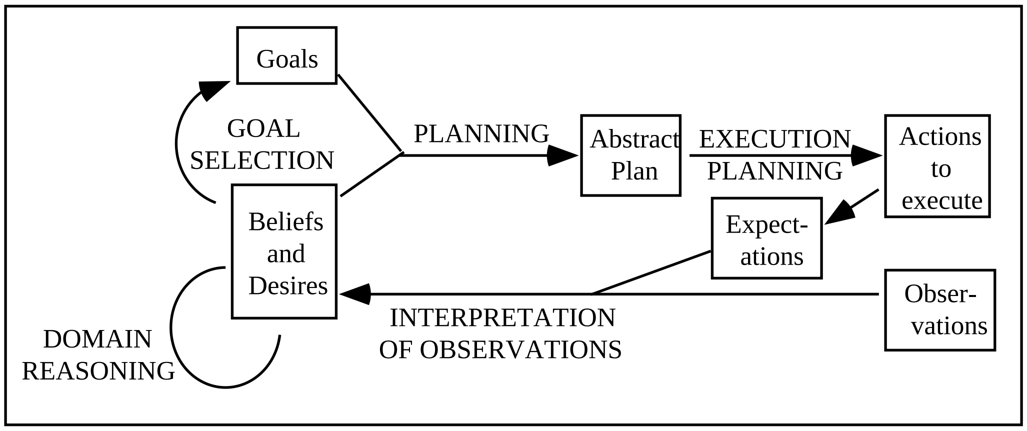

Figure 3: A model of a simple deliberative agent

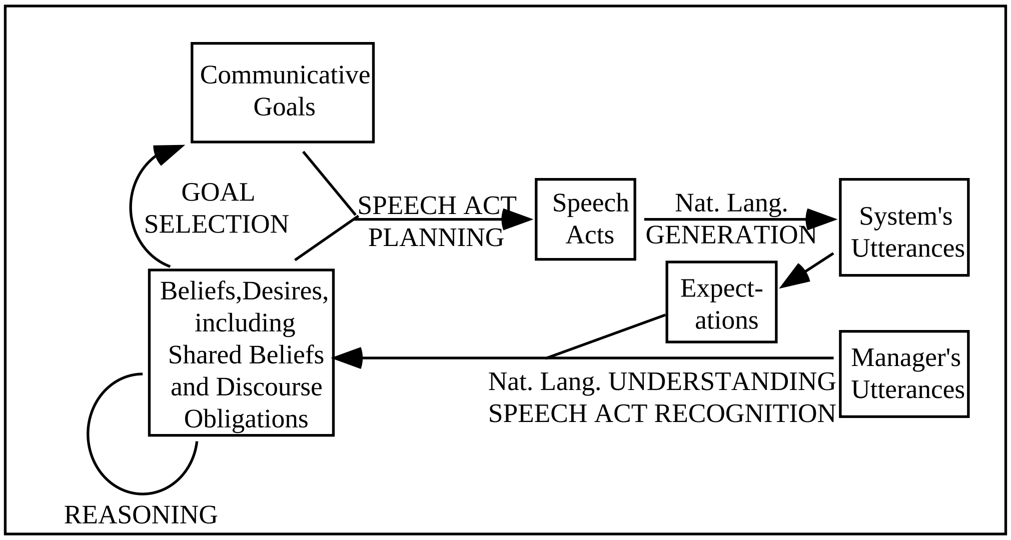

Figure 4: A model of TRAINS as a conversational agent

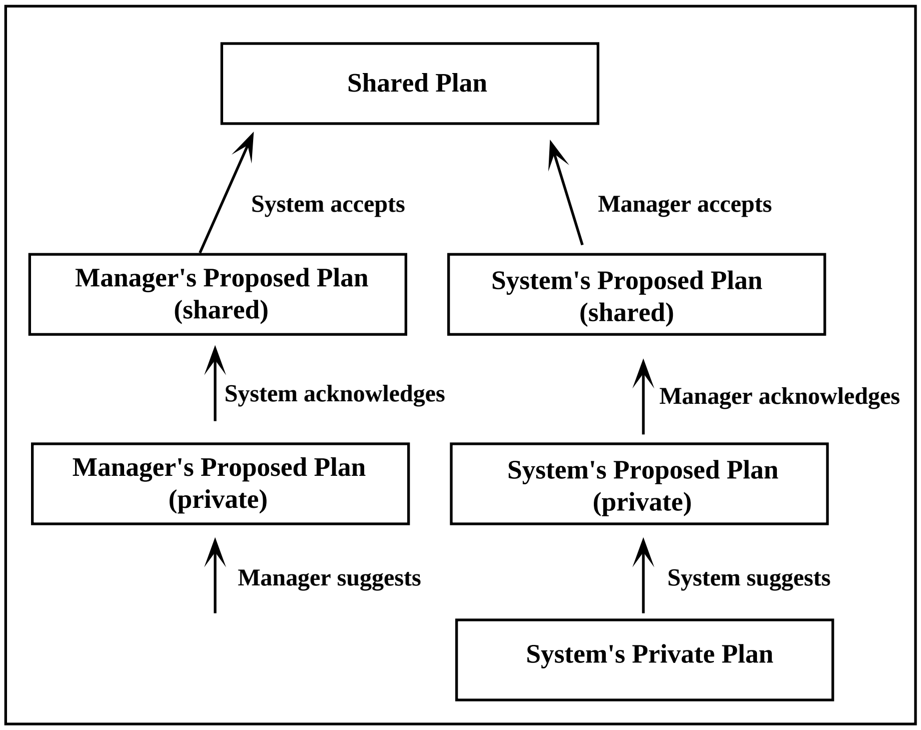

Figure 5: The different plan modalities in handling suggestions

what discourse obligations each agent has, the system selects communicative goals. By considering these goals, together with beliefs about the current state of the dialogue, the system decides what speech acts to perform next. The speech acts are executed by generating natural language utterances, thereby creating the system output. When the manager speaks, the utterance is analyzed and the intended speech acts identified, then the system's beliefs about the discourse state and the system's discourse obligations are updated.

Another complication arises because the topic being discussed is itself a plan. This is not apparent from Figure 4, but reasoning about domain plans (*i.e.*, about action in the Trains world) is a major component of the system's reasoning as it plans its own communicative acts and interprets the manager's communicative acts. Specifically, the dialogue is driven by the task to find a mutually agreed-upon (or shared) domain plan. Each agent plans additions to the shared plan privately, and then tries to get the other agent to agree to the extensions by communicating. Thus, the speech acts are motivated by reasoning about the domain. Likewise, interpreting speech acts depends on identifying how suggestions from the other agent relate to the shared plan. To encode this, speech act definitions may include constraints that relate their content to the current shared domain plan, as first proposed by Litman and Allen [1987].

The flow of information regarding suggestions is shown in Figure 5. The Trains system supports a set of different plan modalities that capture the status of plan fragments, indi-

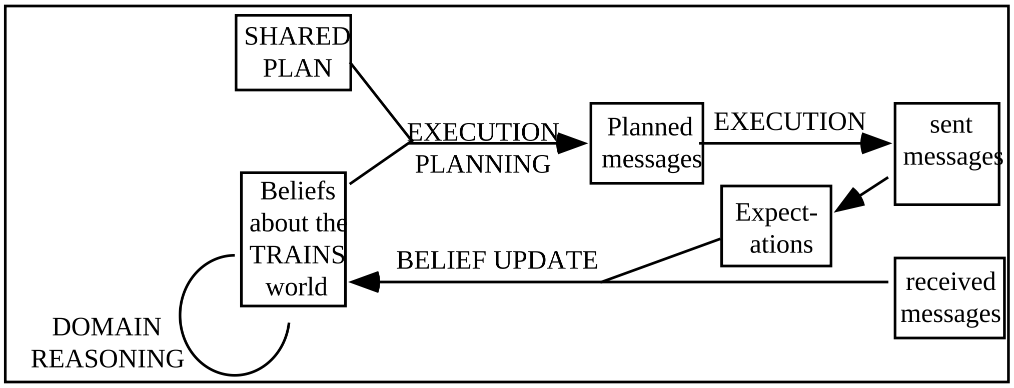

Figure 6: The abstract model of domain plan execution

cating whether they have been acknowledged, accepted or simply just heard. By generating appropriate speech acts such as suggestions, acknowledgments and acceptances, the two agents move the plan fragments through the different modalities until they have a shared plan.

The Trains system also executes and monitors the domain plan. Remember that the plan that was constructed by the dialogue involves actions by agents in the Trains world—the train engineers and factory supervisors. To get this plan executed, the Trains system must perform another task, planning messages to send to the agents so that they execute the desired actions. This could be handled just like the speech act planning for the dialogue, but instead we pursue a simpler route that assumes there is no ambiguity in the messages generated and that all messages sent are received and understood. Thus, each message uniquely identifies its content in a formal communication language. In addition, since the Trains system cannot directly perceive the world, it can only monitor the progress of the plan from the messages it receives back from the engineers and factory managers. As a result, the Trains agent may have to plan to communicate with the other agents to get them to produce the desired reports.

Figure 6 shows the model of the system's activities while controlling the execution of the domain plan produced from the dialogue. The system uses knowledge of the shared plan and its beliefs about the world to plan communication messages to the agents in the Trains world. Such messages may involve requests to perform certain actions, including actions of producing reports about the current state of the world. Messages generated by the Trains world agents are observed and interpreted by the system in order to monitor the plan's execution. A problem arising during the execution causes the system to start a replanning task. If the modifications to the plan require confirmation by the manager, new discourse goals are created.

```
Manager: We have to make OJ.                                        (1.1)
         There are oranges at I                                     (1.2)
         and an OJ factory at B.                                    (1.3)
         Engine E3 is scheduled to arrive at I at 3PM.              (1.4)
         Shall we ship the oranges?                                 (1.5)
System: Yes,                                                        (2.1)
        shall I start loading the oranges in the empty car at I?    (2.2)
Manager: Yes,                                                       (3.1)
         and we'll have E3 pick it up.                              (3.2)
         OK?                                                        (3.3)
System: OK                                                          (4.1)
```

System then plans and dispatches a series of messages to the simulated TRAINS *agents, who successfully execute the plan.*

Figure 7: The TRAINS-91 Dialogue

4 The Demonstration Systems

While we developed a system every year for four years, two of the systems stand out as significant milestones. These were the TRAINS-91[3] system and the TRAINS-93 system. The principal goal of the TRAINS-91 system was to demonstrate the basic ideas of using dialogue for interactive plan construction and plan execution. The natural language processing was fairly limited, and focused exclusively on handling the demonstration dialogue (thereby avoiding most ambiguity resolution problems). The system illustrated the plausibility of our model of collaborative problem solving, and demonstrated the process all the way from interactive plan specification to plan execution. To see this, consider what the system did at each stage of the dialogue shown in Figure 7, which was constructed by cleaning up an excerpt from an actual dialogue in the corpus. The relevant part of the map that the manager and system shared is shown in Figure 8.

The TRAINS-91 system could determine plausible intentions underlying the manager's utterances and find the semantic connections that were necessary to construct the plan that was being suggested. The system also planned its own speech acts and used a simple template-driven sentence generation module.

Consider the different tasks that the system must perform in order to participate in this dialogue. First, the sentences must be parsed and an initial semantic interpretation computed. The sentence "We have to make OJ" is interpreted as the literal speech act:

The manager tells the system that there is an obligation to make OJ.

The system then interprets this action in the context of the TRAINS domain and determines that the intended speech act is a suggestion of a goal, namely:

[3]Much of the TRAINS-91 system is documented in the TRAINS technical notes on the TRAINS-90 system. The '91 system is described here as it was the first to include at least a real parser and simple language generator.

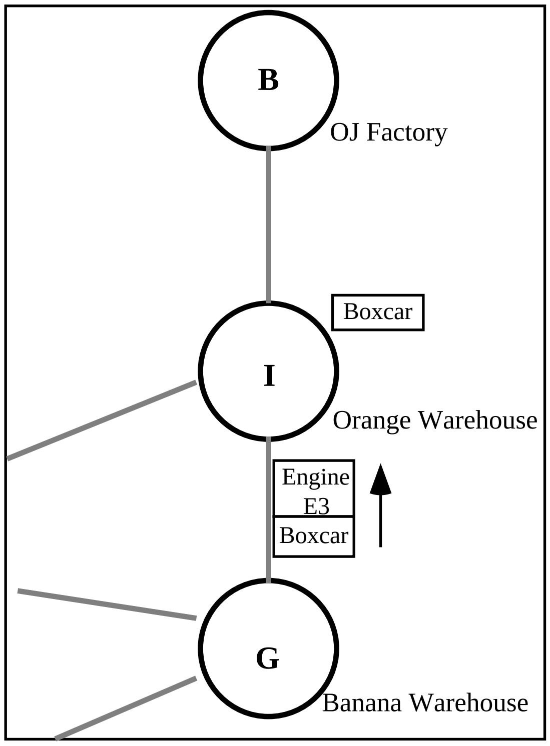

Figure 8: Part of the map for the TRAINS-91 Dialogue

The manager suggests a goal of making OJ.

In response to this, the system creates a new plan structure to represent the manager's proposed plan, with the goal of making OJ. Since the manager has not yet released the turn, the system does not attempt to respond at this stage. The next utterance, "There are oranges at I," is parsed and the literal interpretation is the action:

The manager tells the system that there are oranges at I.

The system interprets this act in the context of the plan suggested so far, and concludes that the manager is suggesting where to get the oranges to make OJ. To reach this conclusion, the system reasons about how the information that there are oranges at I is relevant to making OJ. It successfully finds the connection, and in doing so expands out the proposed plan. In particular, it infers that the manager is suggesting that they move the oranges at I to an OJ factory, even though this is never explicitly mentioned. The next utterance is interpreted similarly, namely as a suggestion about which OJ factory to use in the plan.

After processing utterance (1.3) the system now believes that the manager has suggested moving the oranges from I to city B, and then using the OJ factory at city B to make OJ, thereby satisfying the goal. The next utterance, "Engine E3 is scheduled to arrive at I at 3PM," is parsed literally as the action of telling the system when engine E3 is to arrive. But in the context of the currently developed plan, the system identifies the reason that the engine's arrival is relevant, namely that the engine could be used to move the oranges. Thus, this utterance is interpreted as a suggestion to use engine E3 to move the oranges. The final utterance in this turn, "Shall we ship the oranges," is interpreted in context as a more specific question asking whether the oranges should be shipped from I to B using engine E3. To see why this interpretation is necessary, note that the system could not answer yes to this question if it thought they should use another engine, say engine E1, to ship the oranges at I to city G. Thus, the question cannot be answered out of the context of the entire dialogue.

Because utterance (1.5) is a question, the system interprets it as releasing the turn and evaluates the proposed plan. The plan as suggested is acceptable, but there are two plausible ways to further specify it. One possibility is to use the boxcar currently at I to move the oranges. This way, the boxcar could be loaded by the time the engine arrives, thus saving some time. The other possibility is to wait and load the boxcar that arrives with engine E3. While this would take more time, it might be preferable as it eliminates the need to switch the boxcars. Faced with two plausible options, the system creates a discourse goal to query the manager about what to do. At this stage, the system's actions are affected by three different needs:

1. The manager asked a question that requires an answer.

2. The proposed plan seems acceptable as far as it goes, so the system should accept it.

3. A decision needs to be made as to which boxcar should be used in the plan.

These three factors affect the system as it plans its response. The first two needs can be satisfied by the single utterance (2.1), namely "yes." This both answers the question, and implicitly accepts the entire plan as suggested by the manager. So the proposed plan now has been agreed upon, and becomes part of the shared plan of the manager and system. To satisfy the third need, the system arbitrarily picks one of the options and plans a question about whether that option is desired. This results in a speech act realized by utterance (2.2), "Shall I start loading the oranges in the empty car at I?" In asking this question, the system releases the turn and awaits the manager's response.

Utterance (3.1), "yes," both answers the question and accepts the system's suggestion. So the shared plan now includes the steps required to load the boxcar at I, and then couple it to engine E3 when E3 arrives. If the manager had said "no," the suggestion would have been rejected and the other option could then be discussed. Utterance (3.2), "and we'll have E3 pick it up," simply serves to confirm what the system has already concluded. Utterance (3.3), "OK?," serves to request confirmation from the system. This was required with the TRAINS-91 system because it only took up the turn when faced with a question or request. The request to confirm the final suggestion is satisfied simply by utterance (4.1), "OK."

At this stage, the system believes that it has constructed a workable plan to achieve the goal of making OJ, shown in Figure 9. Events are shown in rounded rectangles, and facts are shown in regular rectangles. The arrows depict various forms of causal relations (*e.g.*, effect, enables, generates). Shaded events that correspond to planned actions. Unshaded events are either external events or events that are generated as a consequence of actions in the plan. Object O1 is the oranges, F1 the factory at city B, and C1 the car used to move the oranges. For the sake of clarity, the diagram generally ignores temporal details. The system believes that both it and the manager have accepted the plan, so the plan is shared knowledge.

The system then enters its next phase, which is the execution planning. It takes the abstract plan in Figure 9 and plans a series of messages to the TRAINS agents in order to get the plan executed. This involves reasoning about what actions the TRAINS agents can perform using knowledge about the current state of the world. For the current plan, the system generates the messages like the following to the agents:

To the warehouse supervisor at city I: load boxcar C1 with oranges

To the engineer of E3: When you arrive at city I, couple E3 to boxcar C1

To the engineer of E3: When boxcar C1 is coupled to E3, go to city B

To the factory supervisor at B: When boxcar C1 is at city B, unload it

To the factory supervisor at B: When the oranges are at the factory, make OJ

These messages are then sent to the appropriate agents in the simulated TRAINS world. For the TRAINS-91 demo, the simulator was set so that all actions always succeed, so this plan executed successfully and the orange juice was made.

The TRAINS-93 system performed the same type of task as the TRAINS-91 system, but involved a significant effort to improve the natural language capabilities of the system.

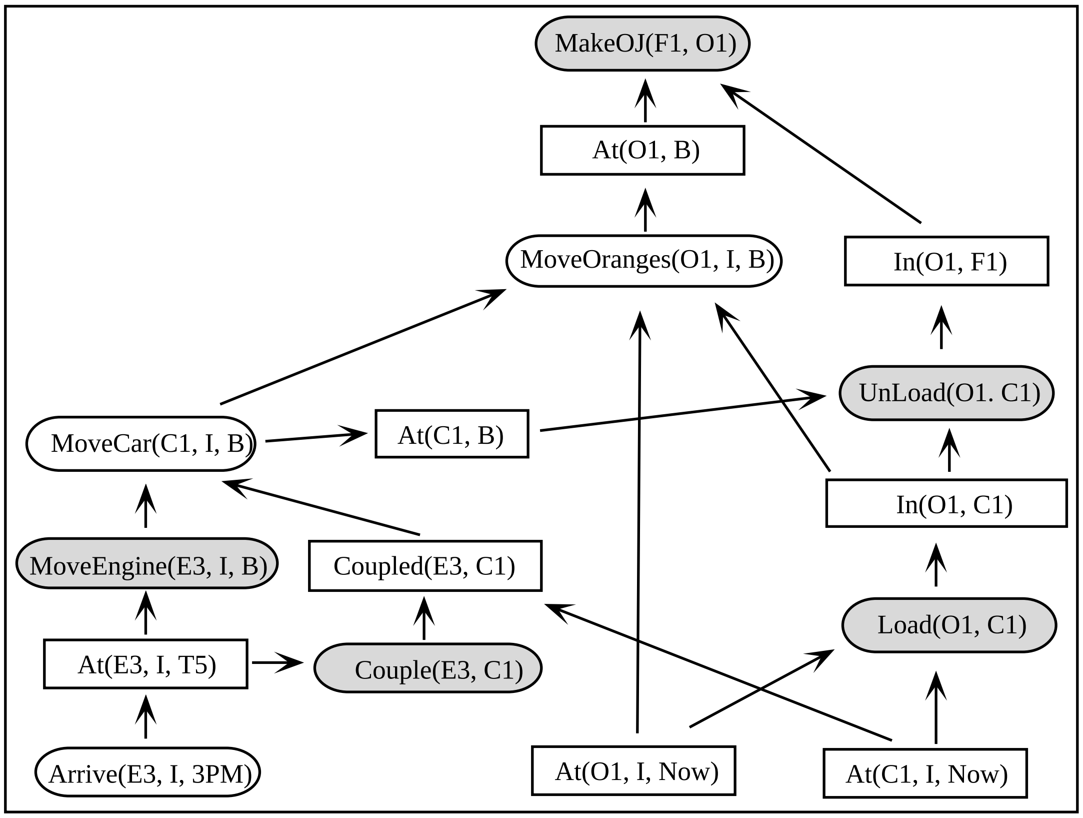

Figure 9: The plan constructed for the TRAINS-91 dialogue

```
Manager: We better ship a boxcar of oranges to Bath by eight a.m.       (1.1)
System: OK                                                              (2.1)
Manager: So we need to get a boxcar to Corning, where there are oranges. (3.1)
         There are oranges at Corning.                                  (3.2)
         Right?                                                         (3.3)
System: Right                                                           (4.1)
Manager: So we need an engine to move the boxcar.                       (5.1)
         Right?                                                         (5.2)
System: Right                                                           (6.1)
Manager: So there's an engine at Avon.                                  (7.1)
         Right?                                                         (7.2)
System: Right                                                           (8.1)
Manager: So we should move the engine at Avon, engine E1, to Dansville
         to pick up the boxcar there                                    (9.1)
System: Okay                                                            (10.1)
Manager: and move it from Dansville to Corning                          (11.1)
         load up some oranges in the boxcar                             (11.2)
         and then move it on to Bath                                    (11.3)
System: Okay                                                            (12.1)
Manager: How does that sound?                                           (13.1)
System: That's no problem                                               (14.1)
Manager: Good                                                           (15.1)
```

Figure 10: The TRAINS-93 demonstration dialogue

The issue of ambiguity, both during parsing and in the subsequent phases of discourse interpretation up to the generation of speech act hypotheses, was tackled in a systematic way. The focus dialogue for this system is shown in Figure 10, and the map was shown earlier as Figure 1. This dialogue involves sentences that are considerably more complex to understand. For instance, utterance 3.1 identifies a subgoal of getting a boxcar to Corning and adds additional information on why this is a relevant goal (there are oranges there). Another example is utterance (9.1) which both suggests an action (moving the engine at Avon to Dansville) and the reason for doing the action (to enable the action of picking up the boxcar there). It also includes some "run-on" utterances where the manager conveys a plan in a series of clauses. For example, utterances 9.1, 11.1, 11.2 and 11.3 together describe a plan in what appears to be a single sentence syntactically. There are also some complex noun phrase constructions including appositive constructs as in (9.1) "the engine at Avon, engine E1," and referring expressions such as "the boxcar there," which refers to the boxcar at Dansville, and pronouns such as "that" in utterance 13.1, which appears to refer to the plan constructed in the entire dialogue so far.

The natural language components were built to have significantly more coverage than demonstrated by this one dialogue, and the parser and lexicon developed actually cover much of our corpus of dialogues (after speech repairs are removed). Besides expanding the basic linguistic capabilities, the TRAINS-93 system achieved a closer integration of the techniques for reference resolution, scoping and speech act interpretation. In addition, it also contained an implementation of a rich set of conversation acts [Traum and Hinkelman, 1992] and the domain plan representation and reasoning component was significantly extended.

5 The TRAINS System Infrastructure

Before looking at each component of the TRAINS-93 system, there are several general issues that must be discussed. These concern the overall system architecture and the knowledge representations used.

The central question concerning the architecture was whether there would be system-wide uniform mechanism for handling alternative hypotheses for dealing with ambiguity. An example of such a formalism would be a blackboard-style control, where each module stored its hypotheses in a global data structure so that they could be accessed by other modules. The advantage of such an architecture is the flexibility it provides for adding new modules and for supporting unanticipated interactions. The disadvantage is that such architectures can be cumbersome and provide a significant overhead to the system operation if there are frequent "direct" calls required between modules. The opposite extreme is a fixed configuration of modules, where each interaction can be hand-tailored to the needs of the modules. The prime advantage of this scheme is its simplicity once the configuration of the system is defined. In the first year of the project, our desire to produce a short-term demonstration system dictated the latter approach. As the project continued into later years, the rigidity of the system architecture was never as pressing an issue as developing better component modules. As a result, the subsequent systems through TRAINS-93 have continued using a fixed configuration, although the details of the interaction between the modules participating in the speech act recognition phase have changed significantly. The fixed configuration has made some kinds of ambiguity more difficult to handle, but did not greatly impact the system development. The actual architecture for the TRAINS-93 system is described in Section 5.1.

A related issue was whether to use a common knowledge representation language throughout the system. A uniform representation language would have obvious advantages, but it was not clear that they would outweigh the inherent disadvantages. Different knowledge representation languages seem better suited for different purposes. In particular, to facilitate the natural language processing, the KR should be richly expressive and should provide for the efficient encoding of ambiguity. Thus, a language that mirrors the semantic structures revealed in English would be most convenient, as it could directly represent such constructs as predicate modifiers, modal operators, and generalized quantifiers. The representation used to support plan reasoning, on the other hand, requires efficient reasoning techniques specialized to reasoning about action, such as temporal reasoning, non-logical matching of hypothesized actions to expectations, explicit representation of action relations such as preconditions and effects, and a representation of plans that provides efficient search methods to support planning and plan recognition. The usual strategy in the planning literature is to restrict the knowledge representation language as far as possible while retaining the essential aspects for reasoning about the effect of actions and plans.

While it may be possible for a single representation to yield the advantages of both, we took the pragmatic route and used two different representations in the system: the representation for language processing was Episodic Logic (EL), while the representation for plan reasoning was Event-Based Temporal Logic (EBTL). These representations will be described in Sections 5.2 and refs:ebtl. To translate between the two, a conversion program was written. Since both languages are logic-based, and have many basic similarities, this was not an difficult task. The translation from EL to EBTL takes the form of an approximation, where the more complex constructs in EL are approximated in the weaker EBTL. Much of the time, no information is lost. For example, even though episodes in EL are significantly more powerful than events in EBTL, most episodes that arise in the dialogues correspond to simple event occurrences. Over the course of the project, the representation used become more similar in many respects. Thus, in the long term, a single representation language with separate specialized reasoning capabilities may be the best approach.

5.1 The TRAINS-93 Architecture

The architecture of the TRAINS-93 system is shown in Figure 11. The modules can be partitioned into two main groups: the linguistic reasoning modules, using EL as a basis for processing and communication, and the plan reasoning modules, using EBTL as a basis for processing and communication. External to the system but a crucial part of the project, the TRAINS simulator provides an "external world" to support plan execution and monitoring.

To see how information flows through the system, consider the processing of the manager's utterance "So we need an engine to move the boxcar" (Utterance 5.1 in Figure 10). The parser produces the following logical form in EL (with some details suppressed):

```
(DECL
  (UTT-IMP SO-COORD
    (WE1
      ((ADV-A (IN-DISCOURSE-RELATION
                (TO1 (MOVE <THE BOXCAR>))))
       (NEED-REQUIRE <A ENGINE>))))))
```

The `DECL` operator indicates that the utterance is a `TELL` surface speech act. The discourse coordinator `SO-COORD` connects the content of the speech act to an implicit utterance (`UTT-IMP`) that must be identified from the discourse context. The speech act describes a need for an engine that is related in some way (signaled by the discourse relation `TO1`) to the action of moving an object described as a boxcar. The angle brackets indicate unscoped operators, which later may be scoped by the deindexing module. For instance, the construction `<A ENGINE>` indicates an indefinite object of type `ENGINE`, but does not indicate the scoping of the operator with respect to other constructs such as the `NEED-REQUIRE` operator. In other words, it retains the ambiguity as to whether there is a specific engine that is needed (the wide scope interpretation), or that any engine would do (the narrow scope interpretation).

This logical form is passed to the deindexing module, which generates a set of hypotheses about scoping and the interpretation of the referential expressions contained in the sentence. For example, it hypothesizes that the pronoun "we" refers to the set consisting of the manager and the system, that the definite description "the boxcar" refers to the boxcar

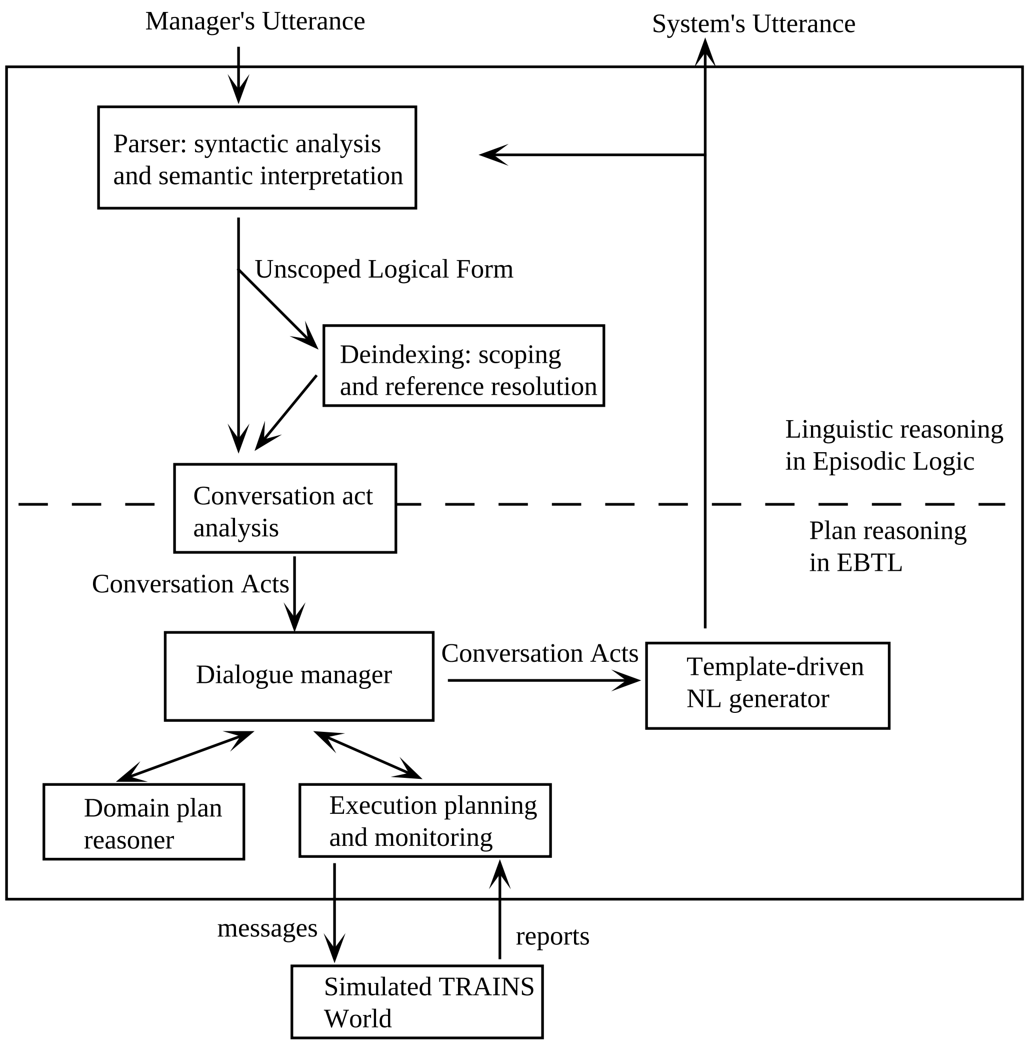
Manager's Utterance
System's Utterance
Parser: syntactic analysis and semantic interpretation
Unscoped Logical Form
Deindexing: scoping and reference resolution
Linguistic reasoning in Episodic Logic
Conversation act analysis
Plan reasoning in EBTL
Conversation Acts
Dialogue manager
Conversation Acts
Template-driven NL generator
Domain plan reasoner
Execution planning and monitoring
messages
reports
Simulated TRAINS World

Figure 11: The TRAINS-93 System Architecture

introduced in utterance (3.1). It also converts the tense operators into explicit temporal relations among episodes, and scopes the unscoped operators as far as possible. The deindexing module arrives at these hypotheses on the basis of information about the structure of the conversation up to that point, together with information about the syntactic structure of the utterance and the interpretation of its lexical items.

The logical form and the results of deindexing are then passed to the conversation act analyzer, which first identifies several possible interpretations of the utterance. At the core speech act level: there are several possible interpretations:

1. An inform of a need for an engine to move the boxcar
2. A check whether there is a need for an engine to move the boxcar
3. A question whether there is a need for an engine to move the boxcar
4. A suggestion that an engine be used in the plan, with a supplementary suggestion of moving the boxcar.

The interpretation of the word "so" produces a hypothesis that this utterance is a followup, subordinate to a previous utterance. In addition, discourse relations `T01` is hypothesized to be a purpose relation between using the engine and moving the boxcar. Note that the conversational act hypotheses are represented using the EBTL language and provide the conversion between the two knowledge representations.

These interpretation hypotheses are passed to the dialogue manager (DM), where they are checked to see which are reasonable given the discourse context. Specifically, the preconditions on each conversational act are checked. For example, to evaluate the plausibility of an inform speech act, the domain plan reasoner is queried to see if it is already shared knowledge that an engine is needed in the plan. The domain plan reasoner returns that this is the case, and hence the inform act is deemed implausible. For the same reason, it is unlikely that the utterance is asking whether an engine is needed. The check interpretation remains plausible because there are few constraints on its use as it often serves to focus or co-ordinate the activity of the agents. To evaluate the suggest interpretation, the domain plan reasoner is called to try to see how an engine and a move event might fit into the plan. It finds that the engine could be used in the move-car action that was introduced when processing utterance (3.1). Thus the interpretation of the utterance as a suggestion also seems plausible.

The current DM accepts all of the plausible act interpretations. Thus the sentence is treated both as a check whether an engine is needed and as a suggestion to use an engine in order to move the boxcar. In addition, it is interpreted as initiating new material, rather than acknowledging, repairing, or continuing previous material.

The verified set of conversation acts are then used to update the discourse context. The suggestion is incorporated into the manager's proposed plan and, based on knowledge of discourse conventions, the dialogue manager also recognizes an obligation to respond to the check and suggest acts. If the manager had not continued speaking, the system would have satisfied these obligations by planning to perform a speech act to accept the suggestions. But in this dialogue, the manager continues with the query "right?" This is easily parsed and is eventually interpreted as a dialogue repair/elaboration that requests that the previous action be confirmed. This creates an obligation to answer the check act about the need for an engine, and the system plans an inform act confirming that an engine is needed. This is passed to the generator module, which associates templates with simple speech acts. The inform act is realized by the utterance "right." This serves both to answer the question and to implicitly accept the suggestion of using an engine to move the boxcar. The system's utterance is fed back through the system so that the system can update the discourse state. It also allows the system to double-check its own utterances to make sure that the intention it would recognize from the utterance is compatible with its original intention that motivated the utterance.

In the dialogue as presented, the manager speaks next and continues the dialogue. If the manager had not spoken by the time the system had completed processing its own utterance, the system would have invoked the domain plan reasoner to try to expand the plan further to find other issues that need to be discussed. This process continues until both agents believe that there are no remaining problems to deal with in the plan.

With the architecture defined, the next two sections consider the major knowledge representation formalisms used in the system.

5.2 Episodic Logic

Episodic Logic was developed for use as a semantic theory for natural language understanding [Hwang and Schubert, 1993a; Hwang and Schubert, 1993b]. It was designed to meet the following requirements:

- expressive adequacy: the language should be powerful enough to allow us to represent various kinds of constructs found in English, as well as the nuances in naturally occurring sentences. Among other things, this includes tense operators (past and present), aspect (perfect and progressive), various kinds of adverbials, including manner adverbials (*e.g.*, "using engine E3"), purpose clauses (*e.g.*, "to pick up the boxcar"), kinds of actions and events (*e.g.*, "we need to get a boxcar"), kinds of natural objects (*e.g.*, "we need oranges"), modals (*e.g.*, "need to," "should," "had better"), and other intensional verbs including creation verbs (*e.g.*, "we have to make OJ").

- derivational adequacy: the language should support a simple, systematic derivation of meaning from English surface structures.

- semantic adequacy: the meaning of the language itself should be precisely defined, *i.e.*, it should have a denotational semantics.

The most distinctive aspect of EL is its use of *episodes*, which are similar to situations in situation semantics [Barwise and Perry, 1983]. Like a situation, an episode characterizes a partial state of affairs over some period of time at some location. It subsumes the notion of events that is used in many representations based on Davidson [1967], because an event is a particular kind of episode. The content of episodes can be described using two special operators. The $*$ operator indicates that an arbitrary formula is true of an episode. A simple example would be the formula

$$[TR1\, Move\, Avon\, Dansville] * E1$$

which is true only if episode $E1$ involves engine $TR1$ moving from Avon to Dansville. Square brackets in EL indicate an argument order where the subject occurs first and the predicate second, the same order as in English. In a Davidson-based event logic, this might be written as $(Move\, E1\, TR1\, Avon\, Dansville)$, where the event is an argument to the predicate. But EL can represent much more complex events than is possible using event arguments to predicates. For example, the episode $E2$ in which every train at Avon went to some city could be described as follows:

$$(\forall t\, .\, [t\, AtLoc\, Avon] \supset \exists c\, .\, [t\, Move\, Avon\, c]) * E2.$$

This could not be encoded as a single event in an event-based logic. Episodic logic also supports another operator, $**$, which asserts that some proposition completely characterizes an episode (*i.e.*, there is no additional information that is part of the episode).

EL makes extensive use of predicate operators: operators that map predicates into other predicates. For example, the predicate OJ is true of quantities of orange juice. The predicate operator *make* takes this predicate and produces another predicate $(make\ OJ)$, which is true of any agent that makes orange juice. Many different syntactic constructs have simple interpretations when viewed as predicate operators. Special forms are used to capture adverbial expressions. For instance, the sentence The train left at 8 PM would be represented by the form $((ADV\text{-}E\ (At\text{-}time\ 8PM))[\langle The\ Train\rangle\ leave])$, where the $ADV\text{-}E$ operator associates an adverbial modifier (in this case, occurring at 8 PM) with an event (in this case, event of the train leaving).

EL also has an operator, K, which constructs a kind or type of object from a predicate. The noun phrase orange juice for example, often refers to orange juice in general rather than to some specific quantity of orange juice, as in the sentence I like orange juice. If OJ is a predicate true of quantities of orange juice, then $(K\ OJ)$ is the kind of thing, or substance, orange juice. Kind terms are used extensively with events as well, as in the sentence I want to load the oranges, where what is desired is the performance of some event of the kind "load the oranges," rather than a specific act of loading oranges.

Episodic Logic readily lends itself to inference; contrary to a widespread myth a rich syntax is no impediment to effective inference. Though only a very limited set of EL inference capabilities has been used in TRAINS-93, EL has been separately implemented in the EPILOG system [Schaeffer *et al.*, 1993]. EPILOG is a powerful knowledge management and inference system allowing data-driven inference, goal-driven inference, and featuring integration with about a dozen specialist modules for accelerating temporal, taxonomic, partonomic, set-theoretic, numeric, and other special types of inference.

5.3 The EBTL Representation

EBTL (Event Based Temporal Logic) is a typed logic based on interval temporal logic [Allen, 1984; Allen and Ferguson, 1994], which was designed to facilitate reasoning about actions and events in temporally complex worlds. It is built on top of the RHET knowledge representation system [Allen and Miller, 1991], which provides the system with several key reasoning capabilities, including type hierarchies and typed unification, equality reasoning, temporal reasoning, frame-like event representations, and an explicit context mechanism for representing belief contexts, hypothetical reasoning and other modalities. EBTL provides additional constructs that proved essential for representing the meanings of natural language input, including explicit quantifiers, discourse markers, lambda expressions and quoted propositions.

The language uses a reified event representation, where event instances are denoted by explicit terms in the language. Figure 12 shows part of the event hierarchy for the TRAINS domain. This hierarchy predominantly deals with events caused by actions, which capture different actions available to the TRAINS domain planner. Events that cannot be directly executed are classified as non-action events. These events are necessary to provide a clean interface with the natural language system, because people often talk in terms of the events caused rather than the actions executed. For example, the system never plans to execute an arrive event. Rather, it plans a move action, one aspect of which is sometimes described as an arrive event. Another important class of events in this domain is the class of speech acts. Figure 13 summarizes the speech acts that are currently used in the TRAINS-93 system. The arguments to events are identified by role functions, which are defined over event classes.

The terms in EBTL are the ones allowed in RHET. Constants are written as atoms within square brackets. Thus `[ENG3]` is a term. Distinct names do not necessarily identify distinct objects, since the language supports equality reasoning. When terms are defined, their type is usually defined as specifically as possible. `[ENG3]`, for instance, is declared to be of type `T-ENGINE`. Variables are indicated using the form ?*varname***typename*. For instance, `?x*T-ACTION` is a variable that ranges over all objects of type `T-ACTION`.

Propositions in EBTL are represented by lists. For instance, since `EQ?` is the equality predicate, the proposition

```
(EQ? [F-ENGINE [M2]] [ENG3])
```

would assert that `ENG3` is the engine causing the move event `M2`. EBTL includes some special quantification forms that provide a convenient interface to the language modules. In particular, the following supports definite descriptions. The formula

(`THE` *var dm restriction form*)

is true only if there is a unique contextually-salient object which satisfies the restriction (including the type restriction on the variable), and which satisfies the form. This is a fairly standard treatment of "the" as a generalized quantifier except for the discourse marker *dm*, which was created by the deindexing module as discourse referents. Once the referent of the definite description is identified, it is asserted to be equal to the discourse marker. This allows the discourse module to resolve subsequent references to the same object. This same technique is used for any construct that creates an object in the discourse history. For example, explicit existential quantification can create a discourse referent as well, as seen by the discourse fragment There is an engine at the station. It has overheated. The pronoun

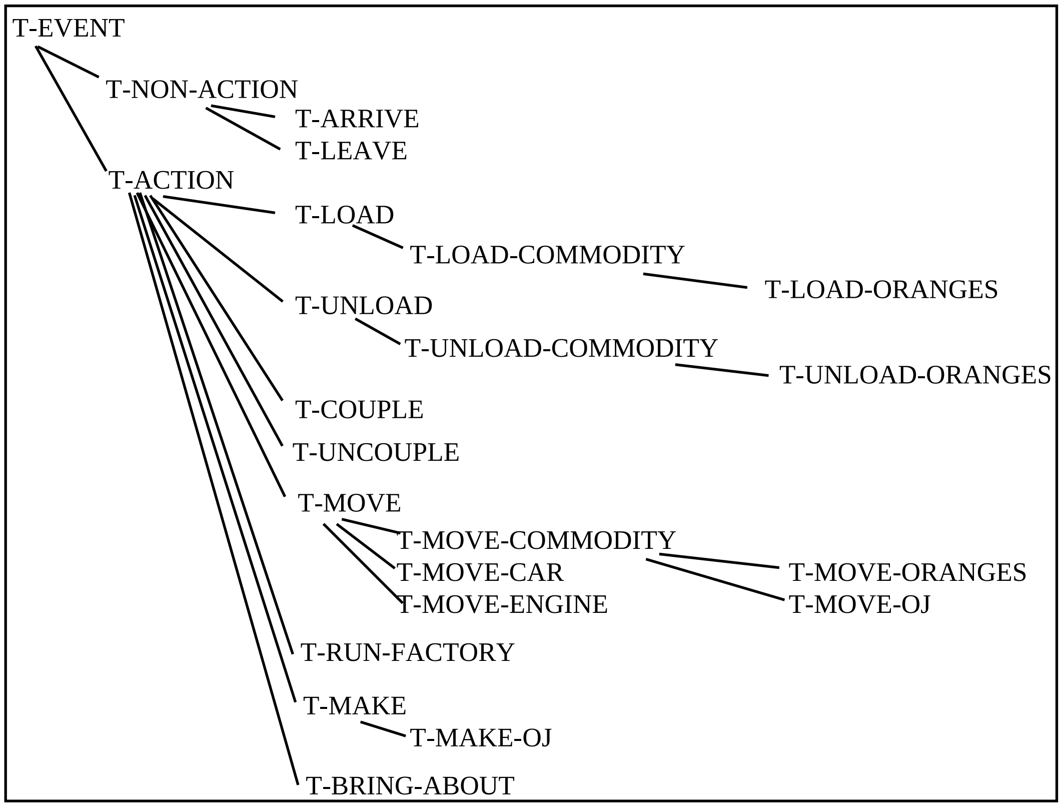

Figure 12: The event hierarchy of TRAINS domain events

T-INFORM	The speaker aims to establish a shared belief in the proposition asserted
T-YNQ	The speaker asks a yes-no question, creating an obligation for the hearer to respond
T-CHECK	The speaker is verifying that a certain proposition is true (that the speaker already suspects is true)
T-SUGGEST	The speaker proposes a new item (action, proposition) as part of the plan
T-REQUEST	The speaker aims to get the hearer to perform some action. In the TRAINS domain, this is treated like a suggest, with the addition of an obligation on the hearer to respond.
T-ACCEPT	The speaker agrees to a prior proposal by the hearer.
T-REJECT	The speaker rejects a prior proposal by the hearer.
T-SUPP-INF	The speaker provides additional information that augments, or helps the hearer interpret some other accompanying speech act.

Figure 13: The illocutionary acts used in TRAINS-93

it in the second sentence refers to the engine introduced in the first sentence. Thus, there is an existential quantifier form in EBTL that contains a discourse marker, namely

`(LF-EXISTS` *var dm restriction form*`)`.

EBTL also contains terms that explicitly denote plans and predicates that are used to describe the content of plans. Three important predicates often occur in the content of suggestions, namely:

`(GOAL-OF` *proposition plan*`)`: the proposition is a goal of the indicated plan

`(USES` *object proposition plan*`)`: the object is used in the plan, in part because of its property described in the proposition.

`(PLAN-EVENT` *event plan*`)`: the event is part of the plan.

For example, the EBTL representation of the suggest interpretation of "We better ship a boxcar of oranges to Bath by eight a.m." would be a suggest act with the following content (after simplifying some lambda expressions):

```
(LF-EXISTS ?P*T-PLAN ?DM NIL
  (GOAL-OF
    (LF-EXISTS ?V*T-ORANGES [X1381] (QUANTITY ?V (BOXCAR-UNIT 1))
      (LF-EXISTS ?M*T-MOVE-COMMODITY [M11]
        (AND (ROLE ?M R-COMMODITY ?V*T-ORANGES)
             (ROLE ?M R-DESTINATION BATH)
             (BEFORE [F-TIME ?M] [8AM]))
        (OCCURS ?M)))
  ?P))
```

This formula is true if there is some plan has a goal that there exists a boxcar of oranges that is moved to Bath before 8AM. The dialogue manager would have to establish what plan is being talked about here based on the discourse context. In this case, since it is the beginning of the dialogue, it is a new plan that is created as part of interpreting this first sentence.

6 An Overview of the TRAINS Modules

Now that the overall system architecture has been discussed, and the knowledge representation languages defined, we can describe the individual modules of the TRAINS-93 system in more detail. The components related to parsing are described in the first three sections. Section 6.1 describes the grammatical formalism that drives the syntactic processing and semantic interpretation. Section 6.2 describes lexical processing and Section 6.3 describes the stochastically-driven parsing algorithm used to find the most plausible interpretation of the input. The next three sections describe the discourse processing. Section 6.4 describes the deindexing module, and Section 6.5 describes the conversational act interpretation.

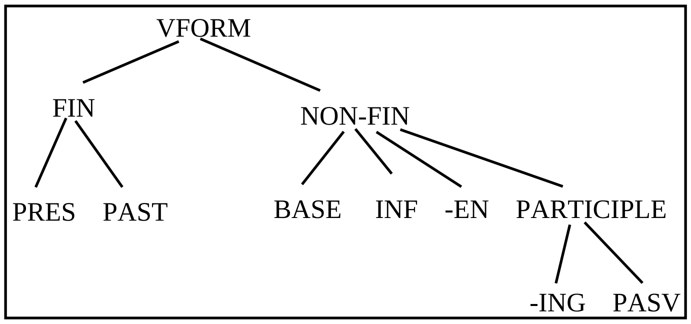

Figure 14: The VFORM feature tree

Section 6.6 describes the dialogue manager. The final two sections describe the domain reasoning components. Section 6.7 describes the domain plan reasoner and Section 6.8 describes the execution planner.

Given the space limitations of this paper, these descriptions are necessarily short on details. More information on each of the modules can be found in the references and in forthcoming papers in the TRAINS technical note series.

6.1 Syntactic and Semantic Rules

The NL front end of TRAINS consists of a chart parser working off a context-free grammar that directs computes logical forms expressed in EL in rule-by-rule fashion.[4] The grammar is like generalized phrase-structure grammar (GPSG) [Gazdar *et al.*, 1985] in its use of categories consisting of features, its treatment of unbounded movement in terms of slash features, and its reliance on feature principles such as the head feature and foot feature principles to constrain feature percolation. It organizes feature values as trees, rather than as sets of alternatives.[5] For instance, there is a VFORM (verb form) feature tree as shown in Figure 14. Thus, rather than the VFORM feature having the values FIN (*i.e.*, tensed) and NONFIN (*i.e.*, non-tensed), it splits into FIN and NONFIN, and FIN in turn splits into PRES and PAST, and so on. In other words, this tree is viewed as a type hierarchy of predicates applicable to verbs (and their projections), where the immediate subtypes of each type are exhaustive and mutually exclusive. Unification of two features lying on a "vertical path" in a feature tree yields the lower (more specific) feature, while unification of features not lying on such a path yields nil. The feature trees are non-overlapping, *i.e.*, the features in any one tree are distinct from the features in all other trees. The feature-tree approach leads to a quite compact syntax for grammatical rules.

Figure 15 shows a few sample phrase structure rules, paired with corresponding semantic rules (with slight modifications for readability). Subcategorization restrictions are handled

[4]See [Allen, 1994] for an introduction to chart parsing and rule-by-rule interpretation.

[5]There is some similarity to the grammatical sort hierarchy in HPSG [Pollard and Sag, 1994], though HPSG does not subdivide atomic features values.

```
S-tell  ;; e.g., [we have to make OJ] [.]
        :SYNRULE (S tell) <-- (S decl) (PUNC tell)
        :SEMRULE (Decl 1)
S_n2+v2  ;; e.g., [we][have to make OJ]
        :SYNRULE  (S decl) <-- (NP nom) (VP)
        :SEMRULE [1 2]
V2_V+V2inf ;; e.g., [have] [to make OJ]
        :SYNRULE (VP) <-- (V _V2inf) (VP inf)
        :SEMRULE (l x (1 [x 2]))
V2_V+V2base ;; e.g., [will/did/to] [make OJ]
        :SYNRULE (VP) <-- (V _V2base) (VP base)
        :SEMRULE (l x (1 [x 2]))
V2_V+N2pred ;; e.g., [make] [OJ];  [start] [a fire]
        :SYNRULE (VP) <-- (V _N2pred) (NP pred acc)
        :SEMRULE (1 2)
```

Figure 15: Some simple grammar rules

by features like `_V2inf`, which apply to verbs that takes an infinitive-form verb phrase complements. These are currently atomic features, though eventual use of complex features is planned. In the semantic rules, round and square brackets are interpreted as in Episodic Logic, `l` is the usual lambda operator, and angle brackets indicate an unscoped operator. Also, numeric arguments 1, 2, ... in a semantic rule refer to the semantic values (logical forms) of the first daughter, second daughter, *etc.*, shown on the right hand side of the syntactic rule.

A set of lexical entries adequate for parsing "We have to make OJ" is shown in Figure 16. The lexical entries are automatically augmented with default features where these do not conflict with explicitly assigned features. For example, verbs receive default features `immobl` (cannot head a question), `numb` (agrees with any number), `per` (the root of the person-tree, indicating agreement with any person), `base`, and `main` (can follow emphatic "do"). Thus the verb `have1` would receive the additional features `immobl`, `per`, and `main`.

The unscoped logical form determined by these rules for "We have to make OJ" (after three lambda-conversions) is

```
(<pres must> [we (make OJ)])
```

or after scoping of the pres operator would be,

```
(pres (must [we (make OJ)])).
```

This is an EL formula asserting that there is a present requirement that we make OJ. The symbols `pres` and `we` in this formula are indexical expressions to be resolved in later processing.

```
we1
        :SYNRULE (NP nom 1per plur anim pron) <-- we
        :SEMRULE we
have1 ;; e.g., "We [have] to make OJ"
        :SYNRULE (V _V2inf pres plur) <-- have
        :SEMRULE <pres must>
 to1   ;; infinitive auxiliary
        :SYNRULE (V inf aux _V2base) <-- to
        :SEMRULE (l x x)  ; \ie, identity function
make2
        :SYNRULE (V _N2pred) <-- make
        :SEMRULE make
OJ1
        :SYNRULE (N 1bar mass) <-- OJ
        :SEMRULE OJ
```

Figure 16: Some simple lexical rules

The TRAINS grammar consists of 246 phrase structure rules and 1844 lexical rules generated from a 770 word lexicon. It uses 326 features organized into 38 feature trees. Some sample sentences accepted by the grammar are shown in Figure 17. The grammar covers most of the basic English constructs, including general syntactic and semantic analyses of the following:

- Infinitives
- Treatment of tense, aspect and temporal adverbials
- Unbounded dependencies (*i.e.*, topicalization, wh-questions, relative clauses).
- Sentential complements (both tensed and untensed, *e.g.*, "we hoped the oranges would arrive before 8 p.m," "we suggested that you ship the oranges to Bath")
- Questions (both yes-no and wh-questions) and imperatives
- Indirect questions (*i.e.*, embedded questions such as "I know who fixed engine E3")
- Prepositional phrases, such as "put the oranges in the box," "the OJ factory in Avon," "shipped it in a hurry," "drive along the rail," clausal adverbials, *e.g.*, "since," "until," and "after" clauses, and modal adverbials, *e.g.*, "without doubt"
- Perception sentences and other sentences involving causative verbs with naked infinitive complements
- Certain forms of ellipses
- Various auxiliaries (including modal ones)

We have to get one tanker of orange juice to Avon and a boxcar of bananas to Corning.

We have to do both by 3 p.m.

So, let's start with the orange juice.

In order to get orange juice to Avon, we have to get the oranges to the orange juice factory, and then the orange juice to Avon.

Essentially, we have to, again, get the boxcar and an engine to Corning.

So, the fastest way to do that is from Elmira.

I assume one boxcar of oranges is enough to make a tanker of orange juice.

Figure 17: Some sample sentences that the TRAINS grammar can handle

- Certain types of partitives and measure terms (*e.g.*, "two gallons of OJ")

There are some areas where the grammar needs to be extended, including a treatment of quotation and parentheticals, some complex referring expressions (*e.g.*, "From Avon to Bath—do you know how far it is?"), complex quantifiers ("almost every one in this room"), and gerundive and participial adverbials ("upon receiving the shipment," "seen from the east"). Problems also arise with some counterfactuals sentences (*e.g.*, "we get back to Corning 2 hours later than we would have"), some compound nominals (*e.g.*, "if we do the route which the banana guy isn't doing"), ordinals, some expressions of pragmatic importance (*e.g.*, "that seems to be the next closest one"), and handling mixed speech acts (*e.g.*, "now let me remember—which is the boxcar we've decided to use for the bananas?").

In retrospect, many of our original ideas in developing the grammar have turned out well. Specifically, the following are positive conclusions:

- The compositional hypothesis has worked out very nicely, with Montague style rule-to-rule syntax-semantics mapping. The implementation of the semantic interpretation mechanism is simple and semantic translation is obtained as a by-product of parsing.
- The fully-declarative grammar, based on general principles rather than domain specific characteristics, made scaling up no problem.
- The tree organization of features turned out to be convenient for stating the grammar, and efficient to use in the parser.

6.2 Lexical Processing

While it was easy to construct the full set of lexical entries for the test dialogues by hand, there is ample evidence that lexicon building is a never-ending process. In the TRAINS dialogues, for instance, the following words occur that are not present in the 60,000 word

Alvey lexicon [Ritchie *et al.*, 1992]: *simultaneously*, *unhitch*, *unsolvable*, *independently*, *approximately*, *solvable*, *decouple*, and *concurrently*. The TRAINS lexical analyzer can produce meaning postulates for classes of words like these that involve derivational affixes.[6]

We use knowledge of the meaning of derivational affixes to hypothesize semantic interpretations of previously unknown words. A new predicate is created for the new word and meaning postulates are posited based on the affix and stem involved. Such an approach to derivational affixes has not been tried previously in computational linguistics, although the idea was proposed by Chierchia and McConnell-Ginet [1990]. Consider an example. The verb *couple* is in the TRAINS lexicon, and describes an action that results in a state where the engine and car are linked together. The word *decouple,* on the other hand, is not in the lexicon as it did not appear in the corpus. When faced with this word, the lexical analyzer would first parse the word into an affix and stem, namely *de-* and *couple*. A new predicate, `DECOUPLE1`, would be defined in the system to stand for the meaning of the new word. The system's knowledge of the uses of the affix *de-* would indicate that *decouple* is, like the interpretation of the word *couple* (say `COUPLE3`), a verb that has a definite resulting state. In addition, it would add a meaning postulate that asserts that the resulting state of a `DECOUPLE1` action is the negation of the resulting state of `COUPLE3`, namely not-attached. Once this information is defined, the system can syntactically and semantically process the sentence as usual.

To make this technique truly robust and general, however, will require the construction of large lexicons with detailed semantic entailments for each entry. This will be very time consuming, but there does not seem to be a way to automatically derive this sort of rich semantic information about word meanings in any other way. More details on this approach can be found in [Light, 1993].

While all the inflectional forms of words used in the demonstration dialogues were hand-coded in the lexicon, words outside this core set are processed using the Alvey morphological analyzer [Ritchie *et al.*, 1992], which itself uses a Kimmo-based segmenter and a unification-based chart parser. Inflectional affixes are treated compositionally where each inflectional process introduces an operator that transforms the meaning of the stem into the meaning of the inflected form. One problem we faced was the inefficiency of chart-parsing for computing word structures. For example, it took 20 minutes on a Sparc 10 to parse the word *reconsideration*. There were 44 entries in the lexicon that were considered during the parse, including words and affixes such as *r-*, *re-*, *ret-*, *con-*, *-side*, *rat-*, *-ion*, *-ration*, and, of course, *consider*). Some sort of preference-based morphological parser is obviously needed to produce the most likely readings quickly.

[6]In morphology, there is an important distinction between inflectional and derivational forms. Inflectional forms typically relate to agreement and other phenomena in language. The inflected form is the same syntactic category as its root form, and there is a systematic relationship between the meaning of the derived word and that of its base form. The verb *booked,* for example, indicates the past tense form of the verb *book.* Derivational forms, on the other hand, often change the syntactic category and can have quite arbitrary effects on the word's meaning. But many are quite systematic. The prefix *un-*, for instance, typically produces the opposite meaning (compare *ungrateful* with *grateful*).

6.3 The Parser

The parser used in the TRAINS-91 system was a simple left-corner backtracking parser, which was implemented quickly for use in the demo. Even in the TRAINS-91 system, where the grammatical coverage was limited to the specific sentences in the demonstration dialogues, it was difficult to ensure that the parser identified the appropriate analysis of each sentence. With the push to develop a grammar that could cover most of the TRAINS corpus, some efficient disambiguation techniques were essential. First we implemented an optimized chart parser with top-down filtering as a basis for the TRAINS-93 parser, but it did not perform well on the expanded grammar. In fact, some sentences that could be accepted by the grammar in principle could not be parsed because of memory limitations. More telling, in a test suite of 200 sentences that we know the grammar can accept, if we limited the parser to 5000 parser operations,[7] fully 40% of the sentences cannot be parsed. Even when a parse was found, about half the time it was not the correct parse. For those who have built large-scale parsers, a 50% accuracy rate might seem like a promising start, but many utterances in the TRAINS corpus are very simple, such as the single word Okay. So a more revealing accuracy measure is based on utterances that have more than one possible parse given the grammar. For these, the chart parser produced the desired interpretation only 24% of the time. Combining this with the fact that 40% of the sentences were not parsed at all, yields an overall accuracy rate of less than 15% on the test sentences!

To improve both the accuracy and the efficiency of the parser, we explored a range of stochastic techniques. We first tried a pure probabilistic context-free parser (PCFG), but it did not improve things much. The PCFG on the same set of sentences achieved an accuracy rate of only 36%, and still failed to produce a parse on over 40% of the utterances, yielding an overall accuracy rate of less than 22%. So we explored a different approach. Rather than using probability theory to evaluate the likelihood that a constituent will lead to the correct parse, we used probability theory to evaluate parser search decisions given the local context of the sentence. The idea is very simple. The parser is a chart parser that uses trigram probabilities to select which active arc to extend next. As a result, a constituent is built using a rule R, only if each arc extension involving R is favorable based on the trigram probabilities computed from the input. The system does not compute a probability for a constituent once it has been found. Rather, it is added to the chart and is available for any arc extension operation. To see the difference between our approach and standard probabilistic parser, consider the estimates used to parse the noun phrase "the boxcar at Dansville" in the sentence "the boxcar at Dansville is empty." In the PCFG approach, the probability of the constituent NP would be computed as follows

$$\begin{aligned}
&P(\mathtt{NP} \mid \textit{The boxcar at Dansville}) = \\
&\quad P(\mathtt{NP} \leftarrow \mathtt{NP\,PP} \mid \mathtt{NP}) \cdot P(\mathtt{NP} \leftarrow \mathtt{DET\,N} \mid \mathtt{NP}) \cdot \\
&\quad P(\mathtt{PP} \leftarrow \mathtt{P\,NP} \mid \mathtt{PP}) \cdot P(\mathtt{NAME} \mid \mathtt{NP}) \cdot P(\textit{the} \mid \mathtt{DET}) \cdot \\
&\quad P(\textit{boxcar} \mid \mathtt{N}) \cdot P(\textit{at} \mid \mathtt{P}) \cdot P(\textit{Dansville} \mid \mathtt{NAME})
\end{aligned}$$

where $P(\mathtt{NP} \leftarrow \mathtt{NP\,PP} \mid \mathtt{NP})$ is the conditional probability of rule $\mathtt{NP} \leftarrow \mathtt{NP\,PP}$ being used given that you have an NP. Note that this approach would assign the same probability to a

[7]Since this is a chart parser, we implement this by allowing only 5000 active arcs to be constructed.

proposed $\mathtt{NP}$ built from "Dansville at the boxcar," as the same rules and lexical probabilities are involved. In contrast, our parser would build an NP from "the boxcar at Dansville" only if all the following parser transitions were at one stage the best rated operation in the parser (where $\bullet$ is the "dot" operator showing how much of the rule has matched so far, and the second word in the sequence is the current word being processed, so

$$P(\mathtt{NP} \leftarrow \mathtt{DET} \bullet \mathtt{N} \mid \textit{the boxcar at})$$

is the conditional probability of the rule $\mathtt{NP} \leftarrow \mathtt{DET\,N}$ being completed by a noun given the input word "boxcar" is preceded by the word "the" and followed by the word "at."):

$$\begin{gathered}
P(\mathtt{NP} \leftarrow \bullet\, \mathtt{DET\,N} \mid \emptyset \textit{ the boxcar}) \\
P(\mathtt{NP} \leftarrow \mathtt{DET} \bullet \mathtt{N} \mid \textit{the boxcar at}) \\
P(\mathtt{NP} \leftarrow \bullet\, \mathtt{NP\,PP} \mid \emptyset \textit{ the boxcar}) \\
P(\mathtt{NP} \leftarrow \mathtt{NP} \bullet \mathtt{PP} \mid \textit{boxcar at Dansville}) \\
P(\mathtt{PP} \leftarrow \bullet\, \mathtt{P\,NP} \mid \textit{boxcar at Dansville}) \\
P(\mathtt{PP} \leftarrow \mathtt{P} \bullet \mathtt{NP} \mid \textit{at Dansville is}) \\
P(\mathtt{NP} \leftarrow \bullet\, \mathtt{NAME} \mid \textit{at Dansville is})
\end{gathered}$$

This approach would assign a completely different probability to the NP "Dansville at the boxcar" and would never build such an NP constituent unless it were the only possible interpretation. Our algorithm does not estimate the probability of constituents, rather it constrains the insertion of constituents onto the chart. All constituents that are added to the chart are equivalent as far as the likelihood of being used in further operations is concerned. This eliminates the problem found in many algorithms where the probability of a constituent decreases in proportion to the length of input it covers. The probabilities for lexical trigrams above are approximated using category trigrams and lexical probabilities for each category.

Most similar to our approach is the trigram-based component of the algorithm described by Magerman and Weir [1992]. Beyond this similarity, however, the differences between the approaches are significant. They focus on computing the probabilities of constituents, and condition the probabilities on the parent rule. We have not yet be able to compare this algorithm to our own.

The results of our tests show substantial promise. Out of our corpus of 200 parsed sentences with than one interpretation, we successively selected subsets of the dataset as a test set and then trained the grammar on the remaining sentences. This allowed us to do extensive testing with a fairly small corpus using the cross-validation method. As before, we limited the parser to 5000 operations. Our stochastic parser achieved an accuracy rate of 58% on these sentences, and found a parse for all but 6% of the sentences, giving an overall accuracy rate of 54%. A comparison of the results is shown in Figure 18. The average search factor gives the number of arcs proposed for each correct arc used in the final interpretation. Thus a factor of 6.2 means the parser generated 6.2 arcs on average to find one arc in the solution.

Thus the method appears to have strong promise. To see whether it will improve with further training, we also did the same test on the training data. Our parser achieved a

Method	Percentage parsed in 5000 edges	Accuracy rate for sentences parsed	Overall accuracy rate	Average search factor
Our method	94%	58%	54.5%	6.2
PCFG	59%	36%	21.2%	15.4
Chart parser w/top-down filter	61%	24%	14.6%	27.5

Figure 18: Sentence accuracy on test set when limited to 5000 edges

99% accuracy rate when run on the training data, with no sentences failing to parse. The average search factor was only 1.1. In contrast, the PCFG on the same test achieved only a 58% accuracy on the 74% of the sentences that were parsed. Thus, we can expect significant further improvement on our parser as the training set of parsed sentences is increased.

6.4 Deindexing

The role of the deindexing module is to interpret all indexical expressions, including finding plausible hypotheses about the referents of referring terms and resolving scope ambiguities that arise with quantified terms and with operators. There is a large literature on these issues, typically expressed in terms of syntactic preferences for certain scopings. For example, many researchers have noted the tendency for quantifiers in the subject position to take wide scope. Thus the preferred reading of the sentence "Every boy ran in one race" is that there are many races involved, whereas for "One race was run by every boy," the preference would be for one race. The problem with such rules is that the preferred interpretations can be strongly influenced by the previous discourse context and commonsense knowledge about which interpretation is more likely. The key idea explored in this work was that scoping decisions are parasitic on the rest of discourse interpretation. Referent determination is performed on unscoped forms, and then the possible scopings are constrained in order to support the most likely referential interpretations. It may turn out that some scoping decisions are never made at all if they do not affect the interpretation of the sentence one way or another.

Thus, a distinction is made between semantic ambiguity and perceived ambiguity. Even though a sentence may have a very large number of alternative readings (and hence is semantically ambiguous), it may be that all these readings do not have to be generated because the ambiguity can be represented implicitly in an 'underspecified' representation. Clearly a key prerequisite for this research is a representation that supports inference on such underspecified representations. Poesio [1994] discusses a model-theoretic semantics for unscoped forms, where an unscoped formula denotes a set of fully-scoped propositions, one for each possible interpretation.

The theory of discourse interpretation on which the deindexing module is based is Conversation Representation Theory (CRT), an extension of Discourse Representation Theory

(DRT) [Kamp, 1981]. CRT is designed to deal with semantic ambiguity and to allow the representation of pragmatic information present in conversations, such as the presence of multiple discourse topics and the organization of utterances in discourse segments. The deindexing module maintains and reasons about DRSs, which can be thought of as generalizations of theories of first-order logic: in addition to a set of facts, they also include a set of discourse referents (or discourse markers). The operations on DRSs can be thought of as generalizations of inference rules, where, in addition to deriving new facts, they can also create new discourse referents. One thing that distinguishes CRT from DRT is that most of the rules used are interpreted as default rules, and thus they generate hypotheses that might be overridden by other hypotheses. Thus, the module itself can be viewed as a hypothesis generation system that operates in a forward chaining manner. It maintains a queue of hypotheses, then applies discourse interpretation rules to produce a new set of hypotheses, pruning away any that are equivalent to already existing hypotheses. These rules assign referents, such as assigning an anaphoric interpretation to a definite noun phrase, determine the scope of operators, and so on. The behavior of the module is defined in a completely declarative way.

The input to the deindexing module is a parse tree with each node annotated with its semantic interpretation in unscoped EL. A preliminary hypothesis about the state of the discourse situation after the new utterance is obtained by deriving one or more *conversational events* from the underspecified representation, and adding them to a distinguished DRS, called the root DRS, that represents the state of the conversation prior to this last utterance. Consider for example sentence 9.1 in Figure 10, "So we should move the engine at Avon, engine E1, to Dansville and pick up the boxcar there." There are three reference problems in the sentence: the definite "the engine at Avon," the deictic expression "there," and the definite "the boxcar there." As far as scope is concerned, the main problem is to determine the relative scope of the modal "should" and the definite descriptions.

First of all, the module posits two conversational events: one corresponding to the occurrence of the discourse cue "so" (that in this context signals a passage to a new phase in the conversation), the other corresponding to the rest of the utterance. Then the module starts trying to identify a value for all the context-dependent elements of the interpretation of the sentence, applying all of its rules in parallel. Every time the system manages to resolve all of the contextual aspects of the interpretation of a sentence constituent, it applies a "construction rule" like those of DRT which has the effect of a scope assignment rule. For example, as soon as the system is able to identify the referent of the engine at Avon (engine E1) no more contextual aspects of the definite need to be resolved, and therefore the definite can get its scope. The hypothesis behind this is that all operators that need scoping have some contextually dependent element to be resolved, and all sentence constituents that have no such contextually dependent aspect are left unscoped. In the case of 9.1, the deindexing module, besides resolving the two definite descriptions and the deictic "there," also hypothesizes that the modal "should" should be interpreted with respect to the plan, *i.e.*, that it means that there is (at least) one course of action that will achieve our goal, and this is to move the engine at Avon, rather than meaning that we have social obligations to move the engine.

Because all hypotheses are generated in parallel, the module might end up generating distinct hypotheses that differ only in scoping even when all contextual aspects of the interpretation are resolved in the same way. This problem is prevented by pruning any hypothesis that assigns the same values to the context-dependent elements as an existing hypothesis. The module does generate multiple hypotheses, however, when some referential element can be resolved in multiple ways. For example, in utterance (11.1), the pronoun "it" can refer to either the engine or the boxcar. Most of these ambiguities could be resolved by using commonsense knowledge (*e.g.*, once two objects are coupled together, if you move one the other moves as well, which makes the two hypotheses equivalent).

The resolution strategy that leads the module to interpret "there" to refer to Dansville (rather than Avon, also mentioned in the sentence) is based on two hypotheses: (i) that "there" refers to the focus of attention, and (ii) that verbs of movement may move the focus of attention to the final destination of the movement. The module when interpreting the sentence hypothesizes that the manager may have intended for them to have Dansville as a mutual focus of attention after uttering the sentence, which enables the module to interpret "there". Poesio [1994] discusses some of the complexities in modeling the movement of focus of attention.

Finally, the deindexing module is responsible for introducing episodic variables into the formula and deindexing temporal expressions. In addition to tense operators, a preliminary LF may also involve various indexical temporal expressions such as aspect operators, indexical terms (*e.g.*, "tomorrow"), and time adverbials. These expressions need be deindexed with respect to the current context. As a very simple example, the sentence "E3 picked it up" is initially translated into

$$(\mathit{past}\ [E3\ \mathit{pick\text{-}up}\ \mathit{It}]).$$

After deindexing, this becomes

$$(\exists e26 : [[e26\ \mathit{before}\ u25] \wedge [e23\ \mathit{orients}\ e26]] \,.\, [E3\ \mathit{pick\text{-}up}\ C1] ** e26]).$$

where $u25$ is the episode representing the utterance and $e23$ is an episode described in an earlier utterance that provides the context for this utterance. We have developed *tense trees*, part of the context structure, and a set of deindexing rules [Hwang and Schubert, 1992], that transform the indexical logical form into a context-independent LF, simultaneously updating the tense tree component of the context structure. Tense trees allow us to automatically identify orienting episodes (as indicated by the *orients* relations in the above formula).

One obvious weakness of the current implementation is the lack of two-way interaction with the domain reasoner. For instance, it is difficult to handle references to objects that are implicitly introduced in the results of actions. Consider the dialogue fragment

> Send the boxcar to Avon and unload the oranges. Meanwhile, send the engine to Avon, hook it up to *the empty boxcar* and send it to Bath.

The problems arise in interpreting the NP "the empty boxcar." The fact that the boxcar is empty requires knowledge about the effects of the unload action. Also note that the fact that the boxcar is at Avon is also an implicit constraint on the referent that would have to be derived from the fact that the engine is at Avon, and so cannot be hooked up to any car unless it is at Avon as well. In such cases, the deindexing module may delay the identification of the referent and leave it for the domain reasoner to determine during plan recognition.

6.5 Conversation Act Analysis

Conversational action is represented using the theory of Conversation Acts [Traum and Hinkelman, 1992] which involves interactions at four different levels:

- turn-taking: acts that maintain, release, or take the turn in the dialogue;
- grounding: acts that deal with establishing shared knowledge about the dialogue, *e.g.*, acknowledgments and repairs (as in [Clark and Schaefer, 1989]);
- core speech acts: the illocutionary acts found in traditional approaches, *e.g.*, inform acts;
- argumentation: acts that characterize the relationship between utterances (*e.g.*, one utterance elaborates on another).

Each utterance will generally contain acts at each of these levels.

The analysis of core speech acts and argumentation acts occurs in two phases in the TRAINS system, as described by Hinkelman and Allen [1989]. The first phase (interpretation) uses syntactic and semantic patterns that indicate common conventional readings for various utterance forms. For example, there is a rule that says that sentences of the form "Can you do X?" are often a request to do X, although they can also be interpreted literally as a yes-no question. Sentences of the form "Why not do X?" on the other hand, typically signal suggestions, and any utterance that uses the adverbial please is a request. Once all the possible conventional readings are constructed, the second phase (pruning) is a filtering process, where each possible interpretation is considered in context to see if it is reasonable. As described earlier, the most effective filters are based on checking the preconditions of the acts.

In the TRAINS-91 system, the speech act interpreter operated on the logical form of the utterances after being converted into EBTL. While this was sufficient for the initial demo, it was clear that the approach was limited because the module could not use information about the syntactic and semantic structure of the original utterance. As a result, in the TRAINS-93 system, the speech act interpreter works directly on the unscoped forms created by the parser, extended with the discourse markers created by the deindexing module. The result of this module is a list of possible speech act interpretations represented in EBTL. Thus the speech act interpreter subsumes the old converter code and provides the transition from Episodic Logic into EBTL.

The candidate speech act interpretations are generated by rules that match syntactic and semantic properties of the parser output. Each rule that matches identifies a possible speech act and its propositional content. Consider a simple example: processing the sentence "There's an engine at Avon." The deindexer first associates a discourse marker with each major constituent in the parse tree, say D1 for the S structure (which is a proposition asserting the existence of an engine at Avon), and D2 for the NP (which is an indefinite object that is an Engine at Avon). The speech act interpreter has at least two rules that match this structure, characterized as follows:

- If utterance is a declarative sentence rooted at node N, then one possible speech act is an INFORM act of the content associated with the discourse marker of N.
- If utterance is a sentence of the form There is N, for some node N, then one possible speech act is a SUGGEST to use the object associated with the discourse marker of N in the plan.

These two rules produce two possible interpretations:

```
INFORM : (LF-EXISTS ?x*T-ENGINE [D2] T (AT-LOC ?x [AVON]))

SUGGEST: (LF-EXISTS ?x*T-ENGINE [D2] T
           (USES ?x*T-ENGINE (AT-LOC ?x [AVON]) <THE T-PLAN>))
```

Identifying which plan is being discussed is left for the dialogue manager to resolve. In this case it would be assumed to be the domain plan with the goal of shipping oranges, that is already under discussion. The pruning phase decides which of the potential acts actually occurred, as described in Section 5.1 above.

Grounding act analysis is based on comparing the current utterance to the discourse history to determine how the current utterance advances the state of mutual understanding. The current utterance might initiate new content, or continue, repair or acknowledge a previously initiated unit. The grounding theory is described in [Traum, 1994].

Turn-taking analysis is based on a determination of who is expected to speak next. In the current system with typed input delivered a full sentence at a time, every utterance is assumed to release the turn to the other speaker, unless the same agent produces the next utterance before conversation act analysis for the previous utterance has been completed. Special turn-taking signals are also provided, so that the manager can continue or release the turn explicitly without relying on this timing.

6.6 The Dialogue Manager

The dialogue manager (DM) is responsible for maintaining the flow of conversation and making sure that the conversational goals are met. To do this, it maintains a model of its mental state and the discourse context (including the turn, current domain plan, and grounding and plan-based focusing information). For TRAINS, the main goals are that an executable plan which satisfies the manager's domain goal is constructed and agreed upon by both the system and the manager and then that the plan is executed.

The DM updates the mental and conversational state based on the effects of observed conversation acts, deliberates on its conversational state, which may cause further updates.

When appropriate, it invokes the domain plan reasoner, the domain plan executor, and to the generator to generate utterances back to the manager. The model of mental state includes the following (as first seen in Figure 4):

- nested beliefs and proposals about the domain (*cf.* Figure 5);
- A set of current discourse goals representing phases and objectives in the TRAINS conversation (including getting a domain goal and agreeing upon a plan to satisfy the goal);
- A set of intended speech acts that the system will say when it gets a chance;
- A set of pending *discourse obligations.*

In a conversational setting, an accepted offer or a promise will incur an obligation. Also, a command or request by the other party will bring about an obligation to perform the requested action. If the obligation involves speaking, then we call this a discourse obligation. Our model of obligation is very simple and is based on a set of rules that encode discourse conventions. Whenever a new conversation act is determined to have been performed, any future action that can be inferred from the conventional rules becomes an obligation. We use a simple forward chaining technique to introduce obligations.

When deciding what to do next, the DM first considers obligations and decides how to update the intentional structure (add new goals or intentions) based on these obligations. If there are no obligations, then it will consider its intentions and perform any actions which it can to satisfy these intentions. If there are no performable intentions, then it will deliberate on its overall goals and perhaps adopt some new intentions (which can then be performed on the next iteration).

Special consideration must be given to the extra constraints that participation in a conversation imposes. This includes some weak general obligations (such as acknowledging utterances by others and not interrupting) as well as some extra goals coming from the domain setting to maintain a shared view of the world and the domain plans which are to be executed. To summarize this, the DM decides which task to perform at any given time using the following priorities:

1. Discourse obligations resulting from speech act effects (*e.g.*, to address a request by accepting or rejecting it).
2. A general obligation not to interrupt the other's turn. Unless something is more pressing, the system will wait rather than interrupt when the manager has the turn.
3. Intended speech acts. When the system has the turn and doesn't have any pending discourse obligations, it will perform acts which have been planned but not yet generated.
4. A general obligation to ground expressed content (*i.e.*, coordinate shared beliefs): Given that the system has the turn and has no obligations or speech acts it intends to perform, it should acknowledge utterances that have not been acknowledged, or request acknowledgment or repair its own utterances which have so far been ignored.

Discourse Obligations	none
Turn Holder	System
Intended Speech Acts	none
Unacknowledged Speech Acts	`INFORM-1` (need to get oranges to Bath) `SUGGEST-4` (goal is to ship oranges)
Unaccepted Proposals	none
Discourse Goals	`Get-goal`, `Build-Plan`, `Execute-Plan`

Figure 19: The discourse state after interpreting utterance 1.1

5. Discourse goals of domain plan negotiation. Given no higher priority goals, the system will address unsettled suggestions (*e.g.*, accepting or rejecting proposals made by the manager or making new proposals or requesting acceptance of proposals previously made by the system).

6. High-level Discourse Goals. Given no higher priority items, the system will attempt to further the conversation, *e.g.*, by calling the domain plan reasoner to expand the domain plan (in the system's private plan context), which will then provide the content for future system proposals.

The updating of the conversational state upon perceiving conversation acts progresses asynchronously with the other operations of the DM. Whenever the DM is active, it will first decide on which task to attempt, according to the priorities given above, and then work on that task. After completing a particular task, it will then run through the loop again, searching for the next task, although by then the context may have changed, say by the observation of a new utterance. The DM is always running and decides at each iteration whether to speak or not (according to turn-taking conventions); it does not need to wait until an utterance is observed, and need not respond to utterances in a strict utterance by utterance fashion.

As an example, consider the functioning of the DM when handling the first few utterances in the dialogue in Figure 10. Utterance 1.1, "We better ship a boxcar of oranges to Bath by 8 AM," is interpreted as performing two core speech acts. It is interpreted (literally) as the initiation of an inform about an obligation to perform a domain action (shipping the oranges). It is also seen as the initiation of an (indirect) suggestion that this action be the goal of a shared domain plan. The discourse state after interpretation of this utterance is shown as Figure 19. The system decides to acknowledge this utterance—moving the suggestion from an unacknowledged state to a proposed state—and then to accept the proposal, making it shared. The system acts on these intentions and produces the combined acknowledgment and acceptance in utterance 2.1, "OK." This makes the goal shared and also satisfies the first of the discourse goals, that of agreeing to a common goal.

Utterances 3.1, "So we need to get a boxcar to Corning, where there are oranges," and 3.2, "There are oranges at Corning," are interpreted in a similar manner, but not responded to yet since the manager keeps the turn (in this case by following up with

Discourse Obligations	`CHECK-IF` (oranges are at Corning)
Turn Holder	System
Intended Speech Acts	Acknowledge `INFORM-7` (need to get an engine to Corning)
Unacknowledged Speech Acts	none
Unaccepted Proposals	`SUGGEST-10` (subgoal to get engine to Corning), `SUGGEST-15` (use the oranges at Corning)
Discourse Goals	`Build-Plan`, `Execute-Plan`

Figure 20: The discourse state after utterance 3.3

subsequent utterances before the system has a chance to act). Utterance 3.3, "Right?" creates a discourse obligation on the system to respond to the Manager's assertion in 3.2 and also gives the turn to the system. The resulting discourse context after the system decides to acknowledge is shown in Figure 20. The system then proceeds to address the discourse obligation. It queries the knowledge base and decides that the manager is correct here (there are, indeed, oranges at Corning), and so decides to meet this obligation by answering in the affirmative. This results in forming an intention to inform, which is then realized (along with the acknowledgment of the utterances) by the production of utterance 4.1. More detail on the dialogue manager is available in [Traum and Allen, 1994].

6.7 The Domain Plan Reasoner

The domain plan reasoner allows the system to reason about the TRAINS domain. To support language understanding, it provides an algorithm (*incorporation*) that attempts to find causal and motivational connections between potential interpretations of the current utterance and the current plan. It also provides a planning capability (the *elaboration* algorithm), which is used to drive the system's own reasoning about the plan, and provides a question answering facility about the current plan and the state of the world. The domain planner provides reasoning services on demand. When given a task or query together with a plan object as the context, it supplies back the appropriate analysis for that task. These results can then be added to the appropriate context by the caller.

First consider the incorporation task. This subsumes what is typically called plan recognition in the literature, While plan recognition systems generally handle only atomic actions as input, the TRAINS plan incorporation algorithm handles a wide range of phenomena, all of which arise in the TRAINS dialogues. In particular, it has to be able to interpret the following:

- Utterances that suggest courses of action, *e.g.*, "Send engine E3 to Dansville." This is the typical case studied in the plan recognition literature.

- Utterances that identify relevant objects to use, *e.g.*, "Let's use engine E3" and "There's an OJ factory at Dansville." The second sentence is an indirect suggestion to use the OJ factory.

- Utterances that identify relevant constraints, *e.g.*, "We must get the oranges there by 3 PM" and "Engine E2 cannot pull more than 3 carloads at a time."

- Utterances that identify relevant lines of inference, *e.g.*, "The car will be there because it is attached to engine E1."

- Utterances that identify goals of the plan, *e.g.*, "We have to make OJ."

- Utterances that introduce complex relations, *e.g.*, purpose clauses such as "Use E3 to pick up the car" and "Send engine E3 to Dansville to pick up the oranges."

All of these affect the plan being discussed in some way, and finding the appropriate connection is crucial to understanding the intended meaning of the utterance.

To account for these interpretations, we must take a more general view of the plans than is typically found in the planning literature. In particular, our plan structures must be able to represent the motivations for actions and explanations for why a certain action will have the desired effect. To capture this, we have formalized a notion of *plan graphs*, which represent defeasible arguments that the planned actions will accomplish the declared goals. An example of a plan graph was shown in Figure 9. Formally, a plan graph is a graph consisting of nodes labeled by propositions in a temporal logic (in the case of TRAINS, with EBTL formulas). There are two classes of nodes: those that describe events (including action occurrences) and those that describe states. The link types correspond roughly to an intuitive classification of the possible relations between events and states: effect (action to state), enablement (state to action), generation (event to event) and justification (state to state).

We characterize plan reasoning as search through a space of plan graphs. The termination criterion for the search depends on the type of reasoning being done, as will be described shortly. Since the plan graph formalism sanctions arbitrarily complex graphs labeled with arbitrarily complex formulas, searching all possible plan graphs is impossible. We therefore rely on additional properties of the underlying representation to restrict the search.

First, we assume the ability to determine whether two objects (including events and states) unify and to identify assumptions under which they would do so. Second, we assume that events are defined using relations corresponding to enablers, effects, and generators. In the temporal logic, these descriptions can be obtained from the event definition axioms. The search then only considers plan graphs that reflect the structure of the event definitions. We call such plan graphs *acceptable*. In this respect, the search will only find plan graphs that can be constructed from the "event library." However, information returned from failed searches can be used to guide the repair of apparent incompatibilities at the discourse level, and the knowledge base can be augmented with a "bug library" to recognize common ways in which plans fail.

Incorporation of an event (or goal or fact) involves a search through a breadth-first expansion of the plan graph. The plan reasoner first checks if the event is already present in the plan. If so, the plan reasoner returns any equality assumptions that it had to make to allow the match. Otherwise, it expands the plan by adding the decompositions of events in the plan, and by following precondition and effect links. The search terminates when one or more possible matches are found at one level, or some preset depth bound is exceeded, in which case the domain reasoner gives up trying to find a connection. This reflects a hypothesis that if the connection is not fairly obvious, the utterance has probably been misinterpreted. If more than one possible connection is found at a level, the plan reasoner returns the set of choices so that the DM might plan a clarification subdialogue or repair.

As an example, consider the fourth utterance in the TRAINS-91 dialogue, "Engine E3 is scheduled to arrive at 3PM." After interpreting the first three utterances, the domain reasoner has constructed plan (a) shown in Figure 21, which involves moving oranges from city I to city B where they will be made into orange juice. To validate the suggest interpretation, the plan reasoner must find a connection between this fact and the plan. The arrival of engine E3 is not part of the current plan. Expanding the plan one step introduces a Move-Car event to generate the existing Move-Oranges. Expanding one more step introduces a Move-Engine action to generate the Move-Car event. A precondition of the Move-Engine action is that the engine must be at city I (the originating city). This fact can be established by the occurrence of the arrival action described in the utterance. Thus a causal connection has been found between the utterance and the plan, summarized in plan (b) in Figure 21. Because this connection is found, the interpretation of the utterance as a suggestion to use engine E3 after it arrives to move the oranges is considered plausible.

After interpreting utterance 1.5, "Shall we ship the oranges," the system has the turn and attempts to elaborate the plan. Plan elaboration uses fairly traditional means-ends planning to attempt to flesh out the plan. In so doing, it attempts to satisfy or assume preconditions and bind roles to objects in order to generate a complete plan. It freely makes consistent persistence assumptions by assuming inclusion of one unconstrained temporal interval within another known one. It also ignores some details of the plan, for example the exact route an engine should take. These can be reasoned about if necessary (*e.g.*, if the manager mentions them) but can otherwise be left for the execution planning stage. When some choice is encountered that must be confirmed by the manager, the planner stops the expansion of this part of the plan, and returns a goal of making the choice to the DM. For example, Figure 22 show the plan after the elaboration, containing a variable for the car to be used. The choice of car for transporting the oranges is important in the TRAINS-91 scenario because there is an empty car at Avon as well as one attached to E3. The plan will differ depending on which car is used. Rather than deciding, the plan reasoner returns a goal to resolve this ambiguity to the discourse module, which chooses one alternative and proposes it, leading to utterance (2.2).

No previous plan reasoner has provided such a wide range of capabilities as the TRAINS plan reasoner. Much of its power arises from its rich representational formalism, namely the temporal logic of events and action, and the treatment of plans as argument structures. The price we pay for this generality is a lack of the formal completeness and complexity results for planning that can be obtained in more expressively limited formalisms and, of

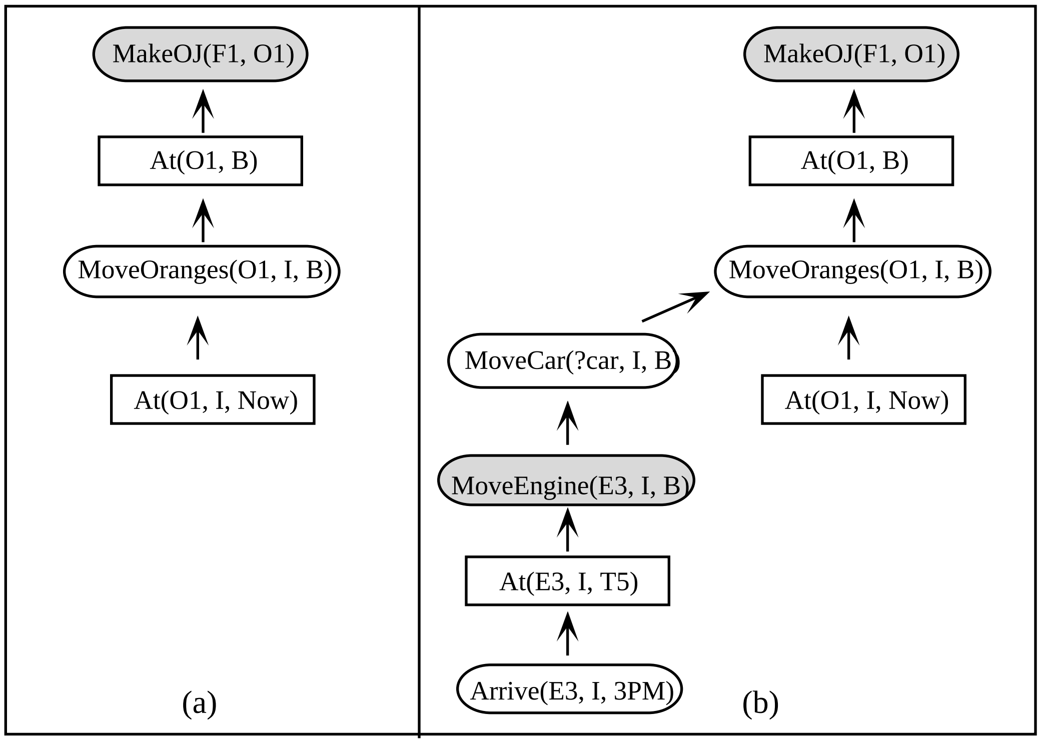

Figure 21: The plans after utterances (a) 1.3 and (b) 1.4 in the Trains-91 dialogue

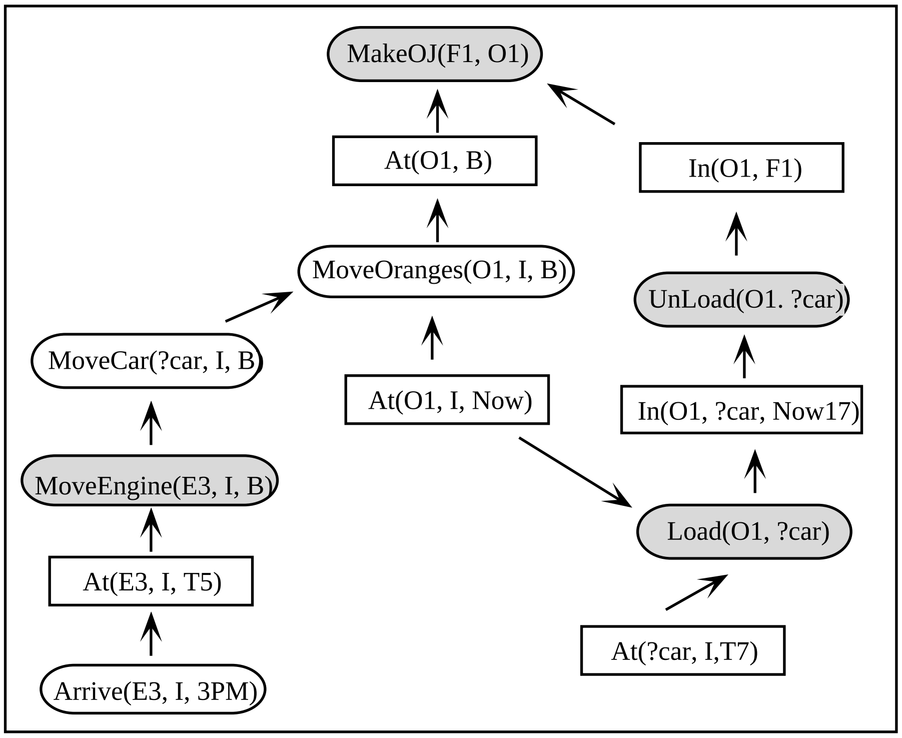

Figure 22: The plan motivating utterance 2.2 in the TRAINS-91 dialogue

course, the difficulty in defining effective algorithms. But we do not see that there is a choice here if one is to cover the range of reasoning needed to comprehend the dialogues. We are currently working on methods of formalizing the plan reasoning process that may allow us to construct some convergence proofs on the plan reasoning process: *i.e.*, given enough time, the plan would converge to a correct plan.

6.8 The Execution Planner

Once a plan has been agreed to by the system and the manager, the DM hands the plan to the execution planner. The execution planner extracts all of the events specified in the plan and looks for ways of making the necessary events occur. As mentioned earlier, the TRAINS System cannot directly cause any of the events specified in the plan because it does not have direct control of the trains, factories, or warehouses. Instead, it must request actions of the agents can cause these events. For example, if the plan specified that oranges should be loaded into a car at Avon and the car should be taken to Bath, it translates these two events into requests to the supervisor of the warehouse at Avon to load oranges into a car and to the engineer of the train connected to the car to drive to Bath.

The execution planner reasons about events at a different level of detail than does the domain reasoner. For instance, instead of reasoning about moving the oranges, it reasons about requests to the engineers and supervisors to move the oranges. Requests are in the form of condition-action pairs that request the agent to perform the action when the conditions become true. For example, in the TRAINS-91 scenario, the following message is sent to the factory supervisor `F1`, at city B:

```
(((IN C1 O1) (AT C1 CITYB)) (UNLOAD F1 C1))
```

This requests that when the oranges (`O1`) are in the boxcar (`C1`) and the boxcar is at city B, then the factory supervisor should unload the oranges. Besides having the agents perform various domain actions such as unload, load, couple, and so on, the TRAINS system may also request that an agent send back reports when certain conditions arise. The system might, for instance, send another message with the one above that requests that when the boxcar is unloaded, the supervisor should send back a report notifying the system. This way, the system can keep track of whether or not its requested actions have succeeded.

The execution planner must reason about what events need to be explicitly addressed in the requests. For instance, if it can request a TRAINS agent to execute actions to cause an event E directly, it does not need to also make requests for the events in the decomposition of E. If there are several different ways to accomplish some event, the execution planner will choose the one that it has found to be most reliable in the past. It collects statistics on the number of times requests result in reports that the requests were satisfied. Formally, it deals with estimates of conditional probabilities of the form $PROB(Occurs(E)|sent(M))$ for each event E and message M. The estimates are computed from the ratios of the number of times the event is reported to occur and the number of times the message is sent. In any situation, the planner chooses the messages that maximize the expected probability of success. Details on the actual formalism can be found in [Martin, 1993].

7 Other Issues

This section describes some additional issues of importance to the TRAINS project, but which are not part of the actual system. Specifically, it describes the TRAINS simulator and two additional projects addressing temporal reasoning and interpreting speech repairs.

7.1 The TRAINS Simulator

While not part of the TRAINS system per se, the TRAINS simulator is a crucial component of the research project. It simulates the physical attributes of the TRAINS World as well as the agents in this world. Unlike many other simulators, it explicitly models the actions of agents, rather than treating an agent's actions just like other simulated events. In the TRAINS Simulator, the actions of agents are asynchronous with respect to the events in the simulator. Agents can perform actions at any time in the course of the simulation and their actions cause cascades of events. The TRAINS Simulator provides a graphical interface that can animate the execution of plans, making it easy to detect interactions between tasks given to different agents.

By directly simulating agents, the simulator can model the exchange of information between agents. In particular, each simulated agent is capable of sending and receiving messages and acting in accordance with requests from the system. Agents obtain information about their local situation by simulated perception actions. Given a set of condition-action commands controlling its behavior, an agent continually perceives the state of its local area of the world to check if a rule applies. When a rule applies, it executes the action. Because there are many agents, the simulator can model the execution of plans that require decentralized execution.

The simulator is driven by a causal model of the world.[8] Every action by agents, as well as external events that occur spontaneously, can have probabilistic effects. By allowing the system designer to modify the causal model, it supports experiments in which the failure rate of actions can be varied, and in which external events may occur that may interfere with (or enhance) the likelihood that the current plan will succeed. The designer can also vary the clock time, so as to support a range of experiments from real-time planning, which requires a relatively slow clock rate, to automatic plan learning, which requires multiple runs done as fast as possible.

The simulator is designed so that it is easy to add new types of objects. The new types of objects can inherit properties from existing objects, such as speed and fuel capacity or behaviors like that ability to move to another location or the ability to store commodities. The current simulator supports railways, trucks, air transport and ships, each with their own different constraints on routes and cargo capacities. It also supports different forms of infrastructure: factories, ports, airports, warehouses, and so on.

[8]The causal model in the simulator is not accessible to the TRAINS system, as it is part of the external world. The world model used by the TRAINS reasoning components is typically less precise, and may be only a rough approximation of the actual model.

7.2 Temporal Reasoning

If Trains is ever to be scaled-up to realistic sized transportation problems, it must be able to efficiently reason about large numbers of activities over extended periods of time. We evaluated existing systems for temporal reasoning to see how well they scale up. The results were disappointing. Interval reasoning systems like our own TIMELOGIC [Koomen, 1989], which the current system uses, and MATS [Kautz and Ladkin, 1991], suffer from serious efficiency problems once they have to handle about 500 intervals, and become unusable before hitting 1000. Systems using constraint-satisfaction techniques for point-based reasoning fare better, but show degradation as the number of times enters into the thousands (see [Yampratoom and Allen, 1993]). The goal of the TIMEGRAPH-II system was to provide a temporal reasoning system that operated in close to linear time as the datasets grew.

As with all reasoning systems, there is an assert time/retrieval time tradeoff in comparing algorithms. Most temporal reasoning systems use constraint propagation techniques and compute out all consequences of information when it is asserted. This makes assertion expensive (order n^3 for point-based logics), but yields a constant time retrieval algorithm. The idea in TIMEGRAPH-II is not to compute all consequences and thus to sacrifice the constant time retrieval. It turns out, however, that temporal databases for natural language understanding and for planning have significant structure. Specifically, the times tend to be organized into long ordered chains. TIMEGRAPH-II exploits this structure, and provides constant time retrieval between times on the same chain, and a retrieval time proportional to the number of chains for the rest. Our empirical studies show that this is a substantial win with large datasets of train schedule information. More information on the algorithm can be found in [Gerevini and Schubert, 1993]. Extensions to deal with temporal disjointness, such as those needed to express non-overlap between actions are reported in [Gerevini and Schubert, 1994].

7.3 Speech Repairs

As mentioned earlier, one of the immediate problems facing a spoken dialogue system is handling speech repairs. Such repairs can be classified into three groups: modification repairs, where part of the earlier utterance is replaced with new material (*e.g.*, "How far is it from Avon to...from Corning to Dansville"), fresh starts, where the entire previous utterance is discarded, and abridged repairs, which consist solely of word fragments and filled pauses such as "um" and "uh." For our initial work, we focused on modification and abridged repairs, of which we annotated over 700 examples, using about a third of the Trains dialogues.

In looking at the data, it seemed that most modification repairs could be detected and corrected using only local information, without the aid of a parser. If this turned out to be true, it would considerably simplify the processing of speech as the parser would not need extensive modification to handle such phenomena. We found that most repairs are signaled by one of a few factors: word fragments (*i.e.*, a word is started but never finished), the presence of editing terms (*e.g.*, "um," "let's see," ...), and the repetition of words. Of course, there are also prosodic signals of repairs (*e.g.*, see [Nakatani and Hirschberg, 1993]), but it is not yet possible to extract such information reliably from the speech signal. So we worked solely from the information that is present in the transcripts. The technique we used involved two phases:

- prediction: a generalized pattern matching procedure which identifies potential repair sites by looking for word fragments, editing terms, and various forms of word repetition, and uses well-formedness constraints to filter out implausible repairs.
- verification: a statistical model, essentially a part-of-speech tagger augmented with information about repairs and repair signals, was used to weed out false positives.

This model is trained on part of our corpus of annotated speech errors in the TRAINS corpus.

The model worked much better than we expected. On a fair test on repairs in the TRAINS data that were not used for training, 80% of the repairs were detected and correctly made. Another 3% were detected but not corrected properly. The false positive rate was quite low, and the test results produced a detection accuracy figure of 89%. There is only space for one example, which shows that the algorithm works on a sentence that contains a sequence of overlapping repairs, a quite common phenomena. The input to the algorithm was

> ...and pick up um the en- I guess the entire um p- pick up the load of oranges at Corning

and the corrected output was

> ...and pick up the load of oranges at Corning

More details on this work can be found in [Heeman and Allen, 1994].

8 Discussion

It has been a large effort over several years to build the TRAINS systems. Was it worth it? As far as the goal of providing a research platform for research is concerned, it has been well worth the effort. Besides supporting many smaller-scale projects and investigations, the TRAINS project has provided the environment for five Ph.D. theses: Nat Martin (execution planning and monitoring), Massimo Poesio (deindexing, *i.e.*, reference resolution and scoping), and David Traum (the dialogue manager and discourse model), George Ferguson (domain reasoning, *i.e.*, interactive plan recognition and construction), and Marc Light (deriving the meaning of unknown words from their morphological structure). The system allowed these theses to explore different problems in depth but in the context of an overall system. As a result, we know that the simplifying assumptions made in each thesis are realistic assumptions to make. Many of the contributions described in this paper have resulted from these works.

As a demonstration system, the success of TRAINS depends on what you are concerned with. The system did demonstrate the feasibility of constructing a large-scale natural language system that is tightly integrated with a collaborative plan reasoning system, and showed how a wide range of different processing techniques (parser, reference resolution, discourse structure and plan reasoning) can be integrated. And many of the individual components demonstrate significant depth in their area of concern. But the fact remains that you cannot sit down in front of the system and spontaneously use it to solve a problem. The system is simply not robust enough. One factor in this is certainly the lack of resources to extensively test and debug the entire system over a wide range of dialogues. But the issue runs deeper than that. Even if the system were extensively debugged and tested on a wide range of dialogues, it would still perform poorly because it lacks any form of intelligent recovery when given an input it cannot fully understand. Given the nature of spontaneous dialogue, it is virtually certain that at least one utterance will be uninterpretable in almost any reasonable sized dialogue. In fact, this problem arises with human-human dialogue as well, except that humans use sophisticated methods to recover: by clarification subdialogues, requesting rephrasing, or simply ignoring what was not understood yet continuing with the task.

The current TRAINS system lacks a model of its own behavior. As a result, it cannot represent or reason about the fact that the last utterance could not be parsed, and so has no reasonable mechanism available to it to handle such situations. In fact, the current system simply ignores any unparsable input. It is not able to extract a partial understanding of what was said because the current parsing process is an all or nothing process. Similar problems of brittleness in message understanding systems or spoken language query systems have led to the development of different techniques for partial, robust understanding. It is clear that the TRAINS system needs to have such abilities. But such "shallow" understanding is not enough. Handling interactions in the TRAINS domain often requires a deep understanding of what is said in order to identify the implications of what has been suggested. Thus we need to retain full parsing and interpretation whenever possible. To handle this, the next versions of the TRAINS system will eventually run several understanding processes in parallel and then adjudicate between the answers when conflicting results are found. The system will also maintain an explicit model of its own actions, so that it can plan and participate in effective clarification subdialogues.

The other obvious issue is extending the system to handle speech input. The system was designed with a spoken dialogue in mind, but the input so far has been via the keyboard. Recent progress in speech recognition systems makes it feasible for us to start exploring this area. Our results on automatically handling speech repairs should help with one of the difficult problems in moving to speech, but there are many other issues beyond the obvious one of dealing with recognition errors that will greatly affect the system. First, spoken input will tend to be much more incremental, with information coming in a phrase at a time, and with sentence boundaries often not clearly marked (nor do they often seem relevant). Also, there are cases where the system should respond (say acknowledging that it understands a referring phrase) before the entire utterance has been produced. So the entire system must be modified to handle incremental understanding all the way through.

When viewing the system as a planning assistant rather than a natural language system, there are additional conclusions and concerns. First, on the positive side, the plan reasoning capability of alternately performing plan incorporation and plan elaboration is necessary for supporting mixed-initiative planning. Such a system has the potential to allow the human and system to best exploit their own strengths: the humans can use their expertise at making high-level decisions about the best strategy to search for solutions, whereas the system can use its capabilities to do detailed evaluation of plans, and to schedule large numbers of interrelated events. On the negative side, it seems clear that a natural language is not always the best medium for communicating all information about plans. We realized this from the start of the project, and the shared map was a way to delay dealing with this problem in the initial years of the project. To handle more complex plans and scenarios, however, it is clear that the system must be able to plan map displays and illustrative charts and graphs in addition to natural language, and that the manager is going to want a capability to point to parts of the map and to use a graphical user interface (GUI) for specifying and exploring plans. While this seems like a major change in direction, in fact the discourse and planning aspects of the system are well suited to these forms of interaction. Since display events and mouse gestures are forms of communication, they can be formulated as speech acts and planned in much the same way as planning speech acts realized by natural language utterances. One goal of the next TRAINS system is to demonstrate this by extending the system I/O capabilities to include a graphical user interface.

We should mention that we don't expect the natural language to be superseded by the GUI. There are many tasks for which natural language is most efficient, especially when discussing general problem solving strategies, performing clarifications, and in explaining and answering questions about plans. In addition, in applications where the user is a novice in using the system, the natural language interface provides an easily accessible channel for the user to start interacting and learn about the capabilities of the system as a whole.

The TRAINS domain has proved to be remarkably flexible, supporting work in a wide range of areas in natural language understanding and reasoning about plans. It provides an excellent testbed for studying complex dialogue behavior even with quite small and simple scenarios. On the other hand, scenarios can easily be expanded so as to challenge the best planning algorithms. So not only can we demonstrate results in the domain in the short term, there is no danger that this domain will become too simple or trivial in the foreseeable future.

The TRAINS project has been and will continue to be a valuable scientific exercise. It supports and encourages the development of theories that advance the state of the art in natural language processing, intelligent agent architectures, and reasoning about plans, among other areas. But as importantly, the requirement that the theories result in an implemented system provides concrete feedback on the theoretical work and produces data that can be used to guide future development. We believe that our work to date has laid a solid and broad theoretical foundation on which to build future generations of conversational and collaborative planning systems.

References

[Allen, 1984] James F. Allen, "Towards a general theory of action and time," *Artificial Intelligence*, 23(2):123–154, 1984.

[Allen, 1994] James F. Allen, *Natural Language Understanding*, Benjamin/Cummings, Redwood City, CA, 2nd edition, 1994.

[Allen and Ferguson, 1994] James F. Allen and George Ferguson, "Actions and events in interval temporal logic," *Journal of Logic and Computation*, 1994. To appear.

[Allen *et al.*, 1989] James F. Allen, S. Guez, Elizabeth Hinkelman, Keri Jackson, Alice Kyburg, and David R. Traum, "The Discourse System Project," Technical Report 317, Department of Computer Science, University of Rochester, Rochester, NY, 1989.

[Allen and Miller, 1991] James F. Allen and Bradford W. Miller, "The RHET system: A sequence of self-guided tutorials," Technical Report 325, Dept. of Computer Science, University of Rochester, Rochester, NY, 1991.

[Allen and Perrault, 1980] James F. Allen and C. R. Perrault, "Analyzing intention in utterances," *Artificial Intelligence*, 15(3):143–178, 1980.

[Barwise and Perry, 1983] J. Barwise and J. Perry, *Situations and Attitudes*, MIT Press, Cambridge, MA, 1983.

[Bratman *et al.*, 1988] Michael E. Bratman, David Israel, and Martha Pollack, "Plans and resource-bounded practical reasoning," *Computational Intelligence*, 4:349–355, 1988.

[Chierchia and McConnell-Ginet, 1990] G. Chierchia and S. McConnell-Ginet, *Meaning and Grammar*, MIT Press, Cambridge, MA, 1990.

[Clark and Schaefer, 1989] H. Clark and E. Schaefer, "Contributing to Discourse," *Cognitive Science*, 13:259–294, 1989.

[Cohen and Perrault, 1979] P. R. Cohen and C. R. Perrault, "Elements of a plan-based theory of speech acts," *Cognitive Science*, 3:177–212, 1979.

[Davidson, 1967] Donald Davidson, "The logical form of action sentences," in N. Rescher, editor, *The Logic of Decision and Action*. University of Pittsburgh Press, Pittsburgh, PA, 1967.

[Gazdar *et al.*, 1985] G. Gazdar, E. Klein, G. Pullum, and I. Sag, *Generalized Phrase Structure Grammar*, Basil Blackwell, Oxford, UK, 1985.

[Gerevini and Schubert, 1993] Alfonso Gerevini and Lenhart K. Schubert, "Efficient temporal reasoning through timegraphs," in Ruzena Bajcsy, editor, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, Chambery, France, 28 August–3 September 1993.

[Gerevini and Schubert, 1994] Alfonso Gerevini and Lenhart K. Schubert, "An efficient method for managing disjunctions in qualitative temporal reasoning," in *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR94)*, pages 214–225, Bonn, Germany, 1994. Morgan Kaufmann.

[Gross *et al.*, 1993] Derek Gross, James F. Allen, and David R. Traum, "The TRAINS-91 dialogues," TRAINS Technical Note 92-1, Department of Computer Science, University of Rochester, Rochester, NY, June 1993.

[Heeman and Allen, 1994] Peter Heeman and James F. Allen, "Detecting and Correcting Speech Repairs," in *Proceedings of the Thirty-Second Annual Meeting of the Association for Computational Linguistics (ACL94)*, pages 295–302, Las Cruces, NM, June 1994.

[Hinkleman and Allen, 1989] Elizabeth Hinkleman and James F. Allen, "Two constraints on speech act ambiguity," in *Proceedings of the Twenty-Seventh Annual Meeting of the Association for Computational Linguistics (ACL89)*, pages 212–219, 1989.

[Hwang and Schubert, 1992] Chung Hee Hwang and Lenhart K. Schubert, "Tense trees as the "fine structure" of discourse," in *Proceedings of the Thirtieth Annual Meeting of the Association for Computational Linguistics (ACL92)*, pages 232–240, 1992.

[Hwang and Schubert, 1993a] Chung Hee Hwang and Lenhart K. Schubert, "Episodic Logic: A situational logic for natural language processing," in P. Aczel, D. Israel, Y. Katagiri, and S. Peters, editors, *Situation Theory and its Applications*, volume 3, pages 303–338. CSLI, Stanford, CA, 1993.

[Hwang and Schubert, 1993b] Chung Hee Hwang and Lenhart K. Schubert, "Meeting the interlocking needs of LF-computation, deindexing and inference: An organic approach to natural language understanding," in Ruzena Bajcsy, editor, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 1297–1302, Chambery, France, 28 August–3 September 1993.

[Kamp, 1981] Hans Kamp, "A theory of truth and semantic representation," in J. Groenendijk, T. Janssen, and M. Stokhof, editors, *Formal Methods in the Study of Language*. Amsterdam Press, Amsterdam, 1981.

[Kautz and Ladkin, 1991] Henry A. Kautz and Peter B. Ladkin, "Integrating metric and qualitative temporal reasoning," in *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 241–246, Anaheim, CA, 12–19 July 1991. MIT Press.

[Koomen, 1989] Johannes A.G.M. Koomen, "Localizing temporal constraint propagation," in R.J. Brachman, H.J. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, pages 198–202, Toronto, Ont., 15–18 May 1989. Morgan Kaufmann.

[Light, 1993] Marc Light, "A computational theory of lexical relatedness," Technical Report 421, Department of Computer Science, University of Rochester, Rochester, NY, 1993.

[Litman and Allen, 1987] Diane Litman and James F. Allen, "A plan recognition model for subdialogues in conversation," *Cognitive Science*, 11:163–200, 1987.

[Magerman and Weir, 1992] D. Magerman and C. Weir, "Efficiency, robustness and accuracy in picky chart parsing," in *Proceedings of the Thirtieth Annual Meeting of the Association for Computational Linguistics (ACL92)*, pages 40–47, 1992.

[Martin, 1993] Nathaniel G. Martin, *Using Statistical Inference to Plan Under Uncertainty*, PhD thesis, Department of Computer Science, University of Rochester, Rochester, NY, 1993. To appear as a Technical Report.

[Martin *et al.*, 1990] Nathaniel G. Martin, James F. Allen, and Christopher M. Brown, "ARMTRAK: A domain for the unified study of natural language, planning, and active vision," Technical Report 324, Department of Computer Science, University of Rochester, Rochester, NY, January 1990.

[Nakatani and Hirschberg, 1993] C. Nakatani and J. Hirschberg, "A speech-first model for repair detection and correction," in *Proceedings of the Thirty-First Annual Meeting of the Association for Computational Linguistics (ACL93)*, 1993.

[Poesio, 1994] Massimo Poesio, *Discourse Interpretation and the Scope of Operators*, PhD thesis, Department of Computer Science, University of Rochester, Rochester, NY, June 1994. Available as Technical Report 518.

[Pollard and Sag, 1994] C. Pollard and I. Sag, *Head-Driven Phrase Structure Grammar*, University of Chicago Press, Chicago, IL, 1994.

[Ritchie *et al.*, 1992] G. D. Ritchie, G. Russell, A. Black, and S. Pulman, *Computaitonal Morphology*, MIT Press, Cambridge, MA, 1992.

[Schaeffer *et al.*, 1993] Stephanie A. Schaeffer, Chung Hee Hwang, J. de Hann, and Lenhart K. Schubert, "EPILOG: The computational system for Episodic Logic (User's Guide)," Technical report, Department of Computational Science, University of Alberta, Edmonton, Alberta, 1993.

[Traum, 1994] David R. Traum, *A Computational Theory of Grounding in Natural Language Conversation*, PhD thesis, Department of Computer Science, University of Rochester, Rochester, NY, July 1994. To appear as a Technical Report.

[Traum and Allen, 1994] David R. Traum and James F. Allen, "Discourse Obligations in Dialogue Processing," in *Proceedings of the Thirty-Second Annual Meeting of the Association for Computational Linguistics (ACL94)*, pages 1–8, 1994.

[Traum and Hinkelman, 1992] David R. Traum and Elizabeth A. Hinkelman, "Conversation Acts in Task-oriented Spoken Dialogue," *Computational Intelligence*, 8(3):575–599, 1992. Also available as University of Rochester TR 425.

[Yampratoom and Allen, 1993] Ed Yampratoom and James F. Allen, "Performance of temporal reasoning systems," *SIGART Bulletin*, 4(3), 1993. Also available as TRAINS Technical Note 93-1.

A TRAINS Technical Note Series

All TRAINS Technical Notes are issued by the Department of Computer Science, University of Rochester, Rochester, NY, 14627-0226.

Allen, James F. and Schubert, Lenhart K. The TRAINS Project. TRAINS Technical Note 91-1, May 1991.

Ferguson, George. Domain Plan Reasoning in TRAINS-90. TRAINS Technical Note 91-2, June 1991.

Light, Marc. Semantic Interpretation in TRAINS-90. TRAINS Technical Note 91-3, June 1991.

Martin, Nathaniel G. and Miller, Bradford W. The TRAINS-90 Simulator. TRAINS Technical Note 91-4, May 1991.

Traum, David. The Discourse Reasoner in TRAINS-90. TRAINS Technical Note 91-6, May 1991.

Gross, Derek, Allen, James F., and Traum, David R. The TRAINS-91 Dialogues. TRAINS Technical Note 92-1, June 1993.

Yampratoom, Ed and Allen, James F. Performance of Temporal Reasoning Systems. TRAINS Technical Note 93-1, May 1993.

Nakajima, Shin'ya and Allen, James F. A Study of Prosody and Discourse Strategies in Cooperative Dialogues. TRAINS Technical Note 93-2, September 1993.

Heeman, Peter A. and Allen, James F. Dialogue Transcription Tools. TRAINS Technical Note 94-1, August 1994.

Heeman, Peter A. and Allen, James F. The TRAINS-93 Dialogues. TRAINS Technical Note 94-2, 1994. To appear.

Allen, James F., Schubert, Lenhart K., Ferguson, George, Heeman, Peter, Hwang, Chung Hee, Kato, Tsuneaki, Light, Marc, Martin, Nathaniel G., Miller, Bradford W., Poesio, Massimo, and Traum, David R. The TRAINS Project: A case study in defining a conversational planning agent. TRAINS Technical Note 94-3, 1994.

Traum, David R. The TRAINS-93 Dialogue Manager. TRAINS technical note, 1994. To appear.

Ferguson, George. Domain Plan Reasoning in TRAINS-93. TRAINS technical note, 1994. To appear.

B Other TRAINS-related Publications

Allen, James F., Guez, S., Hinkleman, Elizabeth, Jackson, Keri, Kyburg, Alice, and Traum, David R. The Discourse System Project. Technical Report 317, Department of Computer Science, University of Rochester, Rochester, NY, 1989.

Allen, James F. and Miller, Bradford W. The Rhet system: A sequence of self-guided tutorials. Technical Report 325, Department of Computer Science, University of Rochester, Rochester, NY, July 1991.

Allen, James F. and Schubert, Lenhart K. Language and discourse in the TRAINS project. In Ortony, A., Slack, J., and Stock, O., editors, *Communication from an Artificial Intelligence Perspective*, pages 91–120. Springer-Verlag, Heidelberg, 1993.

Allen, James F., Schubert, Lenhart K., Ferguson, George, Heeman, Peter, Hwang, Chung Hee, Kato, Tsuneaki, Light, Marc, Martin, Nathaniel G., Miller, Bradford W., Poesio, Massimo, and Traum, David R. The TRAINS project: A case study in defining a conversational planning agent. *Journal of Experimental and Theoretical AI*, 1995. To appear.

Ferguson, George. Explicit representation of events, actions, and plans for assumption-based plan reasoning. Technical Report 428, Department of Computer Science, University of Rochester, Rochester, NY, June 1992.

Ferguson, George and Allen, James F. Cooperative plan reasoning for dialogue systems. In *Papers from the AAAI-93 Fall Symposium on Human-Computer Collaboration: Reconciling Theory, Synthesizing Practice, AAAI Technical Report FS-93-05*, Raleigh, NC, 22–24 October 1993.

Ferguson, George and Allen, James F. Generic plan recognition for dialogue systems. In *Proceedings of the ARPA Workshop on Human Language Technology*, Princeton, NJ, 21–23 March 1993.

Ferguson, George and Allen, James F. Arguing about plans: Plan representation and reasoning for mixed-initiative planning. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)*, Chicago, IL, 13–15 June 1994.

Heeman, Peter and Allen, James. Detecting and correcting speech repairs. In *Proceedings of the Thirty-Second Annual Meeting of the Association for Computational Linguistics (ACL94)*, pages 295–302, Las Cruces, NM, June 1994.

Heeman, Peter and Allen, James F. Tagging speech repairs. In *Proceedings of the ARPA Workshop on Human Language Technology*, Princeton, NJ, March 1994.

Heeman, Peter A. Speech actions and mental states in task-oriented dialogues. In *Working Notes of the AAAI Spring Symposium on Reasoning about Mental States: Formal Theories and Applications*, pages 68–73, Stanford, CA, March 1993.

Heeman, Peter A. Spoken dialogue understanding and local context. Technical Report 523, Department of Computer Science, University of Rochester, Rochester, NY, July 1994.

Hwang, Chung Hee and Schubert, Lenhart K. Episodic Logic: comprehensive, natural representation for language understanding. *Minds and Machines*, 3:381–419, 1993.

Hwang, Chung Hee and Schubert, Lenhart K. Interpreting tense, aspect and time adverbials: A compositional, unified approach. In *Proceedings of the First International Conference on Temporal Logic (ICTL94)*, pages 238–264, Bonn, Germany, 11–14 July 1994.

Martin, Nathaniel B. and Mitchell, Gregory J. A transportation domain simulation for debugging plans. In *Proceedings of the IEEE Dual Use Workshop*, 1994.

Martin, Nathaniel G. *Using Statistical Inference to Plan Under Uncertainty*. PhD thesis, Department of Computer Science, University of Rochester, Rochester, NY, 1993. To appear as a Technical Report.

Martin, Nathaniel G. and Allen, James F. A language for planning with statistics. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, 1991.

Martin, Nathaniel G. and Allen, James F. A decision theoretic planning assistant. In *Working Notes of the AAAI Spring Symposium on Decision-Theoretic Planning*, 1994.

Martin, Nathaniel G. and Allen, James F. Statistical probabilities for planning. Technical Report 474, Department of Computer Science, University of Rochester, 1994.

Martin, N.G., Allen, J.F., and Brown, C.M. ARMTRAK: A domain for the unified study of natural language, planning, and active vision. Technical Report 324, Department of Computer Science, University of Rochester, Rochester, NY, January 1990.

Miller, Bradford W. The Rhet programmer's guide (for Rhet version 17.9). Technical Report 363, Department of Computer Science, University of Rochester, Rochester, NY, December 1990.

Miller, Bradford W. The Rhetorical knowledge representation system reference manual (for Rhet version 17.9). Technical Report 326, Department of Computer Science, University of Rochester, Rochester, NY, November 1990.

Nakajima, Shin'ya and Allen, James F. Prosody as a cue for discourse structure. In *Proceedings of the Second International Conference on Spoken Language Processing (ICSLP-92)*, pages 425–428, October 1992.

Nakajima, Shin'ya and Allen, James F. A study of prosody and discourse strategies in cooperative dialogues. *Phonetica*, 50:197–210, 1993. Also available as TRAINS Technical Note 93-2.

Poesio, Massimo. Expectation-based recognition of discourse segmentation. In *Proceedings of the AAAI Fall Symposium on Discourse Structure*, Asilomar, CA, November 1991.

Poesio, Massimo. Scope ambiguity and inference. Technical Report 389, Department of Computer Science, University of Rochester, Rochester, NY, July 1991.

Poesio, Massimo. Conversational events and discourse state change: A preliminary report. In Nebel, Bernard, Rich, Charles, and Swartout, William, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR92)*, pages 369–380, Boston, MA, 25–29 October 1992. Morgan Kaufmann.

Poesio, Massimo. Assigning a semantic scope to operators. In *Proceedings of the Thirty-First Annual Meeting of the Association for Computational Linguistics (ACL93)*, Columbus, OH, June 1993.

Poesio, Massimo. Definite descriptions and the dynamics of mental states. In *Proceedings of the AAAI Spring Symposium on Mental States*, Stanford, CA, March 1993.

Poesio, Massimo. A situation-theoretic formalization of definite description interpretation in plan elaboration dialogues. In Aczel, P., Israel, D., Katagiri, Y., and Peters, S., editors, *Situation Theory and its Applications, vol.3*, chapter 12, pages 339–374. CSLI, Stanford, CA, 1993.

Poesio, Massimo. *Discourse Interpretation and the Scope of Operators.* PhD thesis, Department of Computer Science, University of Rochester, Rochester, NY, June 1994. Available as Technical Report 518.

Poesio, Massimo, Ferguson, George, Heeman, Peter, Hwang, Chung Hee, Traum, David R., Allen, James F., Martin, Nathaniel G., and Schubert, Lenhart K. Knowledge representation in the TRAINS system. In Ali, S., editor, *Working Notes of the AAAI Fall Symposium on Knowledge Representation in Implemented NLP Systems*, November 1994. To appear.

Traum, David R. Mental state in the TRAINS-92 dialogue manager. In *Working Notes AAAI Spring Symposium on Reasoning about Mental States: Formal Theories and Applications.*, pages 143–149, March 1993.

Traum, David R. Rhetorical relations, action and intentionality in conversation. In *Proceedings ACL SIG Workshop on Intentionality and Structure in Discourse Relations*, pages 132–135, June 1993.

Traum, David R. *A Computational Theory of Grounding in Natural Language Conversation.* PhD thesis, Department of Computer Science, University of Rochester, Rochester, NY, July 1994. To appear as a Technical Report.

Traum, David R. and Allen, James F. A speech acts approach to grounding in conversation. In *Proceedings of the Second International Conference on Spoken Language Processing (ICSLP-92)*, pages 137–140, October 1992.

Traum, David R. and Allen, James F. Discourse obligations in dialogue processing. In *Proceedings of the Thirty-Second Annual Meeting of the Association for Computational Linguistics (ACL94)*, pages 1–8, Las Cruces, NM, June 1994.

Traum, David R., Allen, James F., Ferguson, George, Heeman, Peter A., Hwang, Chung-Hee, Kato, Tsuneaki, Martin, Nathaniel, Poesio, Massimo, and Schubert, Lenhart K.

Integrating natural language understanding and plan reasoning in the TRAINS-93 conversation system. In *Working Notes of the AAAI Spring Symposium on Active NLP*, pages 63–67, Stanford, CA, 21–23 March 1994.

Traum, David R. and Hinkelman, Elizabeth A. Conversation acts in task-oriented spoken dialogue. *Computational Intelligence*, 8(3):575–599, 1992. Also available as University of Rochester TR 425.

Yampratoom, Ed. Using simulation-based projection to plan in an uncertain and temporally complex world. Technical Report 531, Department of Computer Science, University of Rochester, Rochester, NY, 1994.

Yampratoom, Ed and Allen, James F. Performance of temporal reasoning systems. *SIGART Bulletin*, 4(3), July 1993. Also available as TRAINS Technical Note 93-1.