The I3S Project: A Mixed, Behavioral and Semantic Approach to Discourse/Discourse Systems

Bradford W. Miller, Chung Hee Hwang, Yugyung Lee, Janet Roberts, Alexander I. Rudnicky

MCC Technical Report I3S-107-00

Microelectronics and Computer Technology Corporation (MCC)
3500 West Balcones Center Drive
Austin, TX 78759-5938

Abstract

The Intuitive Interfaces to Information Systems (I3S) project at MCC investigated novel approaches to simplifying the construction of spoken dialogue systems. Our goals included designing a system architecture that allows domain independent strategies, such as appropriate conversational gambits, to be separated from domain dependent strategies, such as the most effective prompt to accomplish the immediate task at hand. The system uses plan-based representations for dialogue and domain, and includes such components as a problem solver and plan recognizer. In addition, a new representation called Meta Problem Solving Actions that provides the rationale for problem solving behavior has been introduced to improve overall system behavior and coherence. Other important contributions include the development of a conceptual layer called Interaction Plans that relate Meta Problem Solving Actions to discourse phenomena. We have used these representations to develop an innovative interpretation strategy for user speech acts, using a combination of behavioral and semantic rules and representations to determine the most reasonable interpretation while maintaining real-time response. Our new representations lead to reduced application development and maintenance time. Prototypes have been implemented in an information service-based domain (City Resources).

Overview of Current Technology

As the amount and complexity of interaction between humans and computers increase, the role of the computer is becoming that of collaboration with humans. A key aspect of this role is the support of mixed-initiative interaction (Allen 1999). In order to properly support such interaction, as well as capture the rationale behind communicative and domain actions, an extensive and flexible framework is required. Common frameworks for dialogue systems built to date include graph-based, frame-based and plan-based. We will first look at the advantages and disadvantages of these systems to motivate our design decisions.

First generation spoken dialogue management systems typically involve the construction of conversational flow-charts linking possible dialogue states. Each state specifies a prompt, and enumerates possible user responses that transition the dialogue to a new state (e.g., see (McTear 1998)). A practical difficulty with this approach is that conversational context must be explicitly encoded in the conversational graph. Thus, graph-based systems are exponentially hard in the input domain—i.e., every possible state must be explicitly encoded.

Second generation approaches (for example, (Ward and Issar 1994)) substantially simplify this development work by making an assumption that possible interactions with the application can be expressed as a set of frames, e.g., see (Hayes and Reddy 1983), and that interaction in the domain can be driven by the filling out a frame. Given that the parser also outputs semantic frames, the application frame can be filled in with a few rules (e.g., what to do when the parsed frame specifies an already filled in slot) as well as rules for generating a prompt back to the user.

Frame-and-Rule systems have a number of advantages over the graph-based model. For one, it is relatively simple to allow for a kind of mixed-initiative interaction when the domain supports keyword or phrase-based parsing. A keyword in the input stream allows the parser to make a fairly good guess as to the corresponding semantic frame, and the deep parse is not required. Prompting is simply based on unfilled slots in the application semantic frame. This allows the following sort of interaction:

(S) Where would you like to go?

(U) I need to leave sometime after 4pm today.

(S) Leaving today after 4pm; where would you like to go?

Here the system prompt may have been generated on the basis of having a slot for arrival-city in the application semantic frame. If it is currently unfilled, the rule system may select it as the next thing to ask the user. The user, on the other hand, ignores the prompt and supplies a departure time. So long as the parser can recognize the keyword “leave” and realize that a time has been supplied, it can guess that a departure time has been specified regardless of the deeper meaning of the sentence. If this fills in another unfilled slot in the application frame, the system may enter...
it and continue, allowing itself to exhibit robust behavior, albeit a potentially unnatural one.

Frame-and-Rule systems can also handle sequential tasks (fill in a frame, use it, then fill in the next one), as well as multiple simultaneous tasks (fill in all applicable frames, the first to be filled completely is considered “the task” and evaluated or executed). The flexibility of Frame-and-Rule systems should not be underestimated, as the degree to which they can be used in an arbitrary domain is constrained only by the effort needed to capture the task into frames and then to write appropriate rules (Papineni, Roukos and Ward, 1999).

The limitations of a Frame-and-Rule system derive from the assumption that dialogue context can be mapped directly onto the application semantic frame. That is, all that we need to know about the interaction can be modeled by the degree to which we have filled out the frame, perhaps with some additional state about the current turn. Most frame-based systems also include a history list mechanism that allows for the resolution of simple anaphora. In general, this means that the input stream must have keywords to indicate the contribution of the current input if it does not directly follow from the previous prompt. This can give a dialogue system the appearance of having “attention deficit disorder”, in that it may appear to jump from topic to topic, rather than trying to fit a contribution in to some conversational structure signaled with discourse cues—it is simply filling in the slot(s) associated with the keywords. In short, Frame-and-Rule systems presuppose simple inputs, have a rudimentary dialogue model, and rely on domain-specific rules that are difficult to generalize. They are not capable of handling complex tasks that will require cooperation between the participants, or deal with a realistic range of natural language phenomena.

Another category of dialogue systems, namely, plan-based systems, are designed to address these shortcomings (e.g., (Allen et al., 1996)). There are some overt similarities between the plan- and frame-based approaches. We may consider the plan as a structure that can link together frames coherently, as well as allowing us to specify ‘frame generators’. That is, an abstract plan (e.g., alluding to the sort of plan one might have in ABSTRIPS (Sacerdoti 1974)) can be thought of as a way to specify all of the frames that decompose from the abstract plan specification.

There are other strengths to this approach; namely, we can explicitly model the domain actions intended by our conversational participants and the system itself as a rational agent within the domain, capable of performing actions on behalf of the user or to further the user’s understanding of the domain. A crucial advantage is in the system behavior when the user utters something that we cannot recognize as part of any reasonable or coherent plan (through plan recognition). We can assume that there has been a misrecognition or misunderstanding, and use this information to initiate an appropriate clarifying subdialogue back to the user. For these reasons, we chose a plan-based framework in building our dialogue system.

One essential difference between plan-based systems and Frame-and-Rule systems is that we explicitly model the dialogue itself, using a discourse model. Having a discourse model is important because it imposes structure between utterances. We presuppose that when someone speaks fluently, they intend their current utterance to fit in with their prior utterances, in a way that will be implicitly or explicitly signaled through their manner of speaking. For instance, one may use cue phrases or intonation to signal subject changes. Interaction with the user is based on the fit of the current input with the domain model, which is represented as plan and metaplanning (problem solving) actions (Ramshaw 1989).

Note, however, that our approach reported on in the rest of the paper is distinguished from those using “rational agency” (Sadek, Bretier and Panaget 1997) in that we do not have an explicit logical model of rational behavior and communication. Instead we allow the system behavior to arise from the interaction between verbal representations and the problem solver. We also distinguish our approach from that of (Smith and Hipp 1994) (Smith Hipp and Biermann 1995). While both approaches are concerned with task-oriented dialogue, we are concerned with collaboration between agents, as we assume that neither agent is capable of solving the problem or executing a plan on its own. We chose to focus on a class of problems where the system can be assumed to have considerable information about a domain, but is unaware of the user’s desires and goals within that domain, or where multiple strategies may be appropriate, as in information retrieval-related dialogues, e.g., (Stein Gulla and Thiel 1999) or (Hagen 1999). The communication between the agents, then, deals with describing a desired result and critiquing those proposed by the other agent, where a critique is generally interpreted as some portion of the goal or problem solving strategy that has not yet been communicated (or, at least, recognized).

The I3S System

We based our architecture on a plan-based representation of the domain, similar to the one in the TRAINS system (Allen et al. 1996)(Ferguson et al. 1996). While we have adopted the TRAINS framework, we have developed additional components to aid in the analysis of dialogue and domain. In particular, explicit plan recognition, domain reasoning, plan execution, an external ontology, and a new conceptual layer called Interaction Plans for mapping between dialogues, plans, and actions have been developed. An outline of the overall system architecture is seen in Figure 1. Note that while the parser is capable of parsing the results from gestured, spoken, and written inputs, only the speech (using CMU’s Sphinx II®) and written inputs were used for our prototypes.

The DM (Discourse Manager) consists of several parts. Reference deals primarily with grounding, i.e., determining the internal representation for mentioned objects, in conjunction with the Domain Resolver, which helps map
from descriptions to objects in the application domain. **Prince** maps input (surface) speech acts onto system actions based on analysis from its cohorts: the **Problem Solver (PS)**, **Plan Recognizer (PR)**, and **Plan Executive (Exec)**. **Domain Reasoners** aid the PS in formulating domain plans answering questions and comparing solutions. The **Output Planner** turns Prince’s intentional responses into appropriate speech acts for **Actualization**, which chooses the media and the expressive content to realize these actions.

**Figure 1**

We have augmented the general speech act interface of (Allen et al. 1996) with additional intensional components including *interaction pairs* between the PR, PS, and Exec components. We have also added a discourse interpretation stack (a stack of elements indicating the preferred discourse-related interpretations of the current contribution), and introduced a structure called the **system resolution state** (showing what stage in resolving the user’s speech act we are in—more on this later), information about the current discourse segment, and system plans for interpretation and response.

The Lymphocyte engine (Miller, 1996) forms the basic resolution component in Prince. It implements an anytime (Dean and Boddy 1988) subsumption architecture (Brooks 1986) (Brooks 1991) approach to pattern matching. This is used as the basis of behavioral disposition of the input, and also provides substantial robustness to unexpected or malformed input. It provides both a contrast and a complement to the semantic components of the system because we use them as complementary components while subsumption architecture, as originally conceived, explicitly eschews plans and planning.

The coordination between the dialogue plan and the actual domain interpretation (generally represented as problem solving actions) is accomplished by a special binding layer, or **Interaction Plan (IP)**, following (Miller 98). In particular, interpretation choices that change the current topic and introduce a new sub-plan can be overridden by an interpretation that does not do so, when the latter might have otherwise been discarded (perhaps from the lack of semantic fit to the frame). In other words, we can use the IP to encode the amount of inertia the system should have in sticking to a topic by controlling the transitions between IPs, treated as **pragmatic frames**. In some cases it might require very explicit discourse markers to switch topics, and in others it may decide that a topic is closed, based on the realization from the recognized user IP, that the user is probably done with it, etc.

Because we can choose which subsumption rulebase to use to match within a pragmat**ic frame**, we can easily set up expectations for handling the next input. For instance, given the query “Which is better?” after the system reported on a list of possible problem solutions, the PS could indicate, with an issue\(^2\), that it has no Domain Reasoner that can respond to the notion of “better” between the plan instances. Alternatively, the PS may give the list of comparatives that it could handle between the plans, and indicate how to formulate a refined query to itself with the additional information. Prince can then use this expectation to generate an appropriate prompt for the user (or for analysis with the plan recognizer). Should the next input matches the expectation type, this expectation can be used to handle the response immediately, rather than forcing a more general resolution (which would take place if the input did not match the expectation). Because this expectation system generalizes to encapsulate any sort of resolution the system may do, it can be used to “think on the opponents time” and pre-generate possible dialogue continuations based on the current state. This allows us to provide better performance. In addition, we found that expectations are difficult to represent in a logical rules-based approach, but have a clean translation into rules based on subsumption, such as with Lymphocyte. We will further discuss resolution in a later section.

By designing the system in this fashion, we have constructed a very flexible and portable system that can adapt to new domains easily. Some control of the interaction can be suggested by one of the subordinate component modules using the expectation mechanism described above. This allows for particular viewpoints from domain dependent analysis to be realized in system behavior as needed. Since the cohort modules are coordinated by a single component, Prince, it is still possible to moderate user interaction based on additional issues to allow for a more coherent, fluent exchange. These issues include the discourse/dialogue state (which may include the current discourse segment) and the current dialogue plan (of which the current discourse segment is a step). A particular dialogue plan, for instance, may act as a template, telling us the appropriate speech acts to be used for planning the system’s output. The Output Planner, then, takes the relevant information supplied by the cohort modules, and incorporates it into the discourse segment template for output. The lack of appropriate information would be an indication that the current discourse plan is not appropriate at this point and an alternative must be selected.

One important feature of our system is the use of exter-

\(^2\) **Issues** are abstract data structures containing such information as ‘insufficient information to find any possible plan’, or ‘ambiguous situations that would generate so many possible plans that exploration would be difficult’.
nal ontologies that serve as a means for establishing a conceptually concise basis for communicating knowledge for various purposes such as natural language, discourse context, etc. Ontology is also used to divide generic and domain specific knowledge. This separation provides a framework for organizing components of the system and modeling the application domain. Since the knowledge owned by the system can be organized, retrieved, and reasoned about in both a generic and a domain-specific manner, the distinction provides the application engineers a working space—indeed of the properties of the underlying knowledge representation—where they can model the domain-specific knowledge and behavior for a new application. This allows rapid prototyping and incremental updates of the application.

The planning and plan recognition process we have modeled in our system is specifically to help achieve the information seeking goals of the user. The idea is that the coordinated interaction between the system and the user can be used to achieve their mutual goals based on common understanding and grounding (Horvitz 1999). In our system, the recognizable IPs are represented in the form of communication and problem solving plans, and the semantic and behavioral aspects of interaction between users and the system can be specified using them. The rationale motivating particular moves in the system and user interaction space, as represented with IPs, is used to determine which plans are relevant to a particular situation. These include setting up the interaction (greetings), establishing a goal (meta-task), evaluating a particular proposed solution (task), etc.

Additionally, since plan recognition is essentially a process for integrating observed actions into a coherent “recognizable” plan, as in (Pollack, 1990), nothing prevents us from using an identical mechanism for integrating identified discourse actions (of the user) and/or discourse segment purposes into IPs. That is, by virtue of having a generalized plan recognition mechanism in our system, we already have the capability for recognizing the user’s IPs, provided we supply it with appropriate IPs to reason over, and use as input recognized discourse actions (such as questioning a presupposition, making fun of a prior system suggestion, etc.) in place of domain actions. This can then be used to predict the discourse segment purpose for a new speech act, even when the illocutionary force has not been completely identified (e.g., due to misrecognition).

The role of a Domain Reasoner is to receive a constrained goal from the Problem Solver, and to generate in response a grounded plan solution with a set of actions generated using domain knowledge. Plan generation includes three possible result scenarios: a set of solutions, a single solution with an indication that more are possible, or an issue. Depending on the situation, a Domain Reasoner may propose a possible behavior in the form of an issue to the PS, which encapsulates, e.g. a prompt for appropriate disambiguating constraint.

It is usually nontrivial to select the most effective constraint, particularly in a domain involving a large number of possible constraints. Our desire is to pick a constraint that will lead to the most efficient selection behavior. For instance, it’s probably a poor choice, in a set of 200 restaurants that fit the current goal, to ask that the user tell us the credit card they plan to use; the part of town or type of food is more likely to constrain the list into manageable pieces than something where only a “lucky” answer will help us. For this purpose, we have developed an explicit domain model, based on statistical knowledge about the domain. Constructing such a model is automatic—we probe for each domain object known by the domain reasoner, and the values of all the applicable relations. This is then organized into an information model that can tell us, given some set of known relations in our goal, which of the remaining relations convey the most information. Statistical knowledge also allows estimating the similarities between domain role fillers. For instance, by knowing a user likes rock music, a Domain Reasoner can speculate they might also like blues. Based on this information, the system can suggest alternative constraints when a goal is over-specified. This domain model can be very effective for a mixed-initiative system where a design goal is to minimize the number of user interactions.

**Example Session**

The following transcript has been taken directly from a session with the system. We will walk through this dialogue below.

U01 *How are you?*
S fine.
U02 *Fine.*
S how are you, slaymore fluglenose?
U03 *I’m looking for something to do tonight.*
S your choices are eating, having a drink, and listening to music. which one are you interested in?
U04 *I’d like barbecue.*
S the venues are slack’s bbq, joe’s bar&grill, and hogs. which one are you interested in?
U05 *Which is better?*
S hogs is better.
U06 *How about Joe’s Bar & Grill?*
S joe’s bar&grill is worst of those.
U07 *Nevermind.*
S that’s ok.
U08 *What’s their address?*
S hogs’ address is 801 red river.
U09 *Phone number?*
S hogs’ phone number is 480-8341.
U10 *What about eating at a jazz joint?*
S yeah then try slack’s bbq instead of hogs.
U11 *What about Joe’s Bar & Grill?*  
S joe’s bar&grill is a fine choice also.
U12 *Great.*
S anything else i can help you with?
U13 *Who’s playing there now?*
S the band is tray slick and the slackers.
U14 Great talking to you!
S thanks.
S nice to meet you!
U15 You bet!

The system starts out in an IP to handle any greetings with a two-way greeting (i.e., any greeting initiated should be followed by a reply greeting). As the user initiated the first greeting, the system replies and returns a greeting in turn. This then matches the return of the greeting from the user (U02). Since the user fails to move on, the system moves to a new IP to attempt to elicit a goal from the user, which motivates the response to (U02). When the user gives only a very general goal in (U03), constraining the temporal aspect for execution (tonight), the system responds by generally listing the actions that it can assist the user in “doing”. Note that these actions are verified by subsumption to be user goals (though this isn’t particularly interesting in this case).

In (U04) the user mentions a kind of food; plan recognition suggests that this can be incorporated if the goal is the action Eat, so the current goal is now Eat barbecue tonight. This is sufficiently detailed for the planner to suggest three possible solutions, and because the alternatives are limited, the system decides it is reasonable to simply list them, rather than attempt to elicit additional constraints (as it did when “do something tonight” in (U03) was insufficiently detailed). The IP has also changed to one which presumes that a sufficient goal has been garnered, and we now attempt to select between possible plan solutions, represented by the locations listed. In (U05), the user tries to get the system to order the solutions by the comparative operator “better”. The system responded with the best known location in the set (based on a review database).

The next query by the user, (U06), is about the specific ordering within that set, and it is taken as a question about its position in the currently focused ordering relation. Had the venue not been part of the relation, it would have elicited the reason it was not being considered part of the solution (e.g., perhaps they were closed tonight, or did not serve barbecue). “Nevermind” is considered to shift the focus back from Joe’s Bar & Grill to Hogs. As we can see from the system’s response to (U08), the antecedent for the pronoun “their” was correctly identified. (U09) shows a simple elliptical sentence being handled. At this point the IP is set to answer queries about the solution, so (U10) is considered non-fluent, and the system changes the IP back to the meta-task level (where goals are discovered). (U10) is interpreted as a refinement of the goal, where we now add the additional constraint that the venue must have a jazz band (playing tonight). In this case Hogs is no longer considered, and Slack’s BBQ becomes the best of the three. Because the system is unable to determine a proper order between Slack’s BBQ and Joe’s Bar & Grill, the system allows that Joe’s is a reasonable alternative. And while the system failed to see the need to implicitly confirm the shift of focus (likely a bug), in the response to (U13), the band is the one playing at Joe’s.

Discussion

The various declarative elements of an I3S application fit together hierarchically in an intuitive way. From an application developer’s perspective, the IP serves as the strongest form of expectation, compartmentalizing the application according to the general kinds of things a user (and system) might do within a particular pragmatic context. Each IP provides a separate logical context within which to interpret the user’s utterances. The conversation plan (CP) serves as the strongest form of output planning, providing the mechanism for ordering speech acts between multiple conversational participants. Space limitations prevent us from describing our Output Planner, but see (The I3S System Documentation, 1999). Because each IP has a set of associated CPs that describe conversation “snippets”, each such CP can define separately how the snippet should be handled. For example, the CP to handle “great” in the task situation (say, exploring choices for a movie) would be different from the CP to handle “great” in a social situation (say, replying to “How are you?”). And since they are in separate IPs, they are not in conflict with each other. Thus the IP/CP constructs provide a highly scalable and maintainable architecture. Finally, similar to a CP describing the action sequences (as discourse segments) that are possible for a particular kind of conversation, a PSMP (problem-solving meta-plan) describes the problem solving action (PSA) sequences that are possible for a particular kind of problem solving response, thus encapsulating the information necessary to communicate the appropriate problem solving actions (PSAs) to the Problem Solver (PS) module in Figure 1, and to receive responses. I.e., the PSMP is the “data middleman” between Prince (on behalf of the CP that Prince is following) and the PS. Tying these notions together, a PSMP is designed by the application developer to: (1) unify information from the CP into the actions of the PSMP (where the actions represent the system’s problem solving response to the user’s input, such as create a plan, revise a plan, suggest a solution, etc.), (2) receive results (such as solutions, responses to questions, issues, etc.) from the GPS’ execution of those actions, and (3) unify the results back into the CP, thus making them available for output planning.

IP/CP Resolution

Resolution begins with Prince first checking if it can resolve the discourse expectation (if there is one). If so, expectation resolution produces both the output speech acts and the next IP, and we’re done for this turn. If not, then we try to continue resolution using the current CP. This is simply a “short cut” of the full IP/CP resolution mechanism which skips the “find CP” task. Otherwise, we use the full IP/CP resolution strategy, which tries to resolve the current input in the context of the current IP, typically by finding a suitable CP, and using it to produce the output and segue to the next IP. However, if a suitable CP in the current IP is not found, the system will next try to find a suitable rule in the IP’s rulebase. In the current implemen-
tion, the IP rulebases typically look for patterns based on the input speech act type combined with certain key values (e.g., the speech act semantics). This rule will either resolve the input itself, suggest another IP to try, or invoke the panic rulebase (clearly a last resort). If an IP rule suggests another IP to try, we move to that IP and re-resolve the input using this new IP, invoking panic in case of loops.

Alternatively, if at the start of the IP/CP resolution cycle the IP itself is “unknown” (i.e., the system realizes that it is “lost”, which could happen if it gets to the point where it has no more alternatives, or if it detects that it is looping), the system immediately resorts to the panic rulebase (which is accessible from all IPs), producing its best guess at an appropriate response.

General IP/CP resolution, in which Prince brings to bear all its resources, invoking the full power of its cohorts, involves finding a suitable CP, executing the PSAs of its PSMP, producing the output speech acts to actualize for the next turn, and identifying the IP for the next turn. Finding a suitable CP is an iterative process that continues (and may involve undoing problem solving actions before looking for alternative CPs) until a CP is found whose PSMP’s problem solving actions (PSAs) and prematch and postmatch conditions execute to successful completion.

When the IP/CP resolution mechanism fails, the system’s “fall-back” mechanism is to use a subsumption approach for interpreting the user’s speech acts. The processing of speech acts is divided along two broad lines: discourse-based acts (e.g., greeting, confirmation), and domain-based acts (e.g., goal identification, query). Prince will typically resolve a discourse act using information in the discourse context, without the help of the Plan Recognition and Problem Solver modules. However, for domain-based acts, it relies heavily on the interpretations given to it by PR and PS, and it uses the Plan Executive to perform the plan actions of grounded (or partially-grounded) plans. In particular, Prince uses Lymphocyte rules to determine which of these two modules it should consult, based on the roles of those modules. Although the actual rules are largely application-dependent, the strategies apply generically to applications in a task-based domain.

In developing our information service-based reference application, City Resources, the combination of these resolution mechanisms provided a natural means for modeling the application from a human-human (vs. human-computer) perspective: people typically do try to “plan” their conversational responses, navigating among interaction strategies as appropriate. Presuming that conversational participants are engaged for the purposes of exchanging information (vs. lecturing), the participants are (whether consciously or not) mapping the general notion of a plan composed of a sequence of actions onto specific discourse acts (encapsulated in CPs), and problem solving actions (encapsulated in PSMPs).

We illustrate by examining (U05/S, U06/S) from the Example Section. Let’s begin with a CP which happens to encompass exactly one user/system turn, against which we match the input and from which we produce the output:

```
cp-tc-question-comparative-with-solution
psmp-tc-direct-revise-solution-preference
agent-accept-psa ; plan-id is its action
agent-revise-psa ; comparative-object is its purpose,
and solution-set-id is its action
```

When this PSMP is chosen as a candidate, the PSMP instance contains values for all its properties (plan, comparative object, and solution set), has checked that any prematch conditions succeed, and has checked that the problem solving actions have the values they need. Executing the PSMP (as we shall see shortly) asserts the values from the result of executing the PSAs (i.e., from the GPS result) into properties of the PSMP. These result properties will then be unified into the CP.

The final step before output planning is to execute the problem solving actions corresponding to Prince’s understanding of what the user has said. Taking one candidate PSMP, Prince will attempt to execute its PSAs to completion. If successful, the PSMP postmatch conditions are evaluated. Typical postmatch conditions might include checking the PS “answer score” and asserting the PS results into properties of the PSMP (this is how the results are “saved”). If the conditions resolve, the values they assert into the properties of the PSMP are unified into the CP for later use by the output planner.

Otherwise, the PSMP executor uses the list of remaining possible CPs to find alternatives with the same already-executed PSAs and valid pre/post conditions. If that fails, the PSMP executor might ask PR for help, and that might involve doing a roll-back (undo) by the PS of the already-executed PSAs. If that fails, the PSMP executor would try other CPs, maybe in other IPs. The information for this back-chaining comes from two sources: (1) the original alternatives resulting from finding candidate CPs, complemented with (2) the PSMP-to-CP correspondence table of the IP (designed by the application developer).

Continuing the example for “Which is better?”—the new solution set and solution object are unified into the cp-tc-question-comparative-with-solution CP. The solution property is not used by this CP; it is, however used by other CPs (that use this PSMP) that need to compare if the current solution is the same as or different from the prev-

---

3 In fact it would not be, since a better choice shall shortly be introduced.
ous solution (so that the output planner could say “I still suggest Hogs”, or “Joe’s Bar&Grill is a fine choice also.”).

Finally, the output planning task uses the CP/PSMP to generate the system’s response. Given a pointer to the discourse segment to user, the output planner selects (and then instantiates) the speech acts of that discourse segment that it decides are most appropriate.

Returning to the example “Which is better?” can, one possible output discourse segment (DS) for realizing the response “Hogs is better”) is:

ds-tc-pref-elaborate
sa-pref-elaborate

This discourse segment has properties for the comparative object and solution object, both of which are asserted from the CP. Then, when this discourse segment is evaluated for suitability, the values of those properties are asserted into the output speech act.

Adding (U06/S) to this example, the following (more elaborate) CP becomes a candidate:

cp-tc-comparative-suggest-solution
ds-tc-question-comparative (U: Which is better?)
ds-tc-pref-elaborate (S: Hogs is better.)
ds-tc-suggest-solution (U: How about Joe’s Bar&Grill?)

sa-comparative-pref-elaborate (S: Joe’s Bar&Grill is worst of those.)

This time there is a prefix, namely the two discourse segments representing “Which is better. Hogs is better.” Thus, the CP instance presented to the output planner will direct it to produce output using the discourse segment sa-comparative-pref-elaborate. This illustrates an important point about CPs: they are serve to guide Prince’s interpretation, but they are not an end in themselves; that is, they are “useful for as long as they are useful”. The current CP is no longer relevant as soon as it doesn’t match anymore.

Recognizing User Goals

In addition to an extensive framework which can model the interactions between users and system, a powerful mechanism to recognize user’s intentions and goals from the perspective of interaction (communication and problem solving behaviors) is needed. In order to generate a set of appropriate actions driving the system’s behavior, the system relies on the ability to recognize a user’s goals by considering evidence obtained through the partial observation of the user’s behavior up until the current time, as in (Kautz 1986), where the simplicity heuristic is used to propose an explanation with the fewest goals. Like (Allen 1999), we build plan graphs to infer plans during cooperative interaction, but use mechanisms related to approaches based on terminological or linguistic reasoning, such as (Vilain 1990), (Eugenio et al. 1992), and (Weida et al. 1992) to choose plan templates based on observed linguistic phenomena.

We describe the context in which the plan is generated and how it can be exploited in the recognition process in terms of evidence from conversation and problem solving behavior and the interaction framework. We use a plan recognizer based on semantic nets for analysis of the user’s conversational and domain behavior.

The plan recognizer can be thought of as an internal user; it continuously keeps track of the user’s goals and their interest in receiving active assistance. The role of the Plan Recognition model is to recognize the user’s goal based on knowledge from the current input, the system’s intensional state (system’s accumulated behavior and interpretations), the user’s problem solving and planning behavior, and the user’s express preferences and focus. The recognition model is built based on the following primary components. First, a semantic net is used to combine multiple relation types such as is-a, part-of, evoke, associated-with, etc. from the ontology and efficient heuristics to reason over this semantic net for the likelihood that two terms are substitutable. The heuristics are computed according to the relative distances between concepts in the semantic net, the type of relations which are involved in a inference path between the concepts, and the degree of the association of the relationship. Second, case-based recognition based on the input stream is used in the front end to avoid unnecessary exhaustive search. The particular input pattern cases are defined in the form of rules and the case is selected using the Lymphocyte engine. Third, the user model within plan recognition is used to keep track of user’s focus of attention and preferences and to extract useful information whenever needed. Last, a plan graph, which specifies the user’s problem solving progress, is used to propose the most appropriate problem solving plan according to plan inference based on the current problem state and coherent possible plans in the current IP.

Our approach can be differentiated in several aspects from existing research on plan recognition. Scalability is critical for us since we need to recognize a user’s goal and focus of attention with a real-time response. We recognize a user’s goal based on integrated knowledge from multiple sources such as a user model, the user’s problem solving state, the user’s conversational state, the system’s intensional state, and mentioned concepts derived from ontological distance measures. These ontologically based representations allow us to represent connections between concepts, such as ‘evoke’ which are typical of daily activity in a domain. For instance, if user said “I am hungry”, the system proposes an eating plan: “what kind of food would you like?”. In fact, there are several connections with goals such as eating, drinking, etc. The application designer can assign stronger associations between concepts like eating and hungry rather than eating and drink. Our plan recognition model handles incomplete and partial information by providing the most plausible plans, generating multiple candidate matches and ranking them by some confidence metric. Depending on the confidence score, the system may decide to either pursue the proposed plans from the Plan Recognizer or enter a clarificational subdialogue to elicit needed information directly from the user. Finally, as the plan recognizer discovers the user’s focus of attention and preferences, updating the likelihood of a goal or action by learning the user’s preferences and interaction behavior.
Even though our plan recognition model doesn’t deal with complex mental states nor support sophisticated symbolic recognition processes, it does support dialogue containing ambiguous and partial information by providing incremental and interleaved plan recognition based on integrated knowledge from many different sources.

**Concluding Remarks**

The I3S approach provides a set of abstractions which allow a developer to model discourse phenomena in a natural way, enabling a smoother transition from data collection to application design, leading to reduced application development and maintenance times. We had two target domains, City Resources, which was used for the examples in this paper, and Financial Assistant, a banking/brokerage domain which was only partially implemented. We found that our approach was borne out in the ease in which these domains could be modified for new behavior.

A key contribution of the I3S approach is its resolution strategy, through which the system interprets the input speech acts to produce the output speech act, and determines the context for the next turn. An I3S application is represented by declarative information in the form of interaction plans and conversation plans, problem solving meta-plans and problem solving actions. Declarative pattern-based Lymphocyte rules tie together the resolution steps into a subsumption architecture. The strength of this approach is its robustness and elasticity in the face of both changes in the user’s intentions and system misinterpretation of user’s intention (due to misrecognition, incomplete information, etc.). Furthermore, not only can the system react properly in both cases, but it doesn’t really need to know which case it is. Thus, the system can afford to choose an interpretation and “stand corrected” based on additional dialogue as we naturally do in human-human communication.

**References**


The I3S System Documentation, 1999. MCC, Austin TX.


Miller, B.W. 1998. Interaction Plans and Their Recognition, Interactive and Collaborative Planning Workshop, in conjunction with the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98), CMU.


