ORGANIZING DIALOGUE FROM AN INCOHERENT STREAM OF GOALS*

ELISE H. TURNER

Department of Computer Science
University of New Hampshire
Durham, NH, 03824
USA

Abstract—Human discourse appears coherent when it reflects coherent human thought. However, computers do not necessarily store or process information in the same way that people do and, therefore, cannot rely on the structure of their dialogues. Instead, computer-generated conversation must rely on some other mechanism for its organisation. In this paper, we discuss one such mechanism. We describe a *template* that provides a guide for conversation. The template is built from schemata representing discourse convention. As goals arrive from the problem solver they are added to the template. Because accepted discourse structures are used to connect a new goal to the existing template, goals are organised into sub-groups that follow conventional, coherent patterns of discourse. We present JUDIS, an interface to a distributed problem solver that uses this approach to organise dialogues from an incoherent stream of goals.

I Introduction

Conversation seems coherent and is easy to follow because it reflects the way people think. When the speaker thinks coherently, his or her communication goals will be properly organized to follow linguistic convention. So, models of human language generation can allow domain goals to directly motivate conversation and add clue words only when the occasional utterance violates convention [5; 8].

However, computer-generated conversation cannot rely on problem solving for its organization. Some problem solvers make no attempt to be "cognitively plausible" and do not produce goals in sequences that would appear coherent to human users. The combined goals from a distributed problem solver where several independent reasoners use a single interface to communicate with the user are also likely to be incoherent. Even if individual problem solvers produce coherent streams of goals, the stream of goals from the aggregate is likely to switch back and forth between sub-problems that are being addressed by the individual systems. We call the sequence of goals produced by such systems an *incoherent stream of goals* because the goals are ordered in a way that would not seem reasonable to a human listener. Interfaces to such systems, while being responsive to the goals of problem solving, must rely on something else to give dialogue its organisation.

In this paper, we describe a *template* that can provide computer-generated conversation with a coherent organisation while meeting the needs of the problem solvers. This template is built from schemata representing expected discourse structure. These schemata include general linguistic conventions as well as expected organisations for specific situations. Before conversation begins the template is very abstract, giving only broad descriptions of topics that might be expected in a specific domain. These expectations provide a framework for organising goals from the problem solvers. New goals are associated with the existing template, organising them into groups that are related by conventional discourse structure. Goals are added by finding schemata which connect them to the template. Since the schemata specify acceptable conversational structure, the template represents a coherent conversation.

The dialogue in Figure 1 was organised using such a template. The arrows in the figure show the order in which goals arrived at the interface. The tail of the arrow shows when a goal arrived at the interface; the head, where it is realised in the dialogue. Each time the arrows cross, the goal has been delayed to fit more naturally into the conversation. Goals have been grouped by the course that they relate to as well as being further organised by attributes such as ingredients of the main-dish of a course (utterances 10a-10d). The dialogue also includes a story to connect two utterances (10a and 10b), which adds variety to the structure of the conversation.

This dialogue was generated by JUDIS [15], an interface to Julia, an advisor for meal-planning [3].[1] Ju-

*Some of the work described in this paper was done at the Computer Science Department of Georgia Institute of Technology, Atlanta, Georgia, 30332, supported by the NSF under grant IST-8608362. The author wishes to thank the anonymous reviewers for their comments.

[1] JUDIS is responsible for the organising the conversation and works with concepts instead of actual English utterances. Concepts are converted to English and English translated into the conceptual language by the NLPTool [2]. JUDIS has worked with the NLPTool to participate in English dialogues, but was not connected to the NLPTool for this example. In the figure, English is used to make the dialogue readable.
The Julia described here is the original version of the system.

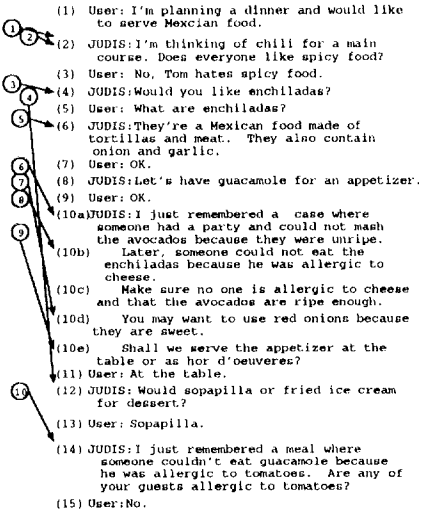

Figure 1: A dialogue with JUDIS.

lia is a distributed problem solver comprised of a case-based reasoner and a planner which uses more traditional problem solving techniques. The problem solvers and the interface share their world knowledge. When a problem solver has a goal to either give or get information, the goal is sent to JUDIS where it is converted into an appropriate utterance and attached to the template. We refer to these goal-motivated utterances as *requests*.

II Representing Discourse Structure

We have chosen *conversation MOPS (C-MOPS)* [6; 16] as the representation for discourse structure in JUDIS. C-MOPS participate in an abstraction hierarchy which allows generalised conventions as well as situation-specific expectations to be represented. A *dynamic memory* [7; 14] can retrieve the most predictive MOP for the current situation.

A MOPs and C-MOPs

A *memory organization packet (MOP)* [14] is a schematic structure used to organise long-term, conceptual, episodic memory. An *episode* is represented by *scenes* which have been performed to achieve some goal. Episodes are stored in and retrieved from dynamic memory. This memory changes when *generalized episodes* are created as individual episodes that share features are stored in memory. The generalisations occur at many different levels forming a hierarchy of generalisations and their *specializations*. Episodes in dynamic memory are linked by *predictive indices*, selected feature-value pairs which mark differences between generalised episodes and their contributing specialisations. These indices are followed when an episode is *retrieved* from dynamic memory, allowing a system to be "reminded" of MOPs which match some predictive feature of the current situation. We use the term "MOP" to describe both a single episode and an episode with its indices and specialisations. In the context of being retrieved from memory or instantiated in the template, "MOP" will refer to a single episode. In the context of representations stored in memory, "MOP" includes the indices and specialisations.

When the events stored are conversations, we refer to these structures as *conversation MOPs* or C-MOPs. Kellermann, *et al.* [6] suggest C-MOPs as the cognitive structures for representing discourse structure. C-MOPs can appear as scenes in other C-MOPs, allowing for the recursion necessary in any representation of discourse structure. The scenes of a C-MOP can be given a total or partial ordering to capture the proper sequencing of a conversation. Also, C-MOPs combine *intention*, in the form of an associated goal, with *convention* captured by generalised episodes. Kellermann *et al.*'s experiments suggested that C-MOPs representing discourse structure are divided into scenes by topic.

In many ways C-MOPs are like other schemata that capture discourse structure [eg., 9; 10]. Their scenes specify conventional patterns of discourse. These scenes can be either *mandatory* or *optional*. Many types of schemata can be easily translated into a declarative representation that explicitly gives the structure of the conversation and, so, are suitable for building the template. All types of schemata must allow recursion, so a template built from any type of schemata could be expanded. C-MOPs have one characteristic, however, that makes them particularly useful for organising requests. They participate in a generalisation/specialisation hierarchy. This hierarchy has two advantages: it allows the best prediction for a given situation to be returned from memory, and it allows those predictions to be tuned as new information is learned about the situation.

B The Generalization/Specialization Hierarchy

The ability to capture convention in the generalisation/specialisation hierarchy is important for our work in organising dialogues. In principle, generalisations

As research on problem solving continued, "Julia" was also used to refer only to the problem solvers and the design of the problem solvers changed. JUDIS receives goals from a micro-version of the caterer and as input from the keyboard. The goals in the example came as input and reflect the problem solving of systems suggested for the caterer architecture. At the time that JUDIS was implemented, a complete and integrated version of the caterer was not available.

are formed as a language user participates in conversation with other language users [7; 14]. Because these conversations follow convention, the generalisations of these conversations will represent abstract discourse convention [6]. When specific circumstances constrain these conventions, specialisations are formed and indexed by these circumstances. Consequently, the C-MOP retrieved for a given situation will be the one that is most predictive.[2] The expectations it represents will be shared by other language users, including the other conversant, and will contain all the information available for the specific situation. Since our research is currently focused on how knowledge of discourse structure can be used to organize goals, instead of on elucidating those structures, JUDIS' C-MOPs and the generalization/specialization hierarchy are hand-coded. Our C-MOPs are derived from others' research on discourse structure, where possible. Although JUDIS does not generalise C-MOPs from experience, we have been careful to use a generalisation/specialisation hierarchy which we believe could have been built from experience.

One important characteristic of the generalisation/specialisation hierarchy is its ability to capture situation specific detail. This ability is especially important given that computers do not think like people. Specialisations can be used to enumerate the acceptable ways to discuss a topic. The interface can then rely on the appropriate C-MOP to organise the conversation instead of being dependent on the knowledge organization and problem solving methods of the domain reasoners. The specialization can also rule out ways of organising the dialogue that follow a standard discourse convention but would not be expected by human reasoners. For example, a general problem-solving convention allows for a goal-subgoal ordering [5]. However, in meal-planning some orderings seem more acceptable than others. In JUDIS, specializations for discussing a meal include talking about the main course before the other courses or discussing the meal in chronological order, but a specialization for discussing the dessert first is not included.[3]

The generalisation/specialisation hierarchy also allows the template to be tuned as the situation changes or new information is discovered. If the new information is an index of a C-MOP, that C-MOP can be replaced with the indexed specialisation. This idea is important for adding new requests to the template. We can think of some cases of adding a request as finding a specialisation that includes the current request as well as the request that has just arrived. For example, request 6, about avocados, is added to the template as a discussion of the ingredients in guacamole. When request 7, concerning onions, arrives, the discussion of the ingredients is specialised to a list that includes both avocados and onions.

C *Conversation MOPs in JUDIS*

JUDIS relies on C-MOPs to represent all parts of the conversation. This includes C-MOPs for the entire conversation, individual topics, utterances, and question/answer sequences. The C-MOPs that are most important for organising conversation in JUDIS are the LIST-CMOP and NARRATIVE-CMOP which can be used to organize topics and the TOPIC-CMOP itself. JUDIS also has a CATERER-CMOP which contains specialised knowledge about conversations for planning meals and organizes the overall discussion of the meal.

Some C-MOPs, such as the CATERER-CMOP, are represented declaratively in memory with topics and their ordering given explicitly. This makes instantiating the C-MOP easy and allows the interface to be independent of problem solving knowledge when organising the dialogue. However, not all C-MOPs should be represented this way. JUDIS also represents some C-MOPs, such as the LIST-CMOP and the NARRATIVE-CMOP, procedurally. Explicit representations are created from other knowledge only when such a C-MOP is instantiated. This allows JUDIS to create conversations that have not yet been experienced but follow conventions that have been generalised from experience. It also saves space because JUDIS builds these C-MOPs from world knowledge that is shared with the problem solvers and does not have to explicitly represent all possible C-MOPs.

The CATERER-CMOP organises the discussion of the meal. JUDIS has very little information about the details of the conversation, but is able to identify the broad topics that are likely to be discussed: GENERAL-INFO, APPETIZER, MAIN-COURSE and DESSERT. The specialisations are indexed by specific problem solving strategies, main-first and chronological-order, that impose acceptable orderings on the topics.

The TOPIC-CMOP has three scenes: CHANGE-TOPIC, DISCUSSION, and CLOSE-TOPIC. The change-topic and close-topic are used to mark unexpected moves in the conversation and do not affect how we organise requests in the template. The discussion scene can be a TOPIC-CMOP, an UTTERANCE-CMOP, or a QUESTION/ANSWER-CMOP. The *subject* of the TOPIC- or DISCUSSION-CMOP tells what the C-MOP will be about. We use this term to avoid confusing the topic of a C-MOP with the TOPIC-CMOP.

The NARRATIVE-CMOP is a simplified version of Rumelhart's [13] story grammar and specifies how to build C-MOPs directly from MOPs in episodic memory.

[2] JUDIS' retrieval algorithm [15] orders the features and pursues the indices sequentially until an index is not found. This is a departure from traditional implementations of dynamic memory that pursue indices "in parallel" [cf., 7]. Our algorithm allows JUDIS' memory to return the single C-MOP that, according to the ordering, best matches the current situation.

[3] There could be further specialisations to allow the dessert to be mentioned first, but we would expect these to arise, and be returned, only under special circumstances.

To a large extent, the narrative-CMOP is a great deal like the episode in memory which it will relate. All of the scenes in the episode become scenes in the C-MOP and keep their ordering. A setting and a conclusion are added as mandatory scenes. Scenes that satisfy some request of the problem solver are also marked as mandatory, as are unusual scenes which enable them.

The LIST-CMOP converts values found in a slot of a frame into TOPIC-CMOPs which become its scenes. The LIST-CMOP has specialisations which include an *ordering function* [11]. The predictive feature "info-type" indexes these specialisations. For example, if constructing a list of ingredients, which are divided into main, secondary and spices, a list with the "main-first" ordering function is returned. All scenes in LIST-CMOPs used to organise requests are optional. Consequently, only requests from the problem solvers will be included in the conversation. However, there are specialisations, retrieved in the context of question answering, that contain mandatory scenes.

Clearly, JUDIS is limited in the types of dialogues that it can organise by the small number of C-MOPs that are implemented. Most noticeably absent from our list is a general problem-solving C-MOP. We were able to avoid implementing this C-MOP because planning a meal can be seen as filling in values for attributes in a frame and captured in the LIST-CMOP. In addition, we expect the interface to have enough control over the conversation to ensure that this limited type of problem solving will suffice. In other domains, or if the user were expected to take a more active part in problem solving, a general problem-solving C-MOP would be needed. A problem-solving C-MOP would be more difficult to instantiate from a procedure than the LIST- or NARRATIVE-CMOP. The interface would have to do a great deal of domain planning to make predictions about topics that would be included in the dialogue. Some of this effort could be saved by creating only abstract templates and allowing the reasoning followed by the problem solvers to link requests to the template. Finding a procedure to create problem-solving C-MOPs without re-creating all of the reasoning needed to solve the problem is an important area of future research.

III Adding a Request to the Template

At the beginning of a meal-planning session, JUDIS retrieves a C-MOP that is instantiated to become the template. This guides the conversation from beginning to end. The opening and closing follow well-established sequences, so we are only concerned here with the middle portion of the dialogue where the meal will be discussed. One of the specialisations of the CATERER-CMOP will be included to predict the middle of the conversation. The template for the example dialogue includes the MAIN-FIRST-CMOP. Requests from the problem solvers are organised into a coherent dialogue by adding them to the template. A new request is added to the template by becoming a scene in a predicted C-MOP. JUDIS first checks to see if the new request matches a request already in the template. If not, the new request is added by merging it into a DISCUSSION-CMOP already in the template.

A Finding Potential Topics

The first step of adding a new request to a DISCUSSION-CMOP is finding a C-MOP where the request can be added. A request can be added to a discussion when their ***subjects match***, when ***the subject of the request is an attribute or value of the subject in the template***, or when ***the new request is associated with the same knowledge structure as the request in the template***. Instead of searching semantic memory to find the possible connections between requests [cf., 4], JUDIS uses knowledge from the reasoners' problem solving. The problem solvers send the interface two pieces of information with each request. The ***chain of reasoning*** from the meal being planned to the attributes that the problem solver was considering when this goal was created is used to find the subject of the request. If a value appears at the end of the path it is the subject. Otherwise the request asks for a value for the attribute. In this case, the attribute will be the subject for the purpose of adding the request to the template. The chain of reasoning also allows a request to be linked with any attribute on the chain. The problem solvers also send information about the ***knowledge structure*** that was being examined when the goal was created. If the request is associated with a frame, a slot can also be sent. If the request is associated with an episode, the episode and any episodes that contain it, if the problem solver has examined them in association with this request, are sent to the interface. For example, when a reasoner sends a goal to find out if guacamole would be appropriate for the appetiser, it also sends **guacamole0**, the frame representing guacamole in semantic memory, and (meal appetiser main-dish) as the chain of reasoning.[4]

We use information from the problem solvers for two reasons. Most importantly, this assures that the connection between the utterances will be acceptable in the context of the current conversation. Also, this information reduces JUDIS' processing effort and can be easily collected as the problem solvers perform the domain task. Using it, JUDIS can simply match information from the problem solvers instead of searching the semantic memory for all possible connections.

To rely on information from problem solving, that information must be "cognitively plausible" in some sense. Information from the same data structures must

[4] Guacamole is placed in the representation of the meal as soon as it is considered by a problem solver and would be the subject of this request.

appear to human users to be linked. Chains of reasoning followed by the problem solvers must appear to be coherent. If this is not the case for problem solvers used by an interface, it must rely on other knowledge structures to provide it with acceptable links between topics. Also, if the problem solvers do not share semantic memory, there must be a way to match knowledge structures that should be considered the same.

B *Merging Requests into Discussions*

All requests can be merged into the template for conversation at some level. If no predicted topic could include the request, it can be added to the maintenance phase where it will be handled as a *true interruption* [5]. If a topic which could have included this request has already been closed, the change-topic scene will mark the return to a previous topic [15]. Utterance 14 in Figure 1 is an example of JUDIS returning to a previous topic.

If problem solver goals on the same subject arrive at the interface sufficiently near each other, they will be grouped together in the template. If not, the topic will be closed before all of the requests that should be associated with it have arrived. JUDIS can return to such topics, so, in the worse case, the conversation is no worse than conversation without the template. If there are not long delays between requests on the same topic, most requests will be merged into the dialogue through a DISCUSSION-CMOP.

JUDIS examines each DISCUSSION-CMOP in the template until it finds one that can be merged with the new request. It looks at the most specific subjects first so that subjects that are most closely connected will be joined. New requests can be merged with DISCUSSION-CMOPs in several ways:

Replace a discussion scene that has no requests as scenes. The simplest form of a DISCUSSION-CMOP is an UTTERANCE or QUESTION/ANSWER-CMOP. These are the forms of a request. If no other requests have been associated with a subject, the discussion-CMOP can be replaced by the new request.

Extend the reasoning to add a new topic. Sometimes the subject of a new request is a very specific aspect of an expected topic. If the request were simply added, the connection between it and the expected topic could be lost. This would cause the dialogue to appear incoherent. It is also difficult for JUDIS to add other requests to a topic which has been filled by a too-specific subject.

We avoid these problems by adding C-MOPs to the template that extend a discussion from a general subject to a more specific one. We have added a ATTR-VAL-CMOP that links the predicted topic to the more specific request. Each attribute in the chain of reasoning and its value are added to the discussion. Figure 2a shows a request concerning "avocado" being added to the template by this method. Because it is connected to the appetiser TOPIC-CMOP through specific attributes, another request, 9, can be easily added through the "presentation" attribute. If a specific attribute will be mentioned, the attributes which connect it to the topic can be mentioned first.

Connect scenes through knowledge structures. Two requests can also be connected because they are part of the same knowledge structure. If both are values in the same slot of a frame or are values of the same attribute of the meal being planned, a LIST-CMOP is used to connect them. If both are scenes in the same episode, a NARRATIVE-CMOP connects them. Here the requests do not have to have the same subject, but are linked to a discussion through one of its scenes. When the type of connection is found, JUDIS searches memory to find the best C-MOP to instantiate. This is done to make sure that any specialisations appropriate for the current situation are found. For example, the LIST-CMOP is specialised to have a main-first ordering when ingredients are connected.

IV Executing the Template

In conversation new problem solving goals arise as the conversation is being conducted. It is impossible to know all of the goals in advance and then arrange them into the best conversation. Instead, the template must be built and executed simulatneously. This means that the template must reflect a coherent conversation at all times. JUDIS achieves this because each goal is added to the template through C-MOPs.

The template is only used as a guide to organise conversation. When JUDIS is to take its turn in conversation, it combines information about the priorities of the requests and how those requests fit into the template to choose its next utterance (see [15] for details). Sometimes the priorities help determine decisions that are not specified by the template, such as chosing the scene to execute first in a partially ordered C-MOP. Other times a goal is so urgent that the template is overridden.

V Organizing a Dialogue with JUDIS: An Example

Consider utterances 10a-10d in the example dialogue. As the arrows indicate, the goals which motivated these requests are re-organised to make the dialogue coherent. When the initial template is built, JUDIS predicts that the GENERAL-INFO, MAIN-COURSE, APPETIZER and DESSERT topics will be included in the dialogue. At this point, JUDIS knows only that the appetiser will be discussed but knows none of the details. Then a request to tell the user about a possible failure with avocados comes from the case-based reasoner. Since the subject of the request is not the appetiser but one of the ingredients of the appetiser, a path is

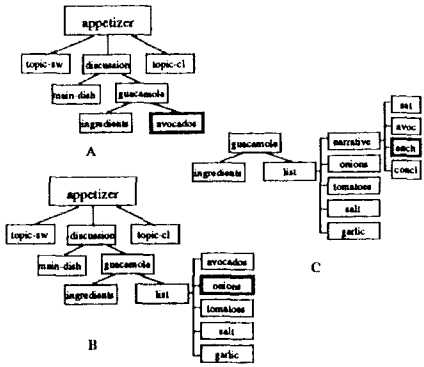

Figure 2: Adding requests to the temlpate.

formed of values and their attributes from the appetizer TOPIC-CMOP to the avocados, as shown in Figure 2a. Next, the from-scratch reasoner discovers that red onions can make guacamole sweeter and decides to send a goal to JUDIS to inform the user. The subject of this request, "onion", is also an ingredient in guacamole. When JUDIS tries to insert the request as a value of the ingredient attribute, it must find a structure that will incorporate both the avocado-request, already associated with the ingredients, and the onion-request which is to be added. JUDIS relies on the information about how the two requests are connected, that they are both values of the same slot of a frame, to begin its search of memory for a C-MOP that can contain both requests. It finds the LIST-CMOP for ingredients and adds a list of all of the ingredients of guacamole to the template (see Figure 2b). The onion and avocado request become the discussions of the onion and avocado TOPIC-CMOPs that are scenes of the new list.

Next, the request about enchiladas comes from the case-based reasoner. Because this request is associated with the same episode as the avocado request, JUDIS makes these requests into a narrative (see Figure 2c). This narrative contains not only the goal-achieving requests (last part of utterance 10a and utterance 10b), but also the mandatory setting (first half of utterance 10a) and conclusion scenes (utterance 10c).

The organisation given by the template and the priorities of the requests determine how this portion of the template will be executed. Though not requested by a problem solver, utterance 8 is included in the dialogue to link the ingredients to the expected appetizer topic. The narrative containing the avocado and enchilada request has a higher goal priority than the onion request, so it is said first. After the narrative, the onion request is executed to finish the list.

VI Discussion

JUDIS is an implemented system which embodies our approach for merging goals into an existing conversational structure. JUDIS relies on its knowledge of conventional discourse structure as a tool for achieving the goals of the system. JUDIS' selection of discourse structures to guide the conversation and options taken within those structures are motivated by the goals of the system. This is in contrast to McKeown's TEXT system [10] which relies on heuristics based on features of the language to select options to pursue within a discourse structure. JUDIS is able to re-organise its goals to fit into the global organisation of the conversation because it relies on predictions and commitments for the whole conversation, as represented in the template. Other methods of generating language from discourse structure [e.g., 10; 12] do not use expectations about the dialogue but rely only on information about the current state of the world and the structure of the discourse to this point.

One of the most important advantages of our approach is that it allows an incoherent stream of goals to be organised to produce a coherent dialogue. There may be cases where an occasional utterance still must be marked with a clue word, but, by forcing the goals to be moderated by the template, we give them a deep structure that provides coherence just as the underlying processing in humans supports their dialogues' conventional structure. Our approach also has two important advantages that are side-effects of using the template to guide the generation of conversation:

The dialogue addresses the needs of the problem solvers. JUDIS' dialogue is motivated by the communication goals that it is sent from the problem solvers. The requests which achieve these goals dictate the details of the template. Only requests and the mandatory scenes of the C-MOPs that connect them are included in the dialogue. This assures that the conversation will be coherent without including optional scenes of a schema that are chosen by language-based heuristics which may have little to do with the current domain task [cf, 10].

Utterances which follow convention are included in the dialogue without being motivated by an explicit goal. When mandatory scenes are needed to connect two requests in a coherent dialogue they are included when a new request is merged into the template. For example, when two requests are connected by a narrative, the setting, conclusion and enabling scenes are added to the template when the narrative is created. In speech act based approaches [e.g., 1; 5; 8], such conventional utterances would be motivated by a speaker's intention. By including these utterances as part of the discourse structure which will achieve the problem solving goals, JUDIS avoids the cost of generating discourse goals and trying to achieve them,

and it can easily distinguish between utterances that are motivated by goals and those that are mandated by convention.

The technique for organisation described above was used successfully to organise the dialogue shown in the example and several others from similar goals. JUDIS was also able to interrupt the organisation prescribed by the template to handle urgent goals and was able to add requests to the dialogue that did not correspond to topics that were given by the caterer's-CMOP.

The success of JUDIS depends, in part, on several characteristics of the problem solving domain and the reasoners. Most importantly, JUDIS is an interface to an advisory system. Although JUDIS was designed to allow the user to take more initiative than is often expected in natural language interfaces [15; 16], JUDIS' method of organising dialogue is most successful if JUDIS controls the conversation to a large extent. To allow more user involvement, more C-MOPs would be needed so that JUDIS could build a template for any organisation known to the user. JUDIS would also need to handle failure of the template – to identify failure and recover when the template no longer predicts the conversation. This is an important area for future research.

Another assumption in our implementation of JUDIS is that the problem solvers will use problem solving strategies and knowledge structures which correspond to those used by humans. Although the combined stream of goals from the problem solvers may not be organised in a way that would seem coherent to people, JUDIS can rely on information from the problem solvers to help it build the connections that lead to a coherent conversation. We feel that the method of organisation described here could also benefit individual problem solvers that do not produce a coherent stream of communication goals and problem solvers that do not have such well-organised knowledge. In these cases, the interface must keep a separate knowledge base or rely on declaratively represented C-MOPs. This also means that we would loose the advantages of using problem solving information as a basis for organising the dialogue.

JUDIS is also helped because very few of its goals are urgent. In a system that very often needs to get additional information from the user in order to continue, it may be difficult to make full use of the template. It would be overridden often or it would delay problem solving.

JUDIS has begun to address the problem of organizing dialogue so that even conversation motivated by an incoherent stream of goals can be easy to understand. Most important to our method is the ability to form partial predictions about the dialogue that can be expanded as goals arrive from the problem solver. In this way, JUDIS can group utterances together to form a coherent whole.

REFERENCES

[1] Sandra Mary Carberry. Pragmatic modeling in information system interfaces. Technical Report 86-07, Department of Computer Science, University of Delaware, 1986. Ph.D. thesis.

[2] Richard E. Cullingford. *Natural Language Processing: A Knowledge Engineering Approach.* Rowan and Littlefield, Totowa, New Jersey, 1986.

[3] Richard E. Cullingford and Janet L. Kolodner. Interactive advice giving. In *Proceedings of the 1986 IEEE International Conference on Systems, Man and Cybernetics*, pages 709–714, Atlanta, Georgia, 1986.

[4] Barbara J. Gross. The representation and use of focus in a system for understanding dialogs. In *Proceedings of the Fifth International Conference on Artificial Intelligence*, pages 67–76, Los Altos, California, 1977. William Kaufmann, Inc.

[5] Barbara J. Gross and Candace L. Sidner. Attention, intention, and the structure of discourse. *Computational Linguistics*, 12(3):175–204, 1986.

[6] Kathy Kellermann, Scott Broetzmann, Tae-Seop Lim, and Kenji Kitao. The conversation MOP: Scenes in the stream of discourse. *Discourse Processes*, 12(1):27–61, 1989.

[7] J.L. Kolodner. *Retrieval and Organizational Strategies in Conceptual Memory.* Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, 1984.

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

[9] William C. Mann. Discourse structures for text generation. In *Proceedings of the Tenth International Conference on Computational Linguistics*, pages 367–375, 1984.

[10] Kathleen R. McKeown. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Languauge Text.* Cambridge University Press, New York, 1985.

[11] L. Polanyi and R. Scha. A syntactic approach to discourse semantics. In *Proceedings of the Tenth International Conference on Computational Linguistics*, pages 413–419, 1984.

[12] Rachel Reichman. *Getting Computers to Talk Like You and Me: Discourse Context, Focus, and Semantics (An ATN Model).* The MIT Press, Cambridge, Mass, 1985.

[13] David E. Rumelhart. Notes on a schema for stories. In Daniel G. Bobrow and Allan Collins, editors, *Representation and Understanding.* Academic Press, New York, 1975.

[14] R.C. Schank. *Dynamic Memory.* Cambridge University Press, New York, 1982.

[15] Elise H. Turner. Integrating intention and convention to organise problem solving dialogues. Technical Report GIT–ICS–90/02, School of Information and Computer Science, Georgia Institute of Technology, 1990. Ph.D. thesis.

[16] Elise H. Turner and Richard E. Cullingford. Using conversation MOPs in natural language interfaces. *Discourse Processes*, 12(1):63–90, 1989.