Pointing: A Way Toward Explanation Dialogue*

Johanna D. Moore
Department of Computer Science
and
Learning Research and Development Center
University of Pittsburgh
Pittsburgh, PA 15260

William R. Swartout
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695

Abstract

Explanation requires a dialogue. Users must be allowed to ask questions about previously given explanations. However, building an interface that allows users to ask follow-up questions poses a difficult challenge for natural language understanding because such questions often intermix meta-level references to the discourse with object-level references to the domain. We propose a hypertext-like interface that allows users to point to the portion of the system's explanation they would like clarified. By allowing users to point, many of the difficult referential problems in natural language analysis can be avoided. However, the feasibility of such an interface rests on the system's ability to understand what the user is pointing at; i.e., *the system must understand its own explanations.* To solve this problem, we employ a planning approach to explanation generation which records the design process that produced an explanation so that it can be used in later reasoning. In this paper, we show how synergy arises from combining a "pointing-style" interface with a text planning generation system, making explanation dialogues more feasible.

Introduction

It has been argued extensively that natural language interaction is critical to the effective use of expert and advisory systems (for example, see [Finin *et al.*, 1986]). Further, we have argued that explanation requires a dialogue [Moore and Swartout, 1989]. In particular, systems must be able to clarify misunderstood explanations, elaborate on previous explanations, and respond to follow-up questions in the context of the on-going dialogue. Moreover, systems must be able to provide further explanations even when the user cannot ask a well-formulated follow-up question.

In [Moore and Swartout, 1989] we described a system that implements a reactive approach to explanation – one that can participate in an on-going dialogue and employs feedback from the user to guide subsequent explanations. Our system explicitly plans the explanations it produces using a set of explanation strategies. The planning process is recorded to capture the "design" of the explanation. This design record tells the system what it was trying to explain, how it explained it, and what alternative ways could have been used to explain the same thing. When a user indicates that he doesn't understand an explanation, the design record is used to provide the conversational context needed in planning a clarifying response. This system demonstrates that dialogue can be supported effectively by explicitly representing and reasoning about the "design" of the system's explanations.

However, building an interface that allows users flexibility in asking follow-up questions poses a difficult challenge. If the system allows users to pose their questions in natural language, it must be able to handle questions or statements that refer to previously given explanations, e.g.:

"Could you please elaborate on the part about applying transformations that enhance maintainability?"

"Could you please explain that last part again?"

This type of question poses a serious problem for a natural language understander. The difficulty arises because such questions make reference to items in the domain of discourse as well as to the discourse itself, so that the natural language analysis system must be capable of understanding both comments made at the object level (about the domain) and comments made at the meta-level (about the discourse). Further, as the first sentence shows, the two levels may be intermixed in a single question. To our knowledge, these referential problems are beyond the capabilities of current natural language understanding systems. Such difficulties mean that it will be hard to achieve dialogue-based explanation capabilities if we rely solely on natural language understanding techniques for accepting feedback from the user.

Fortunately, there is another approach, which we describe in this paper. The idea is to provide the user

*The research described in this paper was supported by the Defense Advanced Research Projects Agency (DARPA) under a NASA Ames cooperative agreement number NCC 2-520.

with a hypertext-style interface, i.e., an interface that allows the user to point to the portion of the system's explanation that he doesn't understand or wants further clarified and then provides a menu of questions that may be asked about the highlighted text. By allowing the user to point to the text he doesn't understand, many of the difficult referential problems in understanding natural language can be avoided. However, for such an interface to be feasible, the system must be able to understand what the user is pointing at; i.e., *the system must understand its own explanations.* Because our system explicitly plans its explanations and records the planning process, it retains the intent behind the explanation, and thus can understand what the user is pointing at.

It is important to note that while we are drawing on a hypertext-like *interface*, the system is not a hypertext system in the traditional sense; i.e., it is not organized as a collection of canned pieces of text interconnected by typed links. Our system differs from a hypertext system in several important ways. First, our system creates text dynamically in response to the user's need for explanations. This text can therefore be tailored to a particular user and situation. A hypertext system would have all the text pre-canned and the user would have to browse through it to find the information he requires. The text cannot be tailored to a particular context.

In a hypertext system, all the things that can be pointed at have to be worked out in advance. It is easy to imagine that users may have questions about items in the texts that were not envisioned, and hence not provided for, by the hypertext designers. In our system, what can be pointed at is determined dynamically, and the links are not worked out in advance. (Indeed, since the texts are not written in advance, it would be difficult to create the links in advance.)

Moreover, as we will see in a later example, what follow-up questions are meaningful is also highly context-dependent. Therefore, the precoded and fixed interconnections employed in a traditional hypertext system would offer the user many possible "links" (corresponding to follow-up questions) that the user might find superfluous. Since one of the main problems with hypertext systems is that users get lost in the network and may even forget what it was they were originally seeking [Carando, 1989, Halasz, 1988], it seems especially important to present a confused user with a small set of pertinent follow-up questions as opposed to a very large set of questions, many of which are irrelevant or even ridiculous. Presenting a user with a follow-up question that the user thinks has just been answered may cause the user to think that he's even more confused than he actually is. In our system, dialogue context (provided by the text plan record) and a user model are used to prune the list of possible questions down to those that appear most relevant.

In this paper, we describe the "pointing" interface we have implemented and combined with our text planning generation system. This combination acts in synergy to support explanation dialogues. The pointing interface allows us to avoid some difficult problems in natural language understanding, while the text planning approach to generation allows us to achieve greater flexibility and sensitivity to context than can be provided with the pre-canned links and text strings of traditional hypertext systems.

Overview of the Reactive Approach

Our explanation facility is part of the Explainable Expert Systems (EES) framework, an architecture for building expert systems that facilitates both explanation capabilities and system maintenance [Neches *et al.*, 1985]. Using EES, we constructed the Program Enhancement Advisor (PEA), an advice-giving system intended to aid users in improving their Common Lisp programs by recommending transformations that enhance the user's code.[1] The user supplies PEA with the program to be enhanced. PEA begins the dialogue with the user by asking what characteristics of the program he would like to improve. The user may choose to enhance any combination of readability, maintainability, and efficiency. PEA then recommends transformations that would enhance the program along the chosen dimensions. After each recommendation is made, the user may accept, reject, or ask questions about the recommendation.

An overview of the explanation generator and its relationship to other components in the system is shown in Figure 1. When the user provides input to the system, the *query analyzer* formulates a discourse goal (e.g., make the hearer know a certain concept, persuade the hearer to perform an action) representing an abstract specification of the response to be produced and posts this goal to the *text planner*. The planner then searches its library of explanation strategies looking for candidates that could achieve the current goal. In general, there may be many strategies capable of achieving a given goal and the planner employs a set of *selection heuristics* to determine which of the candidate strategies is most appropriate in the current situation. These selection heuristics take into account information about the hearer's knowledge state (as recorded in the *user model*), the conversation that has occurred so far (as recorded in the *dialogue history*), and information about whether or not a strategy requires assumptions to be made. Once a strategy is selected, it may in turn post subgoals for the planner to refine. Planning continues in a top-down fashion until the entire plan is refined into primitive operators, which in our system are speech acts ([Searle, 1979]) such as `INFORM` and `RECOMMEND`.

[1]PEA recommends transformations that improve the "style" of the user's code. It does not attempt to understand the content of the user's program.

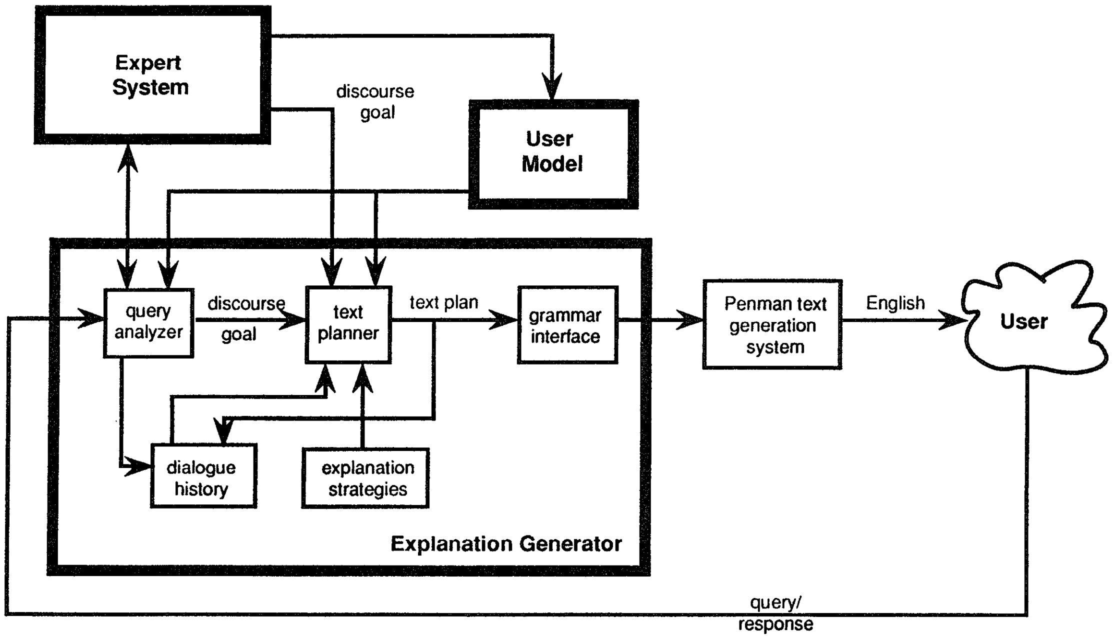


Figure 1: Architecture of Explanation System

As the system plans explanations to achieve its discourse goals, it records the goal/subgoal structure of the response being produced. In addition, it keeps track of any assumptions it makes about what the user knows, as well as alternative strategies that could have been chosen at any point in the planning process. Once a text plan is completed, it is recorded in the dialogue history and passed to the *grammar interface* to be transformed into a form suitable to be passed, a sentence at a time, to the Penman text generation system [Mann and Matthiessen, 1983] for translation into English. After producing an utterance, the system awaits the user's feedback. The user may provide feedback in several ways. He may indicate that the explanation was understood and therefore that the system can move to a new topic. He may ask one of a prescribed set of follow-up questions. In addition, in cases where the user cannot formulate a question, he can type "Huh?" and the system will provide an elaborating or clarifying response. Alternatively, he can use the "pointing" interface which is the topic of this paper, and point to the portion (noun phrase, clause, or sentence) of the explanation that he finds problematic, and a menu of follow-up questions the system can answer about that portion of text will appear.

The completed text plans stored in the dialogue history provide the dialogue context the system needs to respond appropriately to the user's feedback. A completed text plan is an explicit representation of the planning or "design" process that produces an explanation. As described in [Moore and Paris, 1989], a completed text plan represents the roles individual clauses in the resulting text play in achieving discourse goals, as well as how the clauses relate to one another rhetorically. In addition, information about what entities are salient at each point in the explanation (attentional information) can be derived from a text plan. In previous work, we have demonstrated that this context can be used to disambiguate follow-up why-questions [Moore and Swartout, 1989], to select perspective when describing or comparing objects [Moore, 1989a], to avoid repeating information that has already been communicated [Moore and Paris, 1989], and to allow the system to recover from failures when feedback from the user indicates that he hasn't understood the system's utterance [Moore, 1989b]. In this paper we show how a completed text plan allows the system to provide an intelligent hypertext-style interface, one that provides a context-sensitive menu reflecting the ongoing dialogue.

An Example Dialogue

We have found that having the text plans of the system's responses recorded in the dialogue history makes it possible to automatically generate a menu of possible follow-up questions the user may wish to ask about an utterance. Because the text plans provide information about the context in which the highlighted text appears, questions that would appear redundant to the

SYSTEM What characteristics of the program would you like to enhance? [1]
USER Readability and maintainability. [2]

⋮

SYSTEM You should replace (SETQ X 1) with (SETF X 1). [3]
USER Why? [4]
SYSTEM I'm trying to enhance the maintainability of the program by applying transformations that enhance maintainability. SETQ-TO-SETF is a transformation that enhances maintainability. [5]

Figure 2: Sample Dialogue

user can be ruled out.

For example, consider the sample dialogue with our system shown in Figure 2. In this dialogue, the system recommends that the user perform an act, namely replace (SETQ x 1) with (SETF x 1) (line [3]). The user, not immediately convinced that this replacement should be made, responds by asking "Why?" (line [4]). Because the user's why-question follows a recommendation, the query analyzer interprets it as a request by the user to be persuaded to do the recommended act. In our text planning formalism, discourse goals are represented in terms of the effects that the speaker (the PEA-system) wishes to have on the hearer (the user). In this case, the discourse goal posted to the text planner is: (PERSUADED USER (GOAL USER (DO USER REPLACE-1))) where REPLACE-1 is the act of replacing (SETQ x 1) with (SETF x 1). This goal expression can be paraphrased by saying that the system now has the goal to achieve the state where the hearer is persuaded to adopt the goal of performing the replacement.

Figure 3 shows the final result of the planning process, i.e., the completed text plan for achieving the goal (PERSUADED USER (GOAL USER (DO USER REPLACE-1))).[2] Basically this text plan does the following. To persuade the user to do an act, the system motivates that act in terms of a mutual domain goal that the act is a step towards achieving. In this case, the system persuades the user to replace SETQ with SETF (REPLACE-1) by motivating this act in terms of the shared domain goal to enhance the maintainability of the program. Thus, the discourse subgoal (MOTIVATION REPLACE-1 ENHANCE-MAINTAINABILITY) is posted.

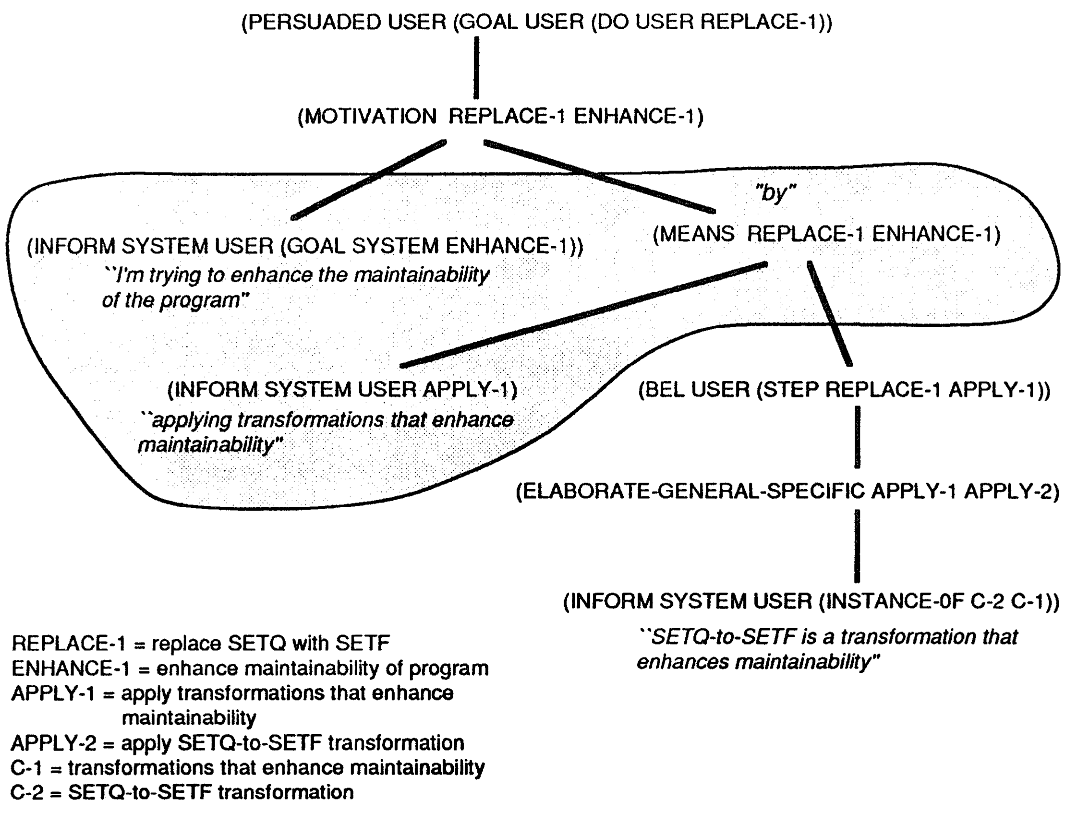
One strategy for achieving this discourse subgoal is to inform the user of the domain goal that the system is trying to achieve and then to establish that the act in question is part of the method for achieving that domain goal. Applying this strategy to achieve the discourse goal (MOTIVATION REPLACE-1 ENHANCE-MAINTAINABILITY) in turn gives rise to two discourse subgoals, one for informing the user of the domain goal the system is trying to achieve ((INFORM SYSTEM USER (GOAL SYSTEM ENHANCE-MAINTAINABILITY))), and one to establish that the act being persuaded is indeed part of the method for achieving this goal ((MEANS REPLACE-1 ENHANCE-MAINTAINABILITY)). Speech acts, such as INFORM, are achieved by operators that construct an input specification for the sentence generator. From the text planner's perspective they are considered primitive.

However, the discourse subgoal (MEANS REPLACE-1 ENHANCE-MAINTAINABILITY) requires further refinement. To establish a MEANS relation between the goal just mentioned and the recommended act, the planner has chosen a strategy that informs the user of the method used for achieving the goal (here APPLY-TRANSFORMATIONS-THAT-ENHANCE-MAINTAINABILITY) and then posts a discourse subgoal of making the hearer believe that the recommended act is a step in this method, i.e., (BEL USER (STEP REPLACE-1 APPLY-TRANSFORMATIONS-THAT-ENHANCE-MAINTAINABILITY)).

PEA's domain knowledge contains the information that the domain goal APPLY-TRANSFORMATIONS-THAT-ENHANCE-MAINTAINABILITY is achieved by applying each of the individual maintainability-enhancing transformations known to the system in turn. The appropriate rhetorical strategy for expressing this domain reasoning is ELABORATION-GENERAL-SPECIFIC, in which a general concept is elaborated by citing a specific instance of it. Making the hearer believe that REPLACE-1 is a step in achieving the goal APPLY-TRANSFORMATIONS-THAT-ENHANCE-MAINTAINABILITY thus boils down to informing him that SETQ-TO-SETF is one of the maintainability-enhancing transformations.

The text plan shown in Figure 3 produces the system's response appearing on line [5] in the sample dialogue of Figure 2. After this utterance is produced, the user wishes to ask a follow-up question about an

[2]A complete discussion of the plan language and planning mechanism is beyond the scope of this paper and has been reported elsewhere; see [?].



(PERSUADED USER (GOAL USER (DO USER REPLACE-1))
(MOTIVATION REPLACE-1 ENHANCE-1)
"by"
(INFORM SYSTEM USER (GOAL SYSTEM ENHANCE-1))
"I'm trying to enhance the maintainability of the program"
(MEANS REPLACE-1 ENHANCE-1)
(INFORM SYSTEM USER APPLY-1)
"applying transformations that enhance maintainability"
(BEL USER (STEP REPLACE-1 APPLY-1))
(ELABORATE-GENERAL-SPECIFIC APPLY-1 APPLY-2)
(INFORM SYSTEM USER (INSTANCE-OF C-2 C-1))
"SETQ-to-SETF is a transformation that enhances maintainability"
REPLACE-1 = replace SETQ with SETF
ENHANCE-1 = enhance maintainability of program
APPLY-1 = apply transformations that enhance maintainability
APPLY-2 = apply SETQ-to-SETF transformation
C-1 = transformations that enhance maintainability
C-2 = SETQ-to-SETF transformation

Figure 3: Completed Text Plan for Persuading User to Replace SETQ with SETF

aspect of the system's response and has positioned the mouse so that the sentence

> I'm trying to enhance the maintainability of the program by applying transformations that enhance maintainability.

is highlighted. To ask a follow-up question about this text, the user clicks the mouse and a menu of possible follow-up questions appears. In this context, the follow-up questions that will be contained in the menu are shown in Figure 4. Note that there are many questions that could be asked about this text that are not included in this menu. For example,

Q1: Why are you trying to enhance the maintainability of the program?

Q2: How do you enhance the maintainability of the program?

Q3: Why are you applying transformations that enhance maintainability?

However, given the dialogue that has already taken place, it is very likely that the user already knows the answers to these questions, and therefore they should not be included in the menu. In the next section we show how our system generates candidate menu items and eliminates those such as Q1–Q3 which are almost certainly inappropriate in this context and therefore only clutter the menu and may confuse the user.

Generating Follow-up Questions for Menu

In the example under consideration, the user has highlighted a complex clause corresponding to the shaded region of the text plan shown in Figure 3. When the user selects a complex clause, there are three sources of follow-up questions: the two simple clauses that make up the complex, and the relation between the two simple clauses. For example, in this case where the user selected the text

> (1) I'm trying to enhance the maintainability of the program by (2) applying transformations that enhance maintainability.

the system generates follow-up questions from each of the simple clauses (1) and (2), but must also consider follow-up questions that arise because of the MEANS relation that exists between (1) and (2), here explicitly

Q4: How do you apply transformations that enhance maintainability?

Q5: Why are you trying to enhance the maintainability of the program by applying transformations that enhance maintainability?

Q6: What are transformations that enhance maintainability?

Figure 4: Follow-up Question Menu After Eliminating Superfluous Questions

expressed by the term "by".

Generating Menu Entries for Simple Clauses. In our system, simple clauses arise from the leaf nodes of a completed text plan tree, i.e., speech act nodes. For each simple clause, the system generates two types of follow-up questions: questions about the entire clause, and questions about the objects that are participants in the clause. The system currently considers two types of questions that can be asked about speech acts: why-questions and how-questions. We have found that the interpretation of why and how-questions asked about a speech act is dependent on the type of speech act and that in some cases it is not possible to form both a why and how-question from a given speech act.[3]

In the current example, clause (1) informs the user of a goal the system is trying to achieve. From this type of `INFORM` speech act, the system can form both a how and a why-question, namely

Q1: Why are you trying to enhance the maintainability of the program?

Q2: How do you enhance the maintainability of the program?

Clause (2) informs the user of the method the system is currently applying. Again, both a how and a why-question can be formed, namely

Q3: Why are you applying transformations that enhance maintainability?

Q4: How do you apply transformations that enhance maintainability?

Each simple clause in the text produced by the system is made up of a process (e.g., a relation or an action), and the participants and circumstances associated with that process. So, for example, the simple clause "I (the system) apply transformations that enhance the maintainability of the program" consists of a process, `APPLY`, whose actor is the concept `PEA-SYSTEM` and whose object is a complex concept that is expressed as "transformations that enhance the maintainability of the program". This expression includes mention of another process `ENHANCE` and the concepts `TRANSFORMATIONS`, `MAINTAINABILITY` and `PROGRAM`. Since each of these concepts is expressed in the final text, the user may have questions about any of them. To generate candidate follow-up questions for the concepts mentioned in a single speech act, the system examines the complete specification of the sentence passed to the text generator by the grammar interface. This specification contains an entry for each of the concepts that will be uttered in the final text. Each concept appearing in that specification becomes a potential questioned item and the system considers generating a question of the form "What is a ...?" for each.

Generating Menu Entries for Complex Clauses. Because the user has highlighted a complex clause, the system must also consider follow-up questions that arise because of the relation that exists between the two simple clauses, in this case `MEANS`. Currently the system attempts to formulate only why-questions from complex clauses. In the current example, the system formulates the question:

Q5: Why are you trying to enhance the maintainability of the program by applying transformations that enhance maintainability?

In other words, why is the system using this particular method (applying transformations that enhance maintainability) to achieve the goal in question (enhancing the maintainability of the program) as opposed to trying some other strategy?

Although not illustrated in this example, there is one additional source of questions to be included in the follow-up menu. As we stated earlier, the planner records any assumptions it makes about the user's knowledge during the planning process. In addition to the questions that come from the highlighted text itself, the system also generates questions if any assumptions were made in planning the text. If there are any assumptions recorded at the plan nodes that

[3]For example, the `ASK` speech act causes a question to be posed to the user. If the user points at text that was generated as the result of an `ASK` speech act, a sensible why-question can be formed. The user may wish to understand why the system needs to know the answer to its question in order to perform its task; i.e., he would like to ask "Why are you asking me this question?". However, it is not possible to form a meaningful how-question for text produced by an `ASK` speech act. It does not make sense for the user to ask "How are you asking me this question?" For a more thorough discussion of the types of speech acts used in our system and what questions can be formed, see [Moore, 1989a].

created the highlighted text or at any of their ancestor nodes higher up in the text plan tree, the system will generate questions for the follow-up menu that serve to check these assumptions. Note that an assumption could have led to the user's need to ask a follow-up question at this point.

Eliminating Candidate Menu Entries. If we simply used the rules described above for generating menu entries when the user selected a piece of text, the menu for our current example would include all of the questions shown in Figure 5. However, many of these questions are questions that the user would probably not wish to ask. As we have argued above, presenting the user with a menu uncluttered by superfluous entries is desirable.

Note that in the context of the current dialogue, the user is almost certainly not asking any of the questions Q1 – Q3, or Q8 – Q10. The user would not ask Q1 because earlier in the sample dialogue, the system asked what characteristics of the program should be enhanced and the user responded that he would like the system to enhance readability and maintainability. Therefore, the user would not ask why the system is achieving the goal of enhancing maintainability. Q8 – Q10 are questions about basic concepts that almost any user of the system would be familiar with. The reason the user is not likely to ask Q2 or Q3 is because both of these questions were answered when the system said

> I'm trying to enhance the maintainability of the program by applying transformations that enhance maintainability.

Our system can detect these various conditions and omit these superfluous options from the menu using the context provided by the text plans recorded in its dialogue history and the knowledge it has about the current user stored in its user model. The system eliminates candidate menu entries using three heuristics:

1. Don't pose questions that have recently been answered.
2. Don't pose questions to justify shared goals.
3. Don't pose questions to which the user knows the answers.

Let us see how these three heuristics are applied in this example. When the user highlights the text shown in Figure 2, mouse-handling code returns a pointer to the portion of the text plan for the previous response that caused this piece of text to be generated. This corresponds to the shaded area in Figure 3.

The system eliminates Q2 and Q3 using the first heuristic. The semantics of the rhetorical relation `MEANS`[4] are that the `MEANS` relation associates the statement of a goal with a statement of the method used to achieve that goal. Thus, since the act `APPLY-TRANSFORMATIONS-THAT-ENHANCE-MAINTAINABILITY` appears in the second position of a `MEANS` relation, the system determines that it has just told the user why it is using this method, namely to achieve the goal `ENHANCE-MAINTAINABILITY`. Therefore, the system determines that the user is almost certainly not asking Q3 and hence it can be omitted from the menu. Similarly, the system can determine that it has just answered the how-question of Q2. Again, from the `MEANS` relation, the system determines that it has just told the user how it is achieving the goal `ENHANCE-MAINTAINABILITY`, namely by employing the method `APPLY-TRANSFORMATIONS-THAT-ENHANCE-MAINTAINABILITY`. Therefore, Q2 is not added to the menu.

Q1 is eliminated using the second heuristic. When the system asks the user what characteristics are to be enhanced, the user's responses are recorded in the user model (see lines [1] and [2] of Figure 2). Thus, when the system considers forming a question asking why it is achieving a goal, it first checks to see if that goal is a mutual goal of both the user and the system. If so, the candidate question is eliminated.

Finally, the system can eliminate many of the "What is a ...?" questions using the third heuristic and the information contained in the user model. Recall that all of the concepts that will be mentioned in an utterance become potential questioned items. When selecting questions for inclusion in the menu, the system compares the list of potential items against the user model and eliminates all of those concepts that the user model indicates the user already knows. In this way, the follow-up question menu will not be cluttered with questions about concepts the user already knows.

In the current example, suppose that the only concept not indicated to be known to the user is the complex concept `MAINTAINABILITY-TRANSFORMATIONS`. Thus only the question

Q6: What are transformations that enhance maintainability?

will be included in the menu.

After pruning out the follow-up questions that can be ruled out by the dialogue context and the user model, the menu of follow-up questions would include only the three questions shown earlier in Figure 4. This menu is uncluttered by questions the user is almost certainly not asking and therefore presents the user with a concise set of the most meaningful follow-up questions that the system can handle in this context. A user who is confused to begin with will be greatly facilitated by being presented with a small set of the most relevant questions.

[4]`MEANS` is one of 25 rhetorical relations whose semantics are defined in Rhetorical Structure Theory (RST) [Mann and Thompson, 1987], a descriptive theory of the organization of English text that has identified the relations that normally occur between portions of coherent text.

Q1: Why are you trying to enhance the maintainability of the program?
Q2: How do you enhance the maintainability of the program?
Q3: Why are you applying transformations that enhance maintainability?
Q4: How do you apply transformations that enhance maintainability?
Q5: Why are you trying to enhance the maintainability of the program by applying transformations that enhance maintainability?
Q6: What are transformations that enhance maintainability?
Q7: What are transformations?
Q8: What is maintainability?
Q9: What is a PEA-system?
Q10: What is a program?

Figure 5: Follow-up Question Menu without Eliminating Superfluous Questions

Conclusions

Explanation requires a dialogue, where the user can formulate questions about previously given explanations. However, the follow-up questions a user is likely to ask are difficult for natural language understanding systems to process because they mix meta-level references to the discourse with object-level references to the domain. In this paper, we have argued that some of the difficult natural language understanding issues can be avoided through the use of a hypertext-like "pointing" interface that lets a user indicate what parts of the system's explanations should be elaborated by pointing at them with a mouse. To respond appropriately to the user's pointing, the system needs to know what it was trying to say in the text the user highlighted. Our approach to explanation generation uses a text planner that captures the intent behind an explanation so that the system can reason with it. Because the text in our system is dynamically generated, it is much more flexible than the pre-canned texts of traditional hypertext systems. Further, by recording the text planning process, important aspects of the dialogue context are captured. This dialogue context can be used to prune irrelevant or unnecessary options out of the pop-up menu of follow-up questions. Our system demonstrates the synergistic support for dialogue that can be achieved by combining a "pointing" interface with a text planning generation system.

References

Patricia Carando. Shadow: Fusing hypertext with AI. *IEEE Expert*, 4(4):65–78, 1989.

Timothy W. Finin, Aravind K. Joshi, and Bonnie Lynn Webber. Natural language interactions with artificial experts. *Proceedings of the IEEE*, 74(7), July 1986.

Frank G. Halasz. Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Communications of the Association for Computing Machinery*, 31(7):836–870, 1988.

William C. Mann and Christian Matthiessen. Nigel: A systemic grammar for text generation. Technical Report RR-83-105, USC/Information Sciences Institute, February 1983.

William C. Mann and Sandra A. Thompson. Rhetorical Structure Theory: A theory of text organization. In Livia Polanyi, editor, *The Structure of Discourse*. Ablex Publishing Corporation, Norwood, N.J., 1987.

Johanna D. Moore and Cécile L. Paris. Planning text for advisory dialogues. In *Proceedings of the Twenty-Seventh Annual Meeting of the Association for Computational Linguistics*, Vancouver, B.C., Canada, June 26-29 1989.

Johanna D. Moore and William R. Swartout. A reactive approach to explanation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 20-25 1989.

Johanna D. Moore. *A Reactive Approach to Explanation in Expert and Advice-Giving Systems.* PhD thesis, University of California, Los Angeles, 1989.

Johanna D. Moore. Responding to "huh?": Answering vaguely articulated follow-up questions. In *Proceedings of the Conference on Human Factors in Computing Systems*, Austin, Texas, April 30 - May 4 1989.

Robert Neches, William R. Swartout, and Johanna D. Moore. Enhanced maintenance and explanation of expert systems through explicit models of their development. *IEEE Transactions on Software Engineering*, SE-11(11), November 1985.

John R. Searle. *Expression and Meaning: Studies in the Theory of Speech Acts.* Cambridge University Press, Cambridge, England, 1979.