NASA-CR-193422

FINAL
IN-82-CR
34CIT.
194132
P-29

EXPLAINABLE EXPERT SYSTEMS

A Research Program in Information Processing

Final Report

Cécile L. Paris
Information Sciences Institute
Of the University of Southern California
4676 Admiralty Way
Marina del Rey, Ca 90292-6695

(NASA-CR-193422) EXPLAINABLE EXPERT SYSTEMS: A RESEARCH PROGRAM IN INFORMATION PROCESSING Final Report (University of Southern California) 29 p

N94-20133

Unclas

G3/82 0194132

Proposal NCC 2-520
Sponsored by the Defense Advanced Research Projects Agency (DoD) through
NASA-Ames Research Center Moffett Field, California 94035

Contents

1 Background and High-Level Project Objectives

Our work in Explainable Expert Systems (EES) had two goals: to extend and enhance the range of explanations that expert systems can offer, and to ease their maintenance and evolution. As suggested in our proposal, these goals are complementary because they place similar demands on the underlying architecture of the expert system: they both require the knowledge contained in a system to be explicitly represented, in a high-level declarative language and in a modular fashion.

With these two goals in mind, the Explainable Expert Systems (EES) framework has been designed to remedy limitations to explainability and evolvability that stem from related fundamental flaws in the underlying architecture of current expert systems. We identified the following flaws:

1. **A weak knowledge representation with no separation of concerns**. The rules or methods of current expert systems mix many different kinds of knowledge, such as domain facts, problem-solving knowledge, and terminology. This intertwining of knowledge makes the explanations such systems can provide confusing. It also reduces a system's modularity, making it difficult to modify.

2. **No design record**. To provide good explanations for expert systems, it is not enough to explain what a system is doing, it is also necessary to explain *why* the system is doing what it's doing. Producing such justifications of an expert system's behavior requires knowing how that system was *designed* and put together. In conventional expert systems, the process of composing general domain facts and problem-solving knowledge into specific rules is performed, unrecorded, in the system builder's mind. The design decisions behind the system are not available; hence, the system cannot provide justifications of its behavior.

3. **Limited explanation techniques**. Most current expert systems use simple methods (templates or canned text) to produce explanations and cannot participate in a dialogue with the user, to clarify mis-understanding or elaborate on a response. Yet, explanation is a highly interactive process and producing explanations is a complex problem.

We first concentrated on the problem of good explanations. As we will see later, the solution to this problem also supports our goal of providing a framework for expert systems in which maintenance and evolution is eased.

We approached the problem of providing good explanations on two fronts. On one hand we developed a framework in which the knowledge necessary to provide good explanations in captured, in a form usable by an explanation module: the knowledge that justifies the system's specific actions is *explicitly* represented, in a *modular and declarative* fashion, using a *high-level knowledge representation language* that supports different levels of abstraction. In this framework, the specific actions necessary to solve an instance of a problem is then *derived automatically* from the underlying knowledge sources, and derivation knowledge (or design record) is kept.

To support good explanations, however, a system not only needs to capture additional knowledge, but it also needs to present its explanations in a flexible and responsive manner. Conventional explanation facilities are neither flexible nor responsive because they use template-based natural language generation techniques to produce 'one-shot' explanations. Each explanation can be presented in only one way. If the user fails to understand an explanation, the system cannot offer a clarifying alternative explanation. Because such systems do not understand how their own explanations were put together, they cannot answer users' queries in the context of an on- going dialogue. To confront these problems, we have developed a new approach to explanation production – one which supports dialogues and in which text is synthesized directly from the underlying knowledge sources.

Section 2 presents our work on representing the knowledge that is necessary to support good explanations and describes the resulting EES framework together with an example. Section 3 presents our approach to explanation and illustrates it with a sample dialogue. Section 4 explains how the EES framework also supports our goal of easing evolution and maintainability of expert systems. Finally, Section 5 summarizes our accomplishments and Section 6 presents the publications done under this contract.

2 Capturing the knowledge required for explanations

An explanation of an expert system's behavior is more than simply an explanation of what a system is doing. It is often also necessary to explain *why* the system is doing what it's doing. To producing such justifications, a system must have general knowledge about the domain, about how to solve a problem in that domain, and about how specific actions were derived from more general principles. A system must thus know how it was *designed* and put together. Of course, capturing the complete design rationale behind a system is a daunting task. In the Explainable Expert Systems framework, we have focussed our attention on capturing the aspects of a system's design that are important for producing good explanations, including justifications of the system's actions, explications of general problem-solving strategies, and descriptions of the terminology used by the system. In particular, we have focussed on capturing *how* a general principle was applied in a particular domain or a particular case. In EES, we can represent both the general principles from which the system was derived *and* how the system was derived from those principles.

2.1 From General Principles to Specific Actions

To capture the design rationale behind an expert system, EES supports representation of the general principles from which the system is designed. For example, in a medical domain, one can explicitly represent the principle that, before giving a drug, one should determine whether the patient may be overly sensitive to it and adjust the dosage accordingly. As a result, this principle can be presented to a user to explain an action derived from it.

Being able to explain this principle, however, is not enough. It is also necessary to explain how the general principle has been applied to the particulars of the problem at hand. In fact, often the users of an expert system will already share an understanding of the general principles upon which the system is based. They will, however, have questions such as: "Why is this particular finding important in diagnosing the patient?" "Why is it important to know this particular lab value?" Producing responses to these questions involves showing how the immediate concerns of the expert system (such as the value of some finding) arise as a consequence of instantiating the general principles with the particulars of the domain. The EES framework has been designed to support exactly this sort of explanation.

To provide this support, the system needs a way of representing the general problem-solving principles, the domain knowledge that will be used to specialize the principles, and mechanisms for linking the principles to the specifics of the domain. To this end, EES provides:

- **A representation for domain-independent and domain-dependent problem-solving principles.** The framework must represent these principles so that it can explain how they have been specialized to particular situations. EES has adopted a plan-based approach to the representation of problem-solving knowledge. Problem-solving principles, whether general or specific, are represented as *plans.* Each plan has a *capability description* describing what the plan is useful for (e.g., 'diagnose faulty component', or 'find the truth value of a conjunction'). During problem-solving, descriptions of tasks to be performed are posted as *goals.* Plans that may be applicable for achieving a particular goal are found by matching the goal against the capability descriptions of the system's plans. Each plan contains a *method* which is a sequence of steps for achieving the goal.

- **A representation for domain knowledge.** The system must have knowledge that describes how the domain operates. For example, in a system for diagnosing problems with an electronic circuit, such knowledge might include the circuit schematic and descriptions of the behavior of the devices that make up the circuit. This knowledge is used by the EES framework in applying a general principle to a specific problem. In EES, domain models are constructed using the conceptual structures of a KL-ONE style knowledge representation system (Brachman and Schmolze, 1985) and are thus organized into a generalization hierarchy.*

  In addition to domain-specific concepts, a domain model in EES also includes a number of abstract, domain-independent concepts such as `decomposable-object` or `generalized-possession`. These abstract concepts serve as the foundations upon which the domain specific portion of the model is built.†

- **A way of linking problem-solving principles and domain knowledge.** A key explanation problem that EES has addressed is the problem of explaining how a specific action that the system is taking follows from one of the general principles that the system is based upon. The *program writer* in EES is responsible for performing the reasoning that produces specific actions from general principles. It does so through two mechanisms. The first is *specialization.* A capability description for a plan may contain variables which are bound when the plan is matched against a goal. For example, in

*We have used several KL-ONE style representation languages during the course of the project, and in fact developed one ourselves because of the expressibility limitations of current languages. Our results while developing this language were later folded into LOOM (MacGregor, 1988; MacGregor, 1991), the knowledge representation language we adopted for the final version of the system.

†Some of these abstract concepts are pre-defined and come from the Penman Upper Model (Bateman *et al.*, 1989). The distinctions that are made among concepts in the upper model reflect some of the distinctions that need to be made in paraphrasing the conceptual model into natural language explanations.

the goal 'diagnose faulty *component*' the term *component* is a variable that can match any concept of the class `component`. If this capability description is matched against the goal 'diagnose faulty DECServer' then *component* will be bound to 'DECServer'. Before the method part of the plan is run, all occurrences of *component* in the method are replaced by 'DECServer'. This process of specialization is recorded so that the relation between specific actions taken by the system and the more general principles they stem from may be explained.

The second mechanism for specializing general principles to specific actions is *reformulation.* In EES, if no plans can be found to achieve a goal, the system attempts to reformulate the goal into a new goal (or set of goals) for which plans can be found, and which is equivalent to the original goal. To reformulate a goal, the system must understand what it means. In EES, we represent goals as conceptual structures composed from concepts in the underlying knowledge representation. By representing goals in this way, we capture the semantics of the goals so that reformulation can be performed. It is primarily through reformulation that domain knowledge is integrated into the process of realizing specific actions from general principles. As an example, if the system's domain knowledge states that *concept-a* $\equiv$ *concept-b* and no plans can be found to achieve the goal, 'diagnose x', where x is a specialization of *concept-a*, the system will use its knowledge of the equivalence to reformulate the original goal into a new one, 'diagnose y', where y is a specialization of *concept-b*. This results in posting a goal which is equivalent to the original goal but has been reformulated so that additional candidate plans can be considered. Like the process of specialization described above, the process of reformulation is recorded, so that the linkage can be explained. Several kinds of reformulations are possible, including:

- a *covering reformulation*, which may be applied when a goal can be split into several subgoals, which *together* cover the original goal.
- an *individualization reformulation*, which occurs when a goal over a set of objects (such as `evaluate symptoms`) can be turned into a set of goals over each of the individual items in the set (e.g., `evaluate symptom-1`, `evaluate symptom-2`, etc).
- a *redescription reformulation*, which can be applied when the goal can be *redescribed* in different terms, using terminological mappings which are defined in the domain model.

A block diagram of the resulting framework is shown in Figure 1.

2.2 An example

To illustrate our points, we give examples from the Program Enhancement Advisor (PEA), a prototype expert system built using EES (Neches *et al.*, 1985). PEA is an advice-giving system intended to aid users in improving their Common Lisp programs by recommending transformations that enhance the user's code. PEA recommends transformations that improve the "style" of the user's code. It does not attempt to understand the content of the

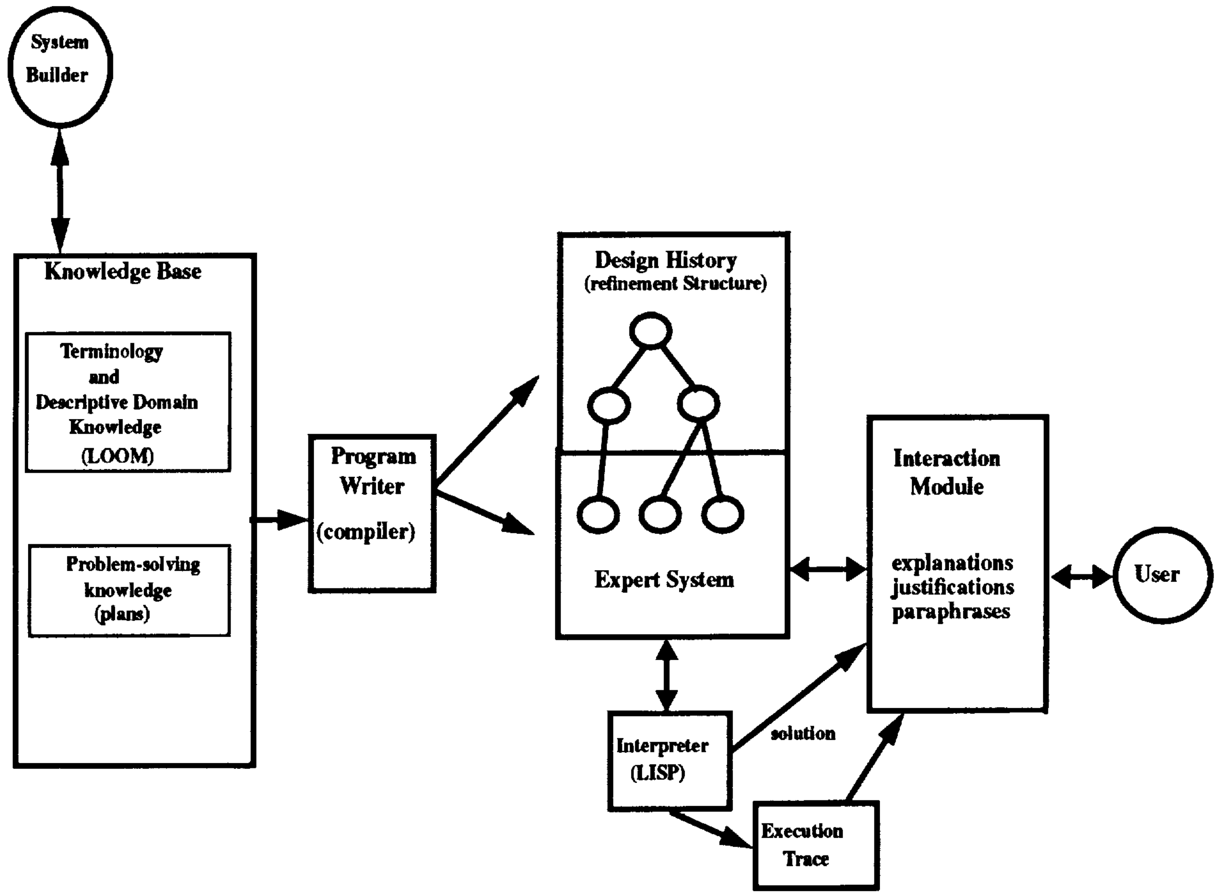
System Builder
Knowledge Base
Terminology and Descriptive Domain Knowledge (LOOM)
Problem-solving knowledge (plans)
Program Writer (compiler)
Design History (refinement Structure)
Expert System
Interaction Module
explanations
justifications
paraphrases
User
Interpreter (LISP)
solution
Execution Trace

Figure 1: The Explainable Expert System framework

user's program. 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 readability and/or maintainability. PEA then recommends transformations that would enhance the program along the chosen dimensions. After each recommendation is made, the user is free to ask questions about the recommendation.

Using a conventional expert system building approach, PEA would have consisted of a collection of transformation rules for each characteristic to be enhanced, indicating a 'bad programming style' (with respect to that specific characteristic) to be replaced by a 'good programming style'. For example, to enhance maintainability, the system would have included a set of rules such as 'replace `setq` with `setf`'. The problem with this approach is that it is not possible to justify why a particular transformation achieves the goal of enhancing the program. One would like the system to explain that `setf` is being recommended as a replacement for `setq` because it is a more general construct and hence using it would make the program easier to maintain.[‡] Unless the rationale behind the transformations is captured,

[‡]Some readers (and users) may disagree with this particular example, and feel that replacing `setq` with `setf` does not improve the maintainability of a program. In such situations, seeing the rationale behind the

it is not possible to give such explanations.

In the EES framework, the knowledge required to produce such explanations is captured. Specific actions, such as the ones above, are *derived* from general principles and domain knowledge. Building PEA in the EES framework requires building a domain model that describes the various concepts and relations used in the domain, as well as a plan knowledge base that includes the general principles for solving the problem of enhancing a program.

The domain knowledge in PEA. The domain knowledge in PEA contains definitions of the terms used in the domain, together with their relations (structural or causal). It includes descriptive knowledge about LISP programming constructs and transformations between actual constructs. This descriptive knowledge explicitly states facts such as:

1. A program is a decomposable object owned-by a user;
2. A transformation has two parts: a left-hand-side (lhs), which is a program construct, and a right-hand-side (rhs), which is a program construct;
3. A maintainability-enhancing-transformation is a transformation whose rhs' use is more general than its lhs' use;
4. Storage locations are named-by s-expressions;
5. An access function is an s-expression;
6. A generalized variable is named-by an access-function;
7. A simple variable is named-by a symbol;
8. Setf can be used to assign a value to a generalized variable;
9. Setq can be used to assign a value to a simple variable;
10. The `setq-to-setf` transformation is a maintainability-enhancing-transformation whose lhs is the `setq` function and whose rhs is the `setf` function.
11. Local transformations and distributed transformations together cover the set of transformations.[§]

Using a KL-ONE style formalism, all of this information is represented as *concepts* organized into a generalization hierarchy, and *relations* (or *roles*) between concepts. So, for example, fact (1) above is represented by having the concept `transformation` be a specialization of the concept `decomposable-object` (using the `is-a` link). Fact (2) above is represented by the relations `lhs` and `rhs` (both specializations of the relation `generalized-possession`),

recommendation is critical, because it helps a user understand why the system made its recommendation.

[§] *Local* transformations are those whose applicability can be determined by looking at a single s-expression (such as `setq-to-setf`), while *distributed* transformations require looking at several places in a user's program before their applicability can be determined. An example of a distributed transformation would be replacing explicit accessors such as **CAR** and **CADR** with record-based accessors. This distinction is important because different methods are used to search for opportunities to apply each kind of transformation.

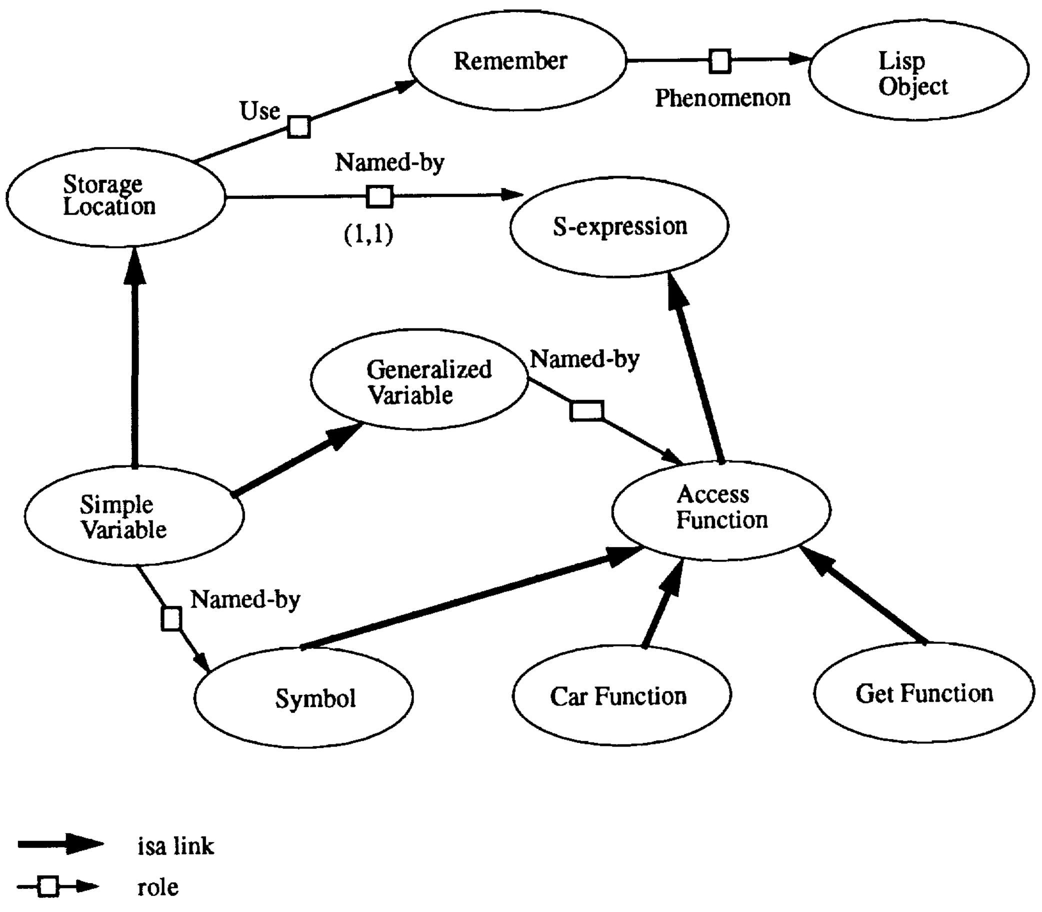

Figure 2: Portion of the domain model in PEA

both of which relate the concept `transformation` to the concept `program-construct`.¶ A portion of this 'semantic net' representation concerned with representing the concepts of simple and generalized variables in Lisp is shown graphically in Figure 2.

In addition to its use in deriving specific actions from general principles, this domain knowledge is employed when responding to users' requests for descriptions of terms used by the system. Novice users often need to ask questions about terminology in order to understand the system's responses and to be able to respond appropriately when answering the system's questions, and experts may want to ask such questions to determine whether the system's use of a term is the same as their own.

¶The representation scheme is actually more complex, allowing value and number restrictions in roles. A complete description of a KL-ONE type knowledge representation scheme is beyond the scope of this paper. See (Brachman and Schmolze, 1985; Moser, 1983) for more information on this topic. We have included here only the elements of the representation needed for our examples.

```
(DEFINE-PLAN enhance-characteristic-of-program
  :GOAL (enhance (obj ((characteristic1 isa characteristic)
                       property-of program))
                 (actor pea-system))
  :INPUT  ((program1 isa program))
  :OUTPUT ((program2 isa program))
  :METHOD (apply-transformation
              (actor pea-system)
              (obj
                (set (transformation
                       that
                         (enhance
                           (obj (characteristic1
                                   property-of program))))))
              (recipient program)))
```

Figure 3: Plan to enhance a characteristic

General Principles in PEA. As mentioned previously, general principles are represented as *plans* consisting of a capability description that describes the goal a plan is capable of achieving and a method, which is a sequence of steps for achieving this goal. The capability description of a plan is written in terms of conceptual structures defined in the domain model. For example, the plan to enhance a characteristic of a program has the following capability description:

```
(enhance (obj ((characteristic1 isa characteristic)
                property-of program))
         (actor pea-system))
```

In this example, `enhance`, `program`, `pea-system` and `characteristic` are all conceptual structures that have been defined in the domain model. Similarly, `property-of` and `actor` are roles defined in the domain model. (These are actually among the abstract concepts defined in the upper model.) The body of this plan embodies the general principle that to enhance a characteristic of a program, the system must apply transformations that enhance that characteristic. This plan is shown in Figure 3.[‖]

Since the method of the plan in Figure 3 will post the subgoal of applying transformations, the system needs ways of doing the application. Once again, those means are described by general principles. For example, PEA includes a general principle stating that to apply a local transformation to a program, the system must iterate over each s-expression in the program attempting to apply the transformation when the lhs of the transformation matches the s-expression under consideration.

[‖]For clarity sake, we have simplified the notation for posting subgoals in this figure.

From general plans to the specific actions of PEA. We now illustrate how specific actions (such as `apply-SETQ-to-SETF` transformation) are obtained from the domain model and the general principles. In EES, the program writer is responsible for deriving the actual expert system from the various knowledge bases. It does so using a goal refinement process. Given a high-level goal that represents the abstract task the expert system is to accomplish, the program writer finds the plan capable of achieving that goal and posts the appropriate subgoals. If no plans are applicable, the program writer tries to reformulate its goal, using the set of reformulations described above.

Consider, for example, the goal of enhancing maintainability. The program writer first applies the plan described in Figure 3, using the fact from the domain model that `maintainability` is a specialization of `characteristic`. This illustrates how the conceptual structures used to express the capability description of a plan allow the system to match specific goals to general plans. In this case, the program writer uses *specialization* and instantiates the general plan to post the subgoal:

```
apply transformations that enhance maintainability
```

The system then tries to achieve this goal and finds that no plans match it. The program writer then tries to reformulate this goal, based on the semantics of the goal itself and the domain knowledge. In this case, the system knows from the domain knowledge that local transformations and distributed transformations together cover the set of transformations (fact (11) on page 8). The program writer can thus perform a covering reformulation to transform its original goal into the two goals:

```
apply local transformations that enhance maintainability
apply distributed transformations that enhance maintainability
```

Because local and distributed transformations together cover transformations, the system knows that if both of these subgoals are achieved then the original goal will also have been achieved.

Let us continue the example by focusing on the first subgoal. The system attempts to find a plan to achieve this goal. Still no plans are found, but the system is able, again based on its domain knowledge (specifically, fact (3) on page 8 above), to reformulate its goal. This time a redescription reformulation is used to create the subgoal:

```
apply local transformations whose right hand side's use is more
general than its left hand side's use.
```

It still cannot find a plan matching this goal, because the system has no plan capable of applying a *set* of transformations. Using an individualization reformulation, however, it is able to split this goal over all the *instances* of 'local transformations whose right hand side's use is more general than its left hand side's use'. It thus now posts the set of subgoals:

```
apply setq-to-setf transformation
apply replaca-to-setf transformation
...
```

The program writer is now able to find plans to achieve these specific subgoals and the specific actions (e.g., 'replace `setq` with `setf`') are generated.

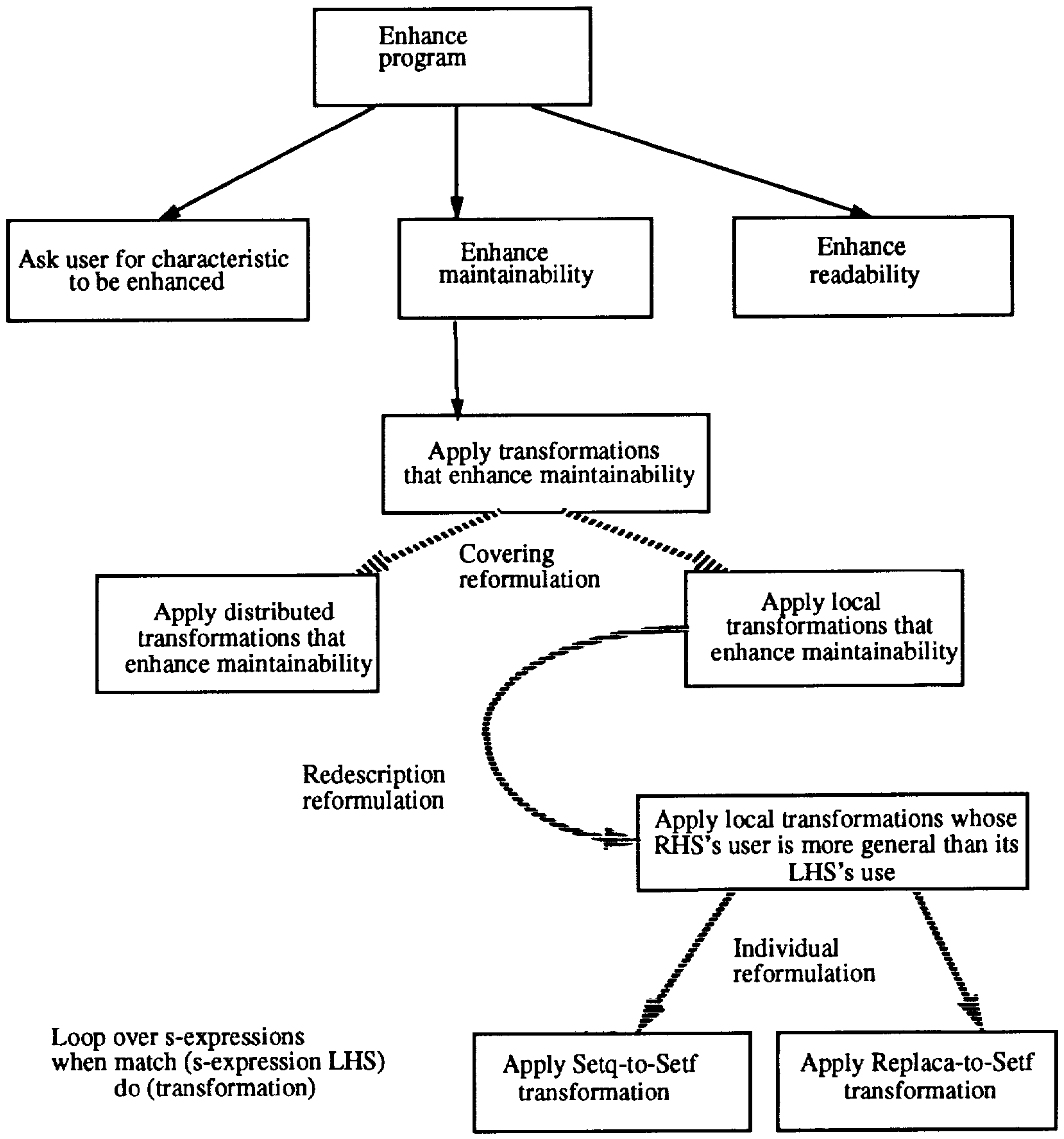
Enhance program
Ask user for characteristic to be enhanced
Enhance maintainability
Enhance readability
Apply transformations that enhance maintainability
Covering reformulation
Apply distributed transformations that enhance maintainability
Apply local transformations that enhance maintainability
Redescription reformulation
Apply local transformations whose RHS's user is more general than its LHS's use
Individual reformulation
Loop over s-expressions when match (s-expression LHS) do (transformation)
Apply Setq-to-Setf transformation
Apply Replaca-to-Setf transformation

Figure 4: The design history

During the refinement process just described, the program writer records all of its steps, including the use of reformulations, keeping track of the type of reformulation that was used at each point, what aspects of the domain model were examined, and how general concepts expressed in a general plan were specialized to form a more specific plan. This gives rise to a *design history*, shown in Figure 4, which indicates how specific actions (the leaves of the tree) were derived from general principles. This design history is what captures the *rationale* behind the specific actions, and, as illustrated below, it is used by the system's generation component to provide explanations of the system's behavior.

3 Better Explanation Production

Explanation systems must produce multisentential texts, including justifications of their actions, descriptions of their problem-solving strategies, and definitions of the terms they use. The template-based explanation generation techniques used in conventional expert systems suffer from a number of limitations. In particular, they are very inflexible and often do not follow standard patterns of discourse experts are used to. As a result, texts are often hard to understand by the end users. Furthermore, they cannot provide elaborations, clarifications, or respond to follow-up questions in the context of an on-going dialogue. Yet, these capabilities are crucial, because studies of naturally occurring advisory interactions show that experts and novices must negotiate the problem to be solved as well as a solution that the novice understands and accepts (Pollack *et al.*, 1982; Moore, 1989b).

Researchers in Natural Language Generation have developed techniques to produce coherent multisentential texts that follow standard patterns of discourse: In particular, they have shown that schemata of rhetorical predicates (McKeown, 1985; McCoy, 1989; Paris, 1988) or rhetorical relations (Hovy, 1991) can be used to capture the structure of coherent multisentential texts. Schemata are script-like entities that encode standard patterns of discourse structure. Associating a schema with a communicative goal allows a system to generate a text that achieves the goal. However, we have found that schemata are insufficient as a discourse model for advisory dialogues. Although they encode standard patterns of discourse structure, schemata do not include a representation of the intended effects of the components of a schema, nor how these intentions are related to one another or to the rhetorical structure of the text. While this may not present a problem for systems that generate one-shot explanations, it is a serious limitation in a system intended to participate in a dialogue where users can, and frequently do, ask follow-up questions (Moore and Paris, 1989a; Moore and Paris, 1992b).

To participate in a dialogue a system must be capable of reasoning about its own previous utterances, as follow-up questions must be interpreted in the context of the ongoing conversation, and the system's previous contributions form part of this context. In particular, an explanation system must represent and reason about the intended effect of individual parts of the text on the hearer, as well as how the parts relate to one another rhetorically. This allows the system to be able to reason about its previous utterances, figuring out what it was trying to say and how it said it, both to interpret and to answer users' follow-up questions.

We have developed a text planner that *explicitly* plans explanations based on the intentions of the speaker at each step (the discourse goals) and that notes the rhetorical relation that holds between each pair of text spans. When a text is produced, the planner records its planning process as well as the user's utterances in a dialogue history, and reasons about it to interpret and to answer users' follow-up questions (Moore and Paris, 1989a; Moore and Paris, 1992b).

The next two sections describe this new text planner and provide an example of the type of dialogue our system is capable of handling.

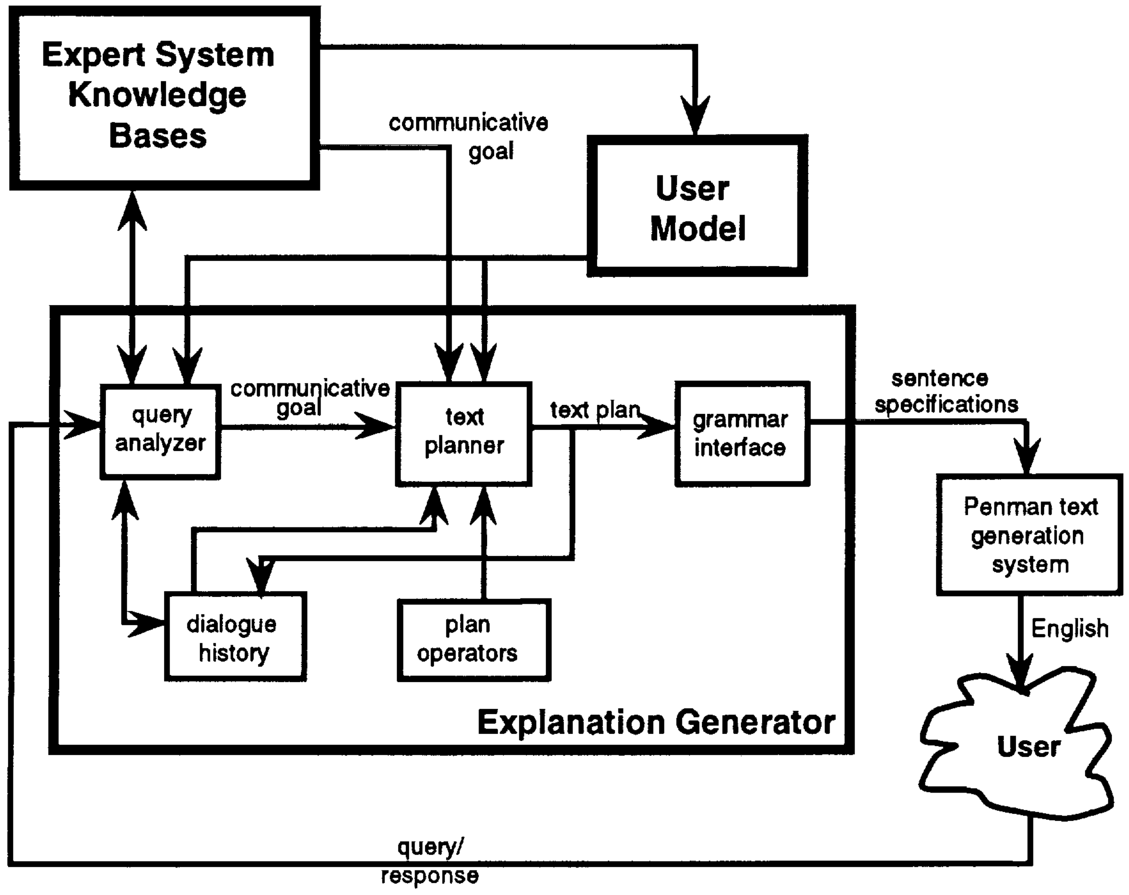

Figure 5: Architecture of Explanation System

3.1 Planning a response and answering a follow-up question

An overview of the explanation generation facility and its relation to other components in the system is shown in Figure 5. The planning process begins when a discourse goal is posted. This may come about in one of two ways. First, in the process of performing a domain task, the expert system may need to communicate with the user, e.g., to ask a question or to recommend that the user perform an action. To do so, it posts a discourse goal to the text planner. Alternatively, the user may request information from the system. In this case, the *query analyzer* interprets the user's question and formulates a discourse goal. Note that a discourse goal such as "achieve the state where hearer knows about concept c" is really an abstract specification of the response to be produced.

When a discourse goal is posted, the text planner searches its library of explanation strategies looking for strategies that can achieve the goal. From these, the planner selects a strategy (an operator) based on several factors, including what the user knows (as indicated in the user model), the conversation that has occurred so far (as indicated in the dialogue history), the relative specificity of the candidate operators, and whether or not each operator requires assumptions to be made. The knowledge of preferences is encoded into a set of selection heuristics – see (Moore, 1989a). The selected operator may in turn posts other subgoals for the planner to refine, and planning continues until primitive operators, or speech acts, are reached.

When a speech act is reached (e.g., `INFORM X, ASK Y`), the system constructs a specification that directs the realization component, Penman (Mann and Matthiessen, 1985; Matthiessen, 1984; Penman Natural Language Generation Group, 1989), to produce the corresponding English utterance. The system builds these specifications based on the type of speech act, its arguments, and the context in which it occurs.** As the planner examines each of the arguments of the speech act, new goals may be posted as a side effect. If one of the arguments is a concept that the user does not know (as indicated in the user model), a satellite subgoal to define this new concept is posted. In addition, if the argument is a complex data structure which cannot be realized directly by English lexical items, the planner will have to "unpack" this complex structure, and, in so doing, will discover additional concepts that must be mentioned. Again, if any of these concepts are not known to the user, subgoals to explain them are posted. Finally, planning is complete when all subgoals in the text plan have been refined to speech acts.

As the system plans explanations, it records its decisions in a text plan, which keeps track of any assumptions made about what the user knows†† as well as alternative strategies that could have been used to achieve discourse goals. This text plan captures the 'design rationale behind the explanation', by recording the discourse goal structure of the text produced, the relationships between parts of the text, and an indication of the role individual sentences and clauses play in the overall explanation structure (Moore and Paris, 1989b).

After a response has been generated, the system awaits feedback from the user. The recorded text plan provides the dialogue context the system needs to 1) disambiguate the user's follow-up questions in context, and 2) plan explanations that elaborate on previous explanations or correct mis-understandings. Due to the central role it plays in supporting dialogue and correcting mis-understandings, capturing the design rationale behind the explanation is as important to providing good explanatory capabilities as capturing the design rationale behind the expert system itself.

3.2 Participating in a Dialogue: An Example

We now illustrate how our system can plan explanations and participate in a dialogue with the user, again using PEA as our sample domain. A sample dialogue with PEA appears in Figure 6. After the user supplies PEA with the program to be enhanced, PEA begins the dialogue by asking what characteristics of the program the user would like to improve. In this example, the user chooses maintainability. PEA then recommends transformations that would enhance the program along this dimension. After each recommendation is made, the user is free to ask questions about the recommendation.

There are several things to note about this dialogue. First, to answer the "why?" in line 4, the system explains its recommendation by examining the design rationale that has

**Bateman and Paris (1989) have also investigated the problem of phrasing utterances for different types of users and situations.

††This is because we cannot assume the user model is complete and correct (Moore and Paris, 1992a).

SYSTEM (1)	What characteristics of the program would you like to enhance?
USER (2)	Maintainability.
	⋮
SYSTEM (3)	You should replace `(setq x 1)` with `(setf x 1)`.
USER (4)	Why?
SYSTEM (5)	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.
USER (6)	Why?
SYSTEM (7)	Transformations that enhance maintainability are defined as transformations whose right-hand-side use is more general than its left-hand-side use. `Setf` can be used to assign a value to any generalized-variable. `Setq` can only be used to assign a value to a simple-variable.
USER (8)	What is a generalized-variable?
SYSTEM (9)	Simple-variables are storage locations named by symbols. This concept can be generalized to a storage location that can be named by any accessor function. These storage locations are called generalized-variables.
USER (10)	Huh?
SYSTEM (11)	For example, the *car* and *cdr* of a cons are generalized-variables, named by the accessor functions `car` and `cdr`.

Figure 6: Sample Dialogue

been recorded (shown in Figure 4). Second, the next "why?" in line 6 is actually ambiguous. Among other things, it could be a question about the super-goal, that is: "Why are you trying to enhance the maintainability of the program?" This is in fact the way that MYCIN (Buchanan and Shortliffe, 1984; Shortliffe, 1976) would interpret this question. However, most people interpret this "why?" as a question about the statement that is currently the focus of attention (i.e., the thing most recently said). That is, most people would interpret the question on line 6 as: "Why is `setq`-to-`setf` a transformation that enhances maintainability?" Our system is able to make this interpretation because it understands the structure of the explanations it has produced – as will be explained below, and because it has heuristics for interpreting such questions based on its dialogue context – see (Moore, 1989a). Note that to produce the answer on line 7, the system makes use of the domain model and of the fact that the goal was reformulated using a redescription. Finally, in line 9,

In Plan Language Notation:

```
EFFECT: (GOAL ?hearer (DO ?hearer ?act))
CONSTRAINTS: (NUCLEUS)
NUCLEUS:
    (RECOMMEND ?speaker ?hearer (DO ?hearer ?act))
SATELLITES:
    (((PERSUADED ?hearer (DO ?hearer ?act)) *optional*)
     ((COMPETENT ?hearer (DO ?hearer ?act)) *optional*))
```

English Paraphrase:

To make the hearer want to do an *act*,
IF this text span is to appear in nucleus position, THEN
  1. Recommend the *act*
AND optionally,
  2. Achieve state where the hearer is persuaded to do the *act*
  3. Achieve state where the hearer is competent to do the *act*

Figure 7: High-level Plan Operator for Recommending an Act

a piece of terminology is defined. When the user indicates (in line 10) that he or she doesn't understand this definition, the system is able to produce a follow-up explanation to clarify things by giving examples (line 11). The system is able to recover from such failures because it understands and can reason about the text it previously produced. In this case, the system needed to know what discourse goal it was trying to achieve and how it achieved this goal (in order to avoid generating the same text). In addition, the system has recovery heuristics that allow it to choose the 'most appropriate' strategy when an alternative explanation is required (Moore, 1989b). This example thus illustrates how the design rationale behind an explanation enables the system to provide responsive explanatory capabilities.

We now explain this dialogue in detail. The dialogue starts when the user indicates a desire to enhance the maintainability of his or her program. To enhance maintainability, the expert system determines that the user should replace `SETQ` with `SETF`. To recommend this transformation, the expert system posts the communicative goal `(GOAL USER (DO USER REPLACE-2))` to the text planner. This goal says that the speaker would like to achieve the state where the hearer has adopted the goal of performing the act `REPLACE-2`, that is the act of replacing `SETQ` with `SETF`.

A plan operator capable of satisfying this goal is shown in Figure 7. As shown in the figure, operators have constraints as to their applicability. Constraints may refer to facts in the system's domain knowledge bases, information in the user model, information in the dialogue history, or information about the evolving text plan. The operator shown in this figure has only one constraint: it is applicable when expanding a goal that is in a nucleus position in the text plan tree so far. This is the case here as this is the first goal posted.

The operator in turn posts three subgoals, separated into the *nucleus* and *satellites*:

- *the nucleus:* this is the subgoal most essential to achievement of the operator's effect. Every operator must contain a nucleus.
- *the satellites:* additional subgoal(s) that may contribute to achieving the effect of the operator. An operator can have zero or more satellites. When present, satellites may be required or optional. Unless otherwise indicated, a satellite is assumed to be required.

In this case, the nucleus of the operator indicate to recommend the action ((RECOMMEND SYSTEM USER (DO USER REPLACE-2))) and, optionally to persuade the user to do the action and make him competent to perform the action. Because the satellites are optional, they are not expanded at this point (Moore and Paris, 1992a), and only the RECOMMEND subgoal is posted. This is a primitive speech act and line [3] of the sample dialogue is generated.

As the user asks "Why?" (line [4]), the query analyzer now posts the goal to persuade the user to perform the action.: (PERSUADED User (Goal User (DO User REPLACE-2))). A plan operator for achieving this goal is shown in Figure 8. When attempting to satisfy the constraints of this operator, the system first checks the constraint (STEP REPLACE-2 ?goal). This constraint states that, in order to use this operator, the system must find a domain goal, ?goal, that REPLACE-2 is a step in achieving. To find such goals, the planner searches the expert system's problem-solving knowledge. In particular, it examines the design history (shown in Figure 4). In this example, the applicable expert system goals, listed in order from most to least specific, are:

```
apply-SETQ-to-SETF-transformation
apply-local-transformations-whose-rhs-use-is-more-general-than-lhs-use
apply-local-transformations-that-enhance-maintainability
apply-transformations-that-enhance-maintainability
enhance-maintainability
enhance-program
```

Thus, six possible bindings for the variable ?goal result from the search for domain goals that REPLACE-2 is a step in achieving.

The second constraint of the current plan operator, (GOAL ?hearer ?goal), is a constraint on the user model stating that ?goal must be a goal of the hearer. Not all of the bindings found so far will satisfy this constraint. Those which do not will not be rejected immediately, however, as we do not assume that the user model is complete. Instead, they will be noted as possible bindings, and each will be marked to indicate that, if this binding is used, an assumption is being made, namely that the binding of ?goal is assumed to be a goal of the user. The selection heuristics can be set to tell the planner to prefer choosing bindings that require no assumptions to be made.

In this example, since the user is employing the system to enhance a program and has indicated a desire to enhance the maintainability of the program, the system infers the user shares the top-level goal of the system (ENHANCE-PROGRAM), as well as the more specific goal ENHANCE-MAINTAINABILITY. Therefore, the two goals that completely satisfy the first two constraints of the operator shown in Figure 8 are ENHANCE-PROGRAM and ENHANCE-MAINTAINABILITY. Finally, the third constraint indicates that only the most specific

In Plan Language Notation:

```
EFFECT: (PERSUADED ?hearer (DO ?hearer ?act))
CONSTRAINTS: (AND (STEP ?act ?goal)
                  (GOAL ?hearer ?goal)
                  (MOST-SPECIFIC ?goal)
                  (CURRENT-FOCUS ?act)
                  (SATELLITE))
NUCLEUS: (((FORALL ?goal
                  (MOTIVATION ?act ?goal)) *required*))
SATELLITES: nil
```

English Paraphrase:

To achieve the state in which the hearer is persuaded to do an *act*,
IF the *act* is a step in achieving some *goal(s)* of the hearer,
  AND the *goal(s)* are the most specific along any refinement path
  AND *act* is the current focus of attention
  AND the planner is expanding a satellite branch of the text plan
THEN motivate the *act* in terms of those *goal(s).*

Figure 8: Plan Operator for Persuading User to Do An Act

goal along any refinement path to the act should be chosen. This constraint encodes the explanation principle that, in order to avoid explaining parts of the reasoning chain that the user is familiar with, when one goal is a subgoal of another, the goal that is lowest in the expert system's refinement structure, i.e., most specific, should be chosen. Note that `ENHANCE-MAINTAINABILITY` is a refinement of `ENHANCE-PROGRAM`. Therefore, `ENHANCE-MAINTAINABILITY` is now the preferred candidate binding for the variable `?goal`.

The nucleus of the chosen plan operator is now posted, resulting in the subgoal `(MOTIVATION REPLACE-2 ENHANCE-2)`, where `ENHANCE-2` is enhance maintainability. The plan operator chosen for achieving this goal is the one shown in Figure 9.[‡‡] This operator motivates the replacement by telling the user which goal it is trying to perform (here `ENHANCE-2`, i.e., to enhance maintainability). To do so, the goals `(INFORM System User ENHANCE-2)` and `(MEANS REPLACE-2 ENHANCE-2)` are posted. Planning continues and the system generates line [5] of the sample dialogue. The corresponding text plan recorded in the dialogue history is shown in Figure 10.

At this point, the user asks "why". This why-question is ambiguous. For example, it could mean "why is setq-to-setf a transformation that enhances maintainability?", or "why

[‡‡]This is only one of the operators available to achieve this goal. In general, there are a number of plans capably of achieving a given discourse goal. This gives the system flexibility and allows it to recover from failure (Moore and Paris, 1991; Moore and Paris, 1992b).

```
EFFECT: (MOTIVATION ?act ?goal)
CONSTRAINTS: (AND (STEP ?act ?goal)
                  (GOAL ?hearer ?goal))
NUCLEUS: (INFORM System ?hearer ?goal)
SATELLITES: (((MEANS ?act ?goal) *required*))
```

Figure 9: Plan Operator for Motivating any Action by Stating the Shared Goals that Act is a Step in Achieving

are you enhancing the maintainability of the program by applying transformations that enhances maintainability?", or still "why are you trying to enhance the maintainability of the program?". Our system is able to disambiguate this question by examining the text plan of the previous explanation as recorded in the dialogue history and employing disambiguation heuristics (Moore, 1989a). In particular, in this case, it uses the local context (or current focus) – as shown in the recorded text plan – to interpret the question correctly as meaning "why is setq-to-setf a transformation that enhances maintainability?". At this point, the system thus posts a goal to explain to the user why setq-to-setf is a transformation that enhances maintainability and produces the explanation on line [7].

Now the user asks a terminological question, line [8], which the system can answer because it has a representation of the domain model. In this case, the system provides a definition of the term, line [9]. This explanation is not understood by the user, as indicated in his or her response, line [10], and the system needs to recover. Note here how the user did not ask a well articulated question. Yet, the system is able to interpret it and provide a new explanation. This is because it can reason about the previous utterance. In this case, the system notes by examining the dialogue history that the previous explanation was a definition, and thus tries another strategy to explain the concept. Here, it chooses examples (line [11]). Note that it would not have been possible to answer this question without the knowledge about what the previous explanation was trying to achieve and how it had attempted to achieve it.

4 Enhancing Maintainability and Evolution of an Expert System

The framework we developed to supports explanation also eases evolution and maintainability of the expert system. Because the knowledge bases are modular, changes can be local to one of the knowledge base (i.e., one can augment the domain model or the problem solving principles independently) and not interfere with the other parts of the system. This is in contrast with conventional expert systems where knowledge tends to be distributed over many rules. As a result, one small change might affect a number of rules, and a common problem is to make sure all the rules affected are identified, and their relationships with other rules examined to make sure the change does not result in a mal-functioning system.

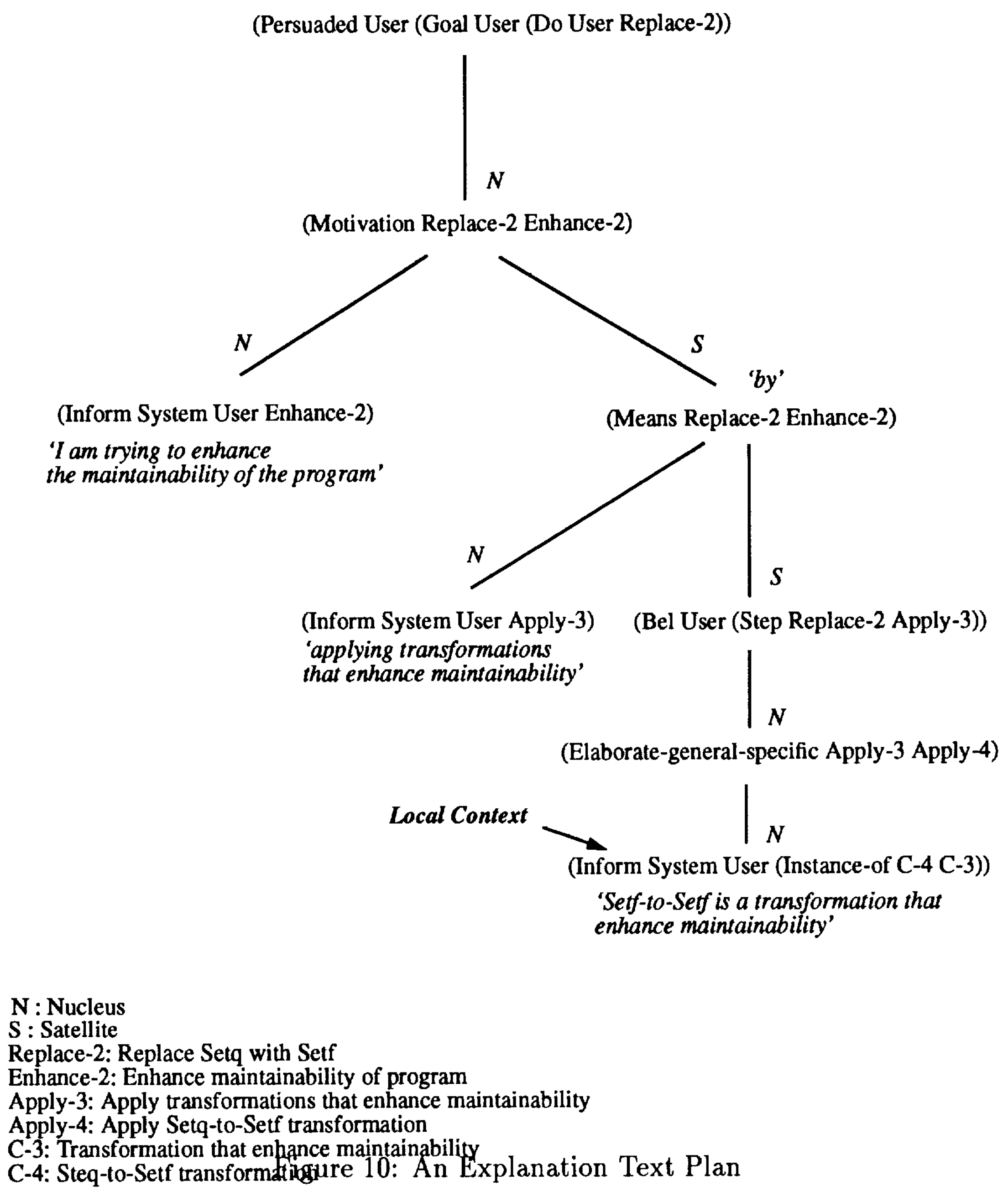
(Persuaded User (Goal User (Do User Replace-2))
N
(Motivation Replace-2 Enhance-2)
N
S
'by'
(Inform System User Enhance-2)
'I am trying to enhance the maintainability of the program'
(Means Replace-2 Enhance-2)
N
S
(Inform System User Apply-3)
'applying transformations that enhance maintainability'
(Bel User (Step Replace-2 Apply-3))
N
(Elaborate-general-specific Apply-3 Apply-4)
Local Context
N
(Inform System User (Instance-of C-4 C-3))
'Setf-to-Setf is a transformation that enhance maintainability'
N : Nucleus
S : Satellite
Replace-2: Replace Setq with Setf
Enhance-2: Enhance maintainability of program
Apply-3: Apply transformations that enhance maintainability
Apply-4: Apply Setq-to-Setf transformation
C-3: Transformation that enhance maintainability
C-4: Steq-to-Setf transformation

Figure 10: An Explanation Text Plan

Furthermore, changed are done at the specification level, in a high-level language, not in the code itself: that is, the domain model, the terminology definitions, or the problem solving plans are changed. The program writer then re-derives the new system automatically. Finally, because of the explanation facility, the user can check for validity of the knowledge contained in the knowledge bases as well as the reasoning of the system, in a language he or she can understand, and thus can identify problems more easily and get feedback on changes.

The resulting architecture thus also achieves our goal or easing evolution and maintainability. Furthermore, it also supports re-usability: as the domain model and the problem solving principles can include some general information about the domain under consideration, much of it can remain constant when a new application in the same domain is created.

5 Accomplishments

Status The EES framework has been implemented, and the examples shown in the paper are actually produced by the system. The system was written in Common Lisp. The Penman text generation system (Mann and Matthiessen, 1985; Matthiessen, 1984; Penman Natural Language Generation Group, 1989) was used to produce actual text from the text plans output by the text planner.

Accomplishments

- The text planner developed as part of this project is the first system to support dialogue-based explanations, and is currently one of the most sophisticated text planner available. It is now used as a point of reference in the computational linguistic research community). Our work on the text planner also influence development of the Penman project (Mann and Matthiessen, 1985; Matthiessen, 1984), one of the major project in sentence generation.
- The EES framework is one of the most sophisticated framework for knowledge-based systems today, and has had a lot of influence on other important research efforts:
  - It is the first framework for knowledge-based systems that integrates problem solving and explanation; Because explanations are always generated from the underlying knowledge bases, they are always guaranteed to be accurate and reflect any changes to the underlying system. Furthermore, the range of explanations EES can offer is larger than that of conventional systems;
  - By using a powerful knowledge representation system (Loom), EES employs a faster and simplified problem solver by taking advantage of Loom's powerful inferencing mechanisms.
  - EES was one of the first systems to explore and extend model-based programming in Loom (MacGregor, 1988; MacGregor, 1991);
  - Its referential capabilities and its language for plans have been later used by the Loom project;
  - The idea of reformulation is now being used by other systems (e.g., SIMS) (Arens and Knoblock, 1992; Arens *et al.*, 1993);
  - EES is the first system to order its library of problem solving plans in a generalization hierarchy and use a classifier (that of Loom) to do the plan-goal match. This technique results in an efficient semantic-based plan-goal matching algorithm.
- We have used the EES framework to construct several demonstration-size expert systems:
  - PEA, the system used here to illustrate our framework;

  - a prototype for an expert system to diagnose local area networks, built in collaboration with DEC;
  - a prototype for an expert system to assist cardiologists in the intensive-care unit (in collaboration with Cedars Sinai Hospital in Los Angeles).

- The EES framework and its explanation module has received a lot of outside interest and has been used by other institutions:
  - As mentioned above, we had outside collaborations with DEC and Cedars Sinai;
  - Robert Kass, a PhD student at the University of Pennsylvania used the EES framework to implement an expert system to test his ideas for his PhD work (Kass, 1988; Kass, 1991);
  - The EES text planner is used in a few institutions as a generation planning system:
    - Norbert Reithinger at the University of Saarbruecken used it as a basis for his incremental generation system (Reithinger, 1991)
    - The WIP project at the German National Research Institute (DFKI) used it the basis of their multi-modal presentational planner (Wahlster *et al.*, 1990; Wahlster *et al.*, 1991b; Wahlster *et al.*, 1991a); This project is one of the two major multi-modal projects in the world today.
    - Dr. Johanna Moore at the Computer Science Department of the University of Pittsburgh and at LRDC uses it for the interaction module of several tutorial systems (Carenini and Moore, 1993).

6 Publications this contract enabled

[1] Kathleen R. McKeown and William R. Swartout. Language generation and explanation. In *Annual Reviews in Computer Science*, 1987.

[2] William R. Swartout and Steve W. Smoliar. On making expert systems more like experts. *Expert Systems*, 4(3), August 1987.

[3] William R. Swartout and Stephen W. Smoliar. Explaining the link between causal reasoning and expert behavior. In W. W. Stead, editor, *Proceedings: The Eleventh Annual Symposium on Computer Applications in Medical Care*, pages 37–42, Washington, DC, November 1987. IEEE.

[4] Johanna D. Moore and William R. Swartout. Explanation in Expert Systems: A Survey. Technical Report ISI/RR-88-228, USC/Information Sciences Institute, 1988.

[5] Cécile L. Paris. Planning a text: can we and how should we modularize this process?, August 1988. Workshop on text planning and Natural Language Generation, Sponsored by AAAI; Organized by Eduard H. Hovy, Doug Appelt, David McDonald, and Sheryl Young.

[6] Cécile Paris, Michael Wick, William Thompson, and William Swartout. The Line of Reasoning vs The Line of Explanation. In Cécile Paris, Michael Wick, William Thompson, and William Swartout, editors, *Proceedings of the AAAI'88 Workshop on Explanation*, St Paul, Minnesota, August 1988. American Association for Artificial Intelligence.

[7] Johanna D. Moore and Cécile L. Paris. Constructing coherent texts using rhetorical relations. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society.* Cognitive Science Society, August 1988.

[8] William Swartout, Henrik Nordin, Cécile Paris, and Stephen Smoliar. Toward a rapid prototyping environment for expert systems. In *Proceedings of the 13th German Workshop on Artificial Intelligence*, pages 438–454, 1989.

[9] William R. Swartout and Stephen W. Smoliar. Explanation: A source of guidance for knowledge representation. In K. Morik, editor, *Knowledge Representation and Organization in Machine Learning*, volume 347 of *Lecture Notes in Artificial Intelligence*, pages 1–16. Springer-Verlag, Berlin, West Germany, 1989.

[10] John A. Bateman and Cécile L. Paris. Phrasing a text in terms the user can understand. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989.

[11] Johanna D. Moore. *A reactive approach to explanation in expert and advice-giving systems.* PhD thesis, University of California, Los Angeles, 1989.

[12] 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.

[13] J. D. Moore and W. D. Swartout. A reactive approach to explanation. In *Proceedings of the Eleventh International Conference on Artificial Intelligence*, pages 1505–1510, Detroit, MI, August 1989. IJCAI.

[14] Johanna D. Moore and Cécile L. Paris. Planning text for advisory dialogues. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 203–211. Association for Computational Linguistics, June 1989.

[15] Cécile L. Paris, William R. Swartout, and William C. Mann, editors. *Natural Language Generation in Artificial Intelligence and Computational Linguistics.* Kluwer Academic Publishers, Boston/Dordrecht/London, 1990.

[16] Vibhu O. Mittal and Cécile L. Paris. On the use of Analogies in Explanations in EES. In Nick Filer, editor, *Proceedings of the 5th Workshop on Explanation*, Manchester, UK, April 1990.

[17] Vibhu O. Mittal and Cécile L. Paris. Analogical Explanations in the EES Framework. In Johanna D. Moore and Michael R. Wick, editors, *Proceedings of the AAAI'90 Workshop on Explanation*, pages 162 – 172, Boston, MA, August 1990. American Association for Artificial Intelligence.

[18] Johanna D. Moore and William R. Swartout. Pointing: A way towards explanation dialogue. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 457 – 464, Boston, Mass, August 1990 1990.

[19] Vibhu O. Mittal and Cécile L. Paris. Analogical Explanations. In *Proceedings of the Third Conference on Knowledge Based Computer Systems – KBCS-90*, pages 17–26, New Dehli, India, December 13 – 15 1990. Center for Development of Advanced Computers. Also published as a chapter in *Frontiers in Knowledge-Based Computing*, edited by V. P. Bhatkar and K. M. Rege, Narosa Publishing House, New Delhi, India.

[20] Cécile L. Paris. Tailoring as a prerequesite for effective human-computer communication. In *Proceedings of the 1990 AAAI Symposium on Knowledge-Based Human Computer Communication*, Stanford, California, March 1990. American Association for Artificial Intelligence.

[21] Johanna D. Moore and Cécile L. Paris. Requirements for an Expert System Explanation Facility. *Computational Intelligence*, 7(4), 1991.

[22] Cecile L. Paris and Elisabeth A. Maier. Knowledge resources or decisions? In *IJCAI-91 Workshop on Decision Making throughout the Generation Process*, pages 11–17, 1991.

[23] Vibhu O. Mittal, Cécile Paris, Ramesh Patil, and Bill Swartout. Organizing plan libraries in subsumption hierarchies: Specificity based plan selection. Technical Report, USC/ISI, December 1991.

[24] Cécile L. Paris. Generation and explanation: Building an explanation facility for the explainable expert systems framework. In Cécile L. Paris, William R. Swartout, and

William C. Mann, editors, *Natural language generation in artificial intelligence and computational linguistics.* Kluwer Academic Publishers, July 1991.

[25] Johanna D. Moore and William R. Swartout. A reactive approach to explanation. In Cécile L. Paris, William R. Swartout, and William C. Mann, editors, *Natural language generation in artificial intelligence and computational linguistics.* Kluwer Academic Publishers, July 1991.

[26] Johanna D. Moore and Cécile L. Paris. The EES Explanation Facility: its Tasks and its Architecture. In *Proceedings of the AAAI'91 Workshop on Comparative Analysis of Explanation Planning Architectures*, Anaheim, Ca, July 1991. American Association for Artificial Intelligence.

[27] B. Chandrasekaran and William Swartout. Explanations in Knowledge Systems: The Role of Explicit Representation of Design Knowledge. *IEEE Expert*, 6(3):47–50, June 1991.

[28] William R. Swartout, Cécile L. Paris, and Johanna D. Moore. Design for Explainable Expert Systems. *IEEE Expert*, 6(3):58–64, June 1991.

[29] Cécile L. Paris. The role of the user's domain knowledge in generation. *Computational Intelligence*, 7 (2):71–93, May 1991. This is an extended version of a paper which appears in the Proceedings of the International Computer Science Conference '88, sponsored by IEEE.

[30] Johanna D. Moore and Cécile L. Paris. Discourse structure for explanatory dialogues. In *Proceedings of the AAAI-91 Fall Symposium on Discourse Structure in Natural Language Understanding and Generation*, Asilomar, California, November 1991. American Association for Artificial Intelligence.

[31] Johanna D. Moore and Cécile L. Paris. Exploiting User Feedback to Compensate for the Unreliability of User Models. *User Model and User Adapted Interaction Journal*, 2(4), 1992. Authors in alphabetical order.

[32] William R. Swartout and Cécile L. Paris. Using Design Rationale for Explanation and Acquisition in Expert Systems. In *Proceedings of the AAAI'92 Workshop on Design Rationale*, Palo Alto, Ca, July 1992. American Association for Artificial Intelligence.

[33] Johanna D. Moore and Cécile L. Paris. Planning Text for Advisory Dialogues: Capturing Intentional, and Rhetorical Information. Submitted for Publication in the Journal for Computational Linguistics, 1993. Authors in alphabetical order.

[34] Cécile L. Paris. *The Use of Explicit User Models in Text Generation.* Frances Pinter, London, England, 1993.

References

(Arens and Knoblock, 1992) Yigal Arens and Craig A. Knoblock. Planning and reformulating queries for semantically-modeled multidabase systems. In *Proceedings of the First International Conference on Information and Knowledge Management*, Baltimore, MD, 1992.

(Arens *et al.*, 1993) Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems*, 1993. In press.

(Bateman and Paris, 1989) John A. Bateman and Cécile L. Paris. Phrasing a Text in Terms the User Can Understand. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1511–1517, Detroit, Michigan, August 20-25 1989.

(Bateman *et al.*, 1989) John Bateman, Robert Kasper, Johanna D. Moore, and Richard Whitney. The Penman Upper Model. Technical report, USC/Information Sciences Institute, 1989.

(Brachman and Schmolze, 1985) Ronald J. Brachman and James G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9:171–216, 1985.

(Buchanan and Shortliffe, 1984) Bruce G. Buchanan and Edward H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley Publishing Company, 1984.

(Carenini and Moore, 1993) Giuseppe Carenini and Johanna D. Moore. Generating explanations in context. In *Proceedings of the International Workshop on Intelligent User Interfaces*, Orlando, Florida, January 1993.

(Hovy, 1991) Eduard H. Hovy. Approaches to the Planning of Coherent Text. In Cécile L. Paris, William R. Swartout, and William C. Mann, editors, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, pages 83–102. Kluwer Academic Publishers, Boston, 1991.

(Kass, 1988) Robert Kass. *Acquiring a Model of the User's Beliefs From a Cooperative Advisory Dialogue.* PhD thesis, University of Pennsylvania, 1988. Published by University of Pennsylvania as Technical Report MS-CIS-88-104.

(Kass, 1991) Robert Kass. Building a User Model. *User Model and User Adapted Interaction*, 1(3):203–258, 1991.

(MacGregor, 1988) Robert MacGregor. A Deductive Pattern Matcher. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, Minnesota, August 1988.

(MacGregor, 1991) Robert M. MacGregor. Using a description classifier to enhance deductive inference. In *Proceedings of the Seventh IEEE Conference on Artificial Intelligence Applications.* IEEE, 1991.

(Mann and Matthiessen, 1985) William C. Mann and Christian M.I.M. Matthiessen. Nigel: A Systemic Grammar for Text Generation. In R. Benson and J. Greaves, editors, *Systemic Perspectives on Discourse: Selected Papers Papers from the Ninth International Systemics Workshop*. Ablex, London, 1985. Also available as USC/ISI Research Report RR-83-105.

(Matthiessen, 1984) C.M.I.M. Matthiessen. Systemic Grammar in Computation: The Nigel Case. In *Proceedings of the First Conference of the European Association for Computational Linguistics*, Pisa, Italy, 1984. European Association for Computational Linguistics. Also available as USC/ISI Research Report RR-84-121, 1984.

(McCoy, 1989) Kathleen F. McCoy. Generating Context Sensitive Responses to Object-Related Misconceptions. *Artificial Intelligence*, 41(2):157–195, 1989.

(McKeown, 1985) Kathleen R. McKeown. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text.* Cambridge University Press, Cambridge, England, 1985.

(Moore and Paris, 1989a) 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*, pages 203–211, Vancouver, B.C., Canada, June 26-29 1989.

(Moore and Paris, 1989b) 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, British Columbia, June 1989.

(Moore and Paris, 1991) Johanna D. Moore and Cécile L. Paris. Requirements for an Expert System Explanation Facility. *Computational Intelligence*, 7(4), 1991.

(Moore and Paris, 1992a) Johanna D. Moore and Cécile L. Paris. Exploiting user feedback to compensate for the Unreliability of User Models. *User Modeling and User-Adapted Interaction*, 2(4), 1992.

(Moore and Paris, 1992b) Johanna D. Moore and Cécile L. Paris. Planning Text for Advisory Dialogues: Capturing Intentional, Rhetorical and Attentional Information, 1992. Technical Report from the University of Pittsburgh, Department of Computer Science (Number 92–22) and USC/ISI; Submitted for publication.

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

(Moore, 1989b) Johanna D. Moore. Responding to "Huh?": Answering Vaguely Articulated Follow-Up Questions. In *Proceedings of the 1989 Conference on Human Factors in Computing Systems*, pages 91–96, Austin, Texas, April 30 – May 4 1989.

(Moser, 1983) M.G. Moser. An Overview of NIKL, the New Implementation of KL-ONE. In *Research in Natural Language Understanding.* Bolt, Beranek, and Newman, Inc., Cambridge, MA, 1983. BBN Technical Report 5421.

(Neches *et al.*, 1985) R. Neches, W. R. Swartout, and J. D. Moore. Enhanced maintenance and explanation of expert systems through explicit models of their development. *IEEE Transactions on Software Engineering*, SE-11(11):1337–1351, November 1985.

(Paris, 1988) Cécile L. Paris. Tailoring object descriptions to the user's level of expertise. *Computational Linguistics*, 14(3):64–78, September 1988.

(Penman Natural Language Generation Group, 1989) Penman Natural Language Generation Group. The Penman user guide, 1989. Available from USC/Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, California 90292-6695.

(Pollack *et al.*, 1982) Martha E. Pollack, Julia Hirschberg, and Bonnie Lynn Webber. User Participation in the Reasoning Processes of Expert Systems. In *Proceedings of the Second National Conference on Artificial Intelligence*, Pittsburgh, Pennsylvania, August 18-20 1982. A longer version of this paper is available as a Technical Report from the University of Pennsylvania, Report Number CIS-82-10.

(Reithinger, 1991) Norbert Reithinger. *Eine parallele Architektur zur inkrementellen Generierung multimodaler Dialogbeiträge.* PhD thesis, Technischen Fakultät der Universität des Saarlandes, Saarbrücken, Germany, 1991.

(Shortliffe, 1976) Edward H. Shortliffe. *Computer Based Medical Consultations: MYCIN.* Elsevier North Holland Inc., 1976.

(Wahlster *et al.*, 1990) Wolfgang Wahlster, Elisabeth André, S. Bandyopadhyay, Winfried Graf, and Thomas Rist. WIP: The Coordinated Generation of Multimodal Presentations from a Common Representation. In *Proc. Int. Workshop on Computational Theories of Communication and their Applications: Problems and Prospects*, Trentino, Italy, November 1990.

(Wahlster *et al.*, 1991a) Wolfgang Wahlster, Elisabeth André, S. Bandyopadhyay, Winfried Graf, and Thomas Rist. The Coordinated Generation of Multimodal Presentations from a Common Representation. Technical report, DFKI (University of Saarbrucken) Technical Report, 1991.

(Wahlster *et al.*, 1991b) Wolfgang Wahlster, Elisabeth André, Winfried Graf, and Thomas Rist. Designing Illustrated Texts: How Language Production is Influenced by Graphics Generation. In *Proc. European Chapter of the Assoc. for Computational Linguistics*, pages 8–14, Berlin, April 1991.