16

Explaining and Justifying Expert Consulting Programs

William R. Swartout

As was mentioned in the introduction to Chapter 14, the ABEL work of Patil, Szolovits, and Schwartz uses a patient-specific model inspired in part by an earlier project from the M.I.T./Tufts group known as the Digitalis Therapy Advisor (Gorry et al., 1978). The Digitalis Therapy Advisor reached an excellent level of performance regarding the appropriate adjustment of digitalis dosing in cardiac patients, and it also provided a rich environment for related work such as the XPLAIN research of William Swartout described in this chapter. Swartout focused on the construction of an explanation capability for the Digitalis Therapy Advisor; the resulting programs have in turn influenced subsequent AI research on explanation.

Traditional methods for generating explanations by a decision-making program have involved displaying "canned" text or converting to English the code of the program (or traces of the execution of that code). While such methods can provide superficially useful explanations of what the program does or did, they generally cannot tell why *what the system is doing is a reasonable thing to be doing. The problem is that the knowledge required to provide these justifications is used (by the programmer) only when the program is being written and does not appear in the code itself.*

Swartout's XPLAIN system, on the other hand, uses an automatic programmer to generate the consulting program by refinement from abstract goals. The automatic programmer uses a domain model, consisting of facts about the application domain, and a set of domain principles that drive the

From the *Proceedings of the Seventh International Joint Conference on Artificial Intelligence,* vol. 2, pp. 815–823, (1981). Used by permission of International Joint Conferences on Artificial Intelligence, Inc.; copies of the *Proceedings* are available from William Kaufmann, Inc., 95 First Street, Los Altos, CA 94022.

refinement process forward. Examining the refinement structure created by the automatic programmer makes possible justifications of the code. This chapter describes XPLAIN and outlines additional advantages this approach has for explanation.

The significance of Swartout's work is not just its use of a system design technique that makes explanation possible. His work reveals how principles *(here, domain strategies by which specific treatment methods are applied) are part of explanation. It is useful to supply not just an "audit trail" of what a problem solver did (on perhaps different levels of detail) but an explanation of* why *the procedure is valid. Swartout's point is that a more powerful knowledge representation is the secret to better explanation, not just better natural language facilities. The same observation holds for tutoring systems (see Chapters 11 and 15).*

16.1 Introduction

To be acceptable, expert programs must be able to explain what they do and justify their actions in terms understandable to the user. Expert programs usually have some heuristic basis. While these heuristics may provide good performance for most cases, there may be unusual cases where they produce erroneous results or where the rationale for using them is faulty. If a user is suspicious of the advice he or she receives, the user should be able to ask for a description of the methods employed and the reasons for employing them. In addition, the scope of expert systems, like that of human experts, is often quite narrow. An explanation facility can help a user discover when a system is being pushed beyond the bounds of its expertise.

In the area of medical consultant programs,[1] the need for explanation is particularly acute. In designing a consultant program, we must consider what sorts of capabilities we are trying to provide for the physician user. If we consider the interaction between a physician and a human consultant, we realize that it is not just a simple one-way exchange where the physician provides data and the consultant provides an answer in the form of a prescription or diagnosis. Rather, there is typically a lively dialogue between the two. The physician may question whether some factor was considered or what effect a particular finding had on the final outcome. Viewed in this light, we realize that a computer program that only collects data and provides a final answer will not be found acceptable by most

[1]Some medical consultant programs include MYCIN, a program that aids physicians with antimicrobial therapy (Shortliffe, 1976); INTERNIST, a program that makes diagnoses in internal medicine (Pople, 1977); and PIP, a program that makes diagnoses primarily in the area of renal disease (see Chapter 6).

physicians. In addition to providing diagnoses or prescriptions, a consultant program must be able to explain what it is doing and justify why it is doing it.

Researchers have recognized this, and many proposals for new expert systems have at least mentioned the need for explanation. Some systems have actually provided an explanatory facility. Yet existing approaches to explanation fail in some important ways. This paper will document these failings and describe an approach toward their solution.

While we have concentrated on the problem of providing explanations to medical personnel, we do not feel that the need for explanation is limited to medicine or that the techniques we have developed for explanation and justification are limited to medical applications. Medical programs provide a good test-bed for the general problem of explaining a consulting program to the audience it is intended to serve.

The next section will describe the Digitalis Therapy Advisor, a program we have chosen as a test-bed for our ideas about explanation, and some of the medical aspects of digitalis therapy. After that, we will describe some of the problems with previous explanation systems and the approach we have taken to overcome those problems.

16.2 Digitalis Therapy and the Digitalis Therapy Advisor

The digitalis glycosides are a group of drugs that were originally derived from the foxglove, a common flowering plant. Their principal effect is to strengthen and stabilize the heartbeat. In current practice, digitalis is prescribed chiefly to patients who show signs of congestive heart failure (CHF) and/or conduction disturbances of the heart. Congestive heart failure refers to the inability of the heart to provide the body with an adequate blood flow. This condition causes fluid to accumulate in the lungs and outer extremities, and it is this aspect that gives rise to the term *congestive*. Digitalis is useful in treating this condition because it increases the contractility of the heart, making it a more effective pump. A conduction disturbance appears as an arrhythmia, which is an unsteady or abnormally paced heartbeat. Digitalis tends to slow the conduction of electrical impulses through the conduction system of the heart, and thus steady certain types of arrhythmias. Due to the positive effect that digitalis has on the heart, it is one of the most commonly used drugs in the United States.

Like many other drugs, digitalis can also be a poison if too much is administered. For a variety of reasons, including a small therapeutic window, subtle signs of toxicity, and high interpatient variability, digitalis is difficult to administer. One complication the physician must deal with is the possibility that a patient may be more sensitive to the drug (for whatever reason) than the average patient. If a physician knows those factors that make a patient more sensitive, he or she can reduce the likelihood of overdosing (or underdosing) the patient by adjusting the dose depending on whether or not the sensitizing factors are observed.

Over the years, a number of factors that increase the automaticity of the heart[2] have been identified. These include a low level of serum potassium (hypokalemia), a high level of serum calcium (hypercalcemia), damage to the heart muscle (cardiomyopathy), and a recent myocardial infarction, among others. When these exist in conjunction with digitalis administration, the automaticity can be increased substantially. This chapter will describe in detail how those fragments of the Digitalis Therapy Advisor that adjust for the first two sensitivities are justified and explained.

16.2.1 The Digitalis Therapy Advisor Test-Bed

A few years ago, the Digitalis Therapy Advisor was developed at M.I.T. by Pauker, Silverman, and Gorry (Silverman, 1975; Gorry et al., 1978). This program was later revised and given a preliminary explanatory capability (Swartout, 1977). The limitations of these explanations (and of those produced by similar techniques) will be discussed below. This program differed from earlier digitalis advisors (Peck et al., 1973; Jelliffe et al., 1970; Jelliffe et al., 1972; Sheiner et al., 1972) in two important respects. First, when formulating dosage schedules, it anticipated possible toxicity by taking into account the factors that increased digitalis sensitivity and reduced the dose when those factors were present. Second, the program made assessments of the toxic and therapeutic effects that actually occurred in the patient after receiving digitalis to formulate subsequent dosage recommendations. This program worked in an interactive fashion. The program asked the physician for data about the patient and produced recommendations after that data was entered. When the dose of digitalis was being adjusted, the physician was asked to consult with the program again to assess the patient's response. This is the program we used as a test-bed for our work in explanation and justification. In the remainder of the paper, we will refer to this program as the old Digitalis Advisor.

---

[2]In the normal heart, there is a place in the left atrium called the sino-atrial (SA) node, which sets the pace for the heart. Under the right circumstances, other parts of the heart can take over the pace-setting function. Sometimes this can be life-saving, if, for example, the SA node is damaged. But at other times it can be life-threatening, since several pacemakers operating simultaneously tend to increase the likelihood of setting up a dangerous arrhythmia. When we say that digitalis increases the automaticity of the heart, we mean that digitalis increases the tendency of other parts of the heart to take over the pace-setting function from the SA node.

16.3 Kinds of Questions That Arise Concerning the Advisor

In the spring of 1979, we conducted a series of informal trials in an attempt to discover what kinds of questions occurred to medical personnel as they ran the old Digitalis Advisor. In this trial, medical students and fellows were asked to run the program and ask questions (verbally) as they occurred to them. The author attempted to answer these questions. The interactions were tape-recorded and later transcribed.

No formal analysis of the data was attempted, but examination of the transcripts did provide an indication of the types of questions that might arise while running a consulting program. These included:

1. Questions about the methods the program employed:

> *User:* "How do you calculate your body store goal? That's a little lower than I anticipated."

This sort of question could be answered by the explanation routines of the old Digitalis Advisor. It can also be answered by the system presented in this paper.

2. Justifications of the program's actions:

> *User (peruses recommendations):* "Why do we want to make a temporary reduction?"
>
> *Author:* "We're anticipating surgery coming up and surgery, even noncardiac surgery, can cause increased sensitivity to digitalis, so it wants to temporarily reduce the level of digitalis."

This is exactly the sort of question we are concentrating on in this paper. It cannot be answered by the explanation routines of the old Digitalis Advisor.

3. Questions involving confusion about the meaning of terms:

> *User (in response to the question IS THE RENAL FUNCTION STABLE?):* "Now this question . . . I'm not really sure . . . 'renal function stable' does it mean stable abnormally or . . . because I mean, the patient's renal function is not normal but it's stable at the present time."
>
> *Author:* "That's what it means."

This paper will not address this last type of question.

16.4 Previous Approaches to Explanation

A number of different approaches have been taken to attempt to provide programs with an explanatory capability. The major approaches include (1) using previously prepared text to provide explanations and (2) producing explanations directly from the computer code and traces of its execution.

The simplest way to get a computer to answer questions about what it is doing is to anticipate the questions and store the answers as English text. Only the text that has been stored can be displayed. This is called *canned text,* and explanations produced by displaying canned text are called *canned explanations.* The simplest sorts of canned explanations are error messages. For example, a medical program designed to treat adults might print the following message if someone tried to use it to treat an infant:

THE PATIENT IS TOO YOUNG TO BE TREATED BY THIS PROGRAM.

It is relatively easy to get a small program to provide English explanations of its activity using this canned text approach. After the program is written, canned text is associated with each part of the program explaining what that part of the program is doing. When the user wants to know what is going on, the computer merely displays the text associated with what it is doing at the moment.

There are several problems with the canned text approach to explanation. The fact that the program code and the text strings that explain that code can be changed independently makes it difficult to guarantee consistency between what the program does and what it claims to do. Another problem with the canned text approach is that all questions and answers must be anticipated in advance and the programmer must provide answers for all the questions that the user might ask. For large systems, that is a nearly impossible task. Finally, the system has no conceptual model of what it is saying. That is, to the computer, one text string looks much like any other, regardless of the content of that string. Thus it is difficult to use this approach if we want our system to provide more advanced sorts of explanations, such as suggesting analogies or giving explanations at different levels of abstraction.

Another approach to explanation is to produce explanations directly from the program (Davis, 1976; Shortliffe, 1976; Swartout, 1977; Winograd, 1971). That is, the explanation routines examine the program that is executed. Then by performing relatively simple transformations on the code, these explanation routines can produce explanations of how the system does things. For example, the old Digitalis Advisor could examine the code it used to check for increased digitalis sensitivity caused by increased serum calcium and produce an explanation of how that code worked (as shown in Figure 16-1).

```
TO CHECK SENSITIVITY DUE TO CALCIUM I DO THE FOLLOWING STEPS:

1. I DO ONE OF THE FOLLOWING:
    1.1 IF EITHER THE LEVEL OF SERUM CALCIUM IS GREATER THAN 10
    OR INTRAVENOUS CALCIUM IS GIVEN THEN I DO THE FOLLOWING SUBSTEPS:
        1.1.1 I SET THE FACTOR OF REDUCTION DUE TO HYPERCALCEMIA TO 0.75.
        1.1.2 I ADD HYPERCALCEMIA TO THE REASONS OF REDUCTION.
    1.2 OTHERWISE, I REMOVE HYPERCALCEMIA FROM THE REASONS OF
    REDUCTION AND SET THE FACTOR OF REDUCTION DUE TO HYPERCALCEMIA TO 1.00.
```

FIGURE 16-1 Explanation of how the old Digitalis Advisor checked hypercalcemia in general.

The old Digitalis Advisor, like most similar systems, also maintained an execution trace. The trace could be examined by the explanation routines to tell what the system did for a particular patient. Figure 16-2 describes how the system checked for myxedema. The system also had a limited ability to explain why it was asking the user a question. Figure 16-3 shows the system's response when the user wants to know why he or she is being asked about serum calcium.

```
I CHECKED SENSITIVITY DUE TO THYROID-FUNCTION BY EXECUTING THE
FOLLOWING STEPS:

1. I ASKED THE USER THE STATUS OF MYXEDEMA. THE USER RESPONDED THAT
THE STATUS OF MYXEDEMA WAS PRESENT.

2. SINCE THE STATUS OF MYXEDEMA WAS PRESENT I DID THE FOLLOWING:

    2.1 I ADDED MYXEDEMA TO THE PRESENT AND CORRECTABLE CONDITIONS.
    THE PRESENT AND CORRECTABLE CONDITIONS THEN BECAME MYXEDEMA.

    2.2 I REMOVED MYXEDEMA FROM THE DEGRADABLE CONDITIONS. THE
    DEGRADABLE CONDITIONS THEN BECAME HYPOKALEMIA, HYPOXEMIA,
    CARDIOMYOPATHIES-MI, AND POTENTIAL POTASSIUM LOSS DUE TO DIURETICS.

    2.3 I SET THE FACTOR OF REDUCTION DUE TO MYXEDEMA TO 0.67.
    THE FACTOR OF REDUCTION DUE TO MYXEDEMA WAS PREVIOUSLY UNDETERMINED.

    2.4 I ADDED MYXEDEMA TO THE REASONS OF REDUCTION.
    THE REASONS OF REDUCTION THEN BECAME MYXEDEMA.
```

FIGURE 16-2 Explaining how thyroid function was checked.

WHAT IS THE LEVEL OF SERUM CALCIUM?

ENTER NUMERICAL VALUE ====> why? *(The user types "why?" indicating that the user questions why he or she is being asked to supply the value of serum calcium.)*

MY TOP GOAL IS TO BEGIN THERAPY. ONE STEP IN DOING THAT IS TO CHECK SENSITIVITIES. I AM NOW TRYING TO CHECK SENSITIVITY DUE TO CALCIUM.

FIGURE 16-3 A *limited* explanation telling why a question is asked.

Since the explanation routines only perform simple transformations on the program code, the quality of the explanations produced in this manner depends to a great degree on how the system code is written. In particular, the basic structure of the program is not altered significantly, and the names of variables in the explanation are basically the same as those in the program. If the explanations are to be understandable, the expert system must be written so that its structure is easily understood by anyone familiar with its domain of expertise, and the variable and procedure names used in the program must represent concepts that are meaningful to the user.

This method of producing explanations has some advantages. It is relatively simple. If the right way of structuring the problem can be found, it does not impose too great a burden on the programmer; since the explanations reflect the code directly, consistency between explanation and code is assured.

Despite these advantages, there are some serious problems with this technique. It may be difficult or impossible to structure the program so that the user can easily understand it. The fact that every operation performed by the computer must be explicitly spelled out sometimes forces the programmer to program operations that a physician would perform without thinking. That problem is illustrated in Figure 16-2. Steps 2.1, 2.2, and 2.4 are somewhat mystifying. In fact, these steps are needed by the system so that it can record what sensitivities the patient had that made him or her more likely to develop digitalis toxicity. These steps are involved more with record keeping than with medical reasoning, but they must appear in the code so that the computer will remember why it made a reduction. Since they appear in the code, they are described by the explanation routines, although they are more likely to confuse than enlighten a physician user. An additional problem is that it is difficult to get an overview of what is really going on here. While the system is explicit about record keeping, it is not very explicit about the fact that it is going to reduce the dose, though it hints at a reduction by saying that the factor of reduction is being set to 0.67.

An additional problem, and the primary one we will address in this paper, is that while this way of giving explanations can state *what* the system does or did, it has only a limited ability to state *why* the system did what it did (see Figure 16-3). That is, the system cannot give adequate justifications for its actions. In the explanations given above, the system cannot state that it reduces the dose because increased calcium causes increased automaticity. The information needed to justify the program is the information that was used by the programmer to write the program, but it does not have to be incorporated into the program for the program to perform successfully—just as one can successfully bake a cake without knowing why baking powder appears in the recipe. Since it is desirable for expert programs to be able to justify what they do as well as do it successfully, we need to find a way of capturing the knowledge and decisions that went into writing the program in the first place. The remainder of this chapter will describe recent efforts we have made toward achieving that goal in the context of the Digitalis Therapy Advisor.[3]

16.5 Providing Justifications

We need a way of capturing the knowledge and decisions that went into writing the program. One way to do this is to give the computer enough knowledge so that it can write the program itself and remember what it did. Automatic programming has been researched considerably (Balzer et al., 1977; Barstow, 1977; Green et al., 1979; Long, 1977; Manna and Waldinger, 1977), but using an automatic programmer to help in producing explanations is a new idea. Since we are primarily interested in explanation, we have chosen not to deal with a number of problems that arise in automatic programming, including choosing between different implementations, backup and recovery from dead-end refinements, and optimization.

16.5.1 System Overview

XPLAIN is our framework for creating expert systems. Systems developed within it can be explained and justified. An overview is given in Figure 16-4. The system has five parts: a writer, a domain model, a set of domain principles, an English generator, and a generated refinement structure. The writer is an automatic programmer. It wrote new code that captured the functionality of major portions of the old Digitalis Advisor.[4] The domain model and the domain principles contain knowledge about the domain of expertise. In this case, they contain information about digitalis and digitalis therapy. They provide the writer with the knowledge it needs to write the code for the Digitalis Therapy Advisor. The refinement structure can be thought of as a trace left behind by the writer. It shows how the writer develops the Digitalis Therapy Advisor. When a physician-user runs the Digitalis Therapy Advisor, he or she can ask the system to justify why the program is doing what it is doing. The generator gives the user an answer by examining the refinement structure and the step of the advisor currently being executed. If we wanted to write a new program

[3]Clancey (1979c) notes that even in rule-based systems, knowledge is often too "compiled," resulting in explanation problems very similar to the ones described here.

[4]The code that has been written includes code to anticipate toxicities and to check for and assess various types of toxicities that may occur. As is discussed by Swartout (1981), it should not be too difficult to complete the remainder of the implementation so that the functionality of the old Digitalis Advisor is completely captured.

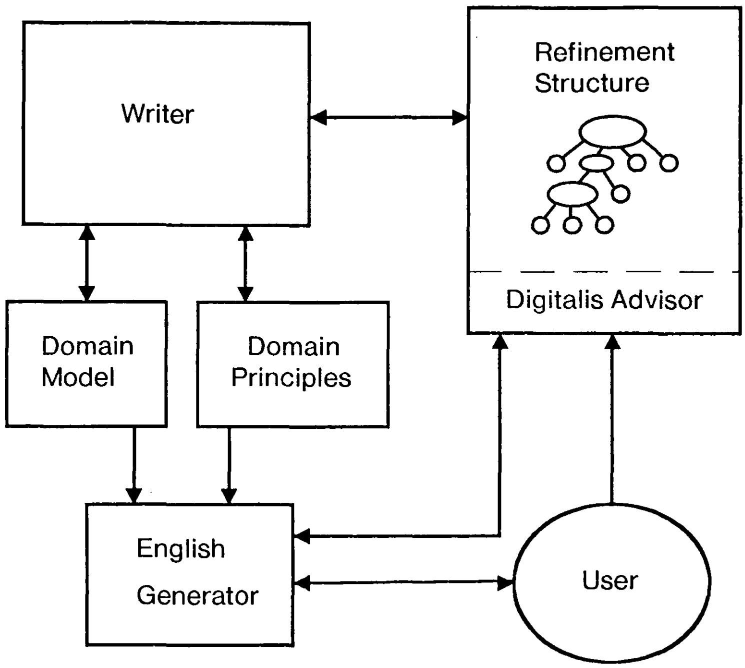


FIGURE 16-4 System overview.

covering a new medical domain, we would have to change the domain model and the domain principles, but we would not have to change the writer or the English generator.[5]

The refinement structure is created by the writer from the top-level goal (in this case, "administer digitalis") as it writes the Digitalis Therapy Advisor. The refinement structure is a tree of goals, each being a refinement of the one above it in the tree (see Figure 16-5). By *refining a goal,* we mean taking a goal and turning it into more specific subgoals. Looking at Figure 16-5, we see that the top of the tree is a very abstract goal, in this case, to administer digitalis. This goal is refined into less abstract steps by the writer. These more specific steps are steps the system executes to administer digitalis. For example, one such step is to anticipate toxicity, that is, to anticipate whether the patient may become toxic due to increased digitalis sensitivity. The writer then refines this more specific goal to a still more specific goal. Eventually, the level of system primitives is reached. System primitives are operations that are built in. Normally they are very basic, simple operations, so the fact that they cannot be explained is usually

[5]Note that the writer writes the program once, and once written, the program is static. It is not written "on the fly" during interaction with the physician user.

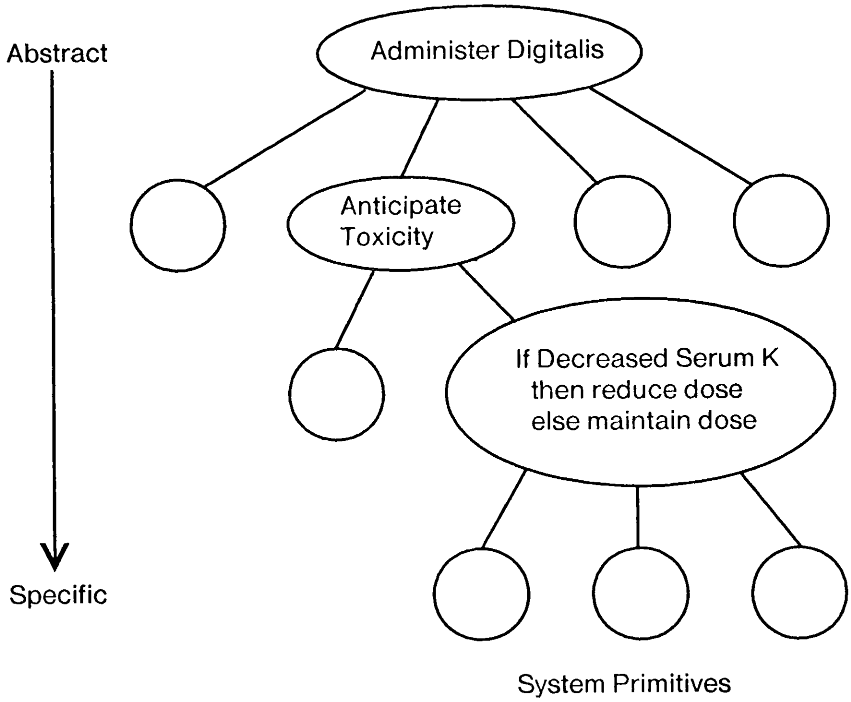


FIGURE 16-5 A sample refinement structure.

not a problem. Typical primitives include those that perform arithmetic operations like PLUS and TIMES and those that set variables to a particular value. The leaves of the refinement structure constitute the basic operations performed by the Digitalis Therapy Advisor, the program that we wanted the automatic programmer to produce.

The domain model is a model of the facts of the domain. In this case, it is a model of the causal relationships in digitalis therapy. A simplified portion of the model is shown in Figure 16-6. In this model, the boxes are states, and the arrows represent causality. This model shows some of the effects of increased digitalis. It also shows that increased serum Ca and decreased serum K can each cause increased automaticity. These facts correspond to the sorts of facts that a medical student learns in class during the first two years of medical school. They are descriptive in the sense that they describe what happens in the domain, but they do not tell how to achieve a goal, such as checking for digitalis sensitivity. The model says that increased digitalis can cause a change to ventricular fibrillation but it does not say what to do about it. Medical students go to medical school for an additional two years, and acquire these procedures by observing more experienced personnel as they practice medicine on the wards. The set of domain principles provides the writer with this sort of problem-solving knowledge.

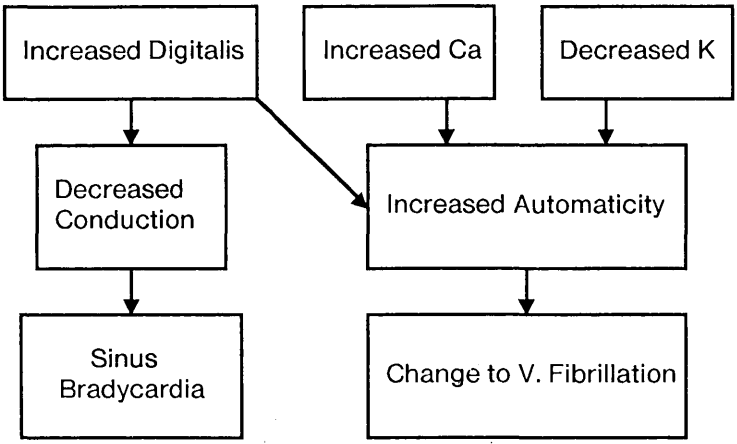


FIGURE 16-6 A simplified portion of the domain model.

Domain principles tell the writer how something (such as prescribing a drug or analyzing symptoms) should be done. They guide it as it refines abstract goals to more specific ones. A (somewhat simplified) domain principle appears in Figure 16-7.[6] This particular principle helps the writer in anticipating digitalis toxicity. It represents the commonsense notion that if one is considering administering a drug and there is some factor that enhances the deleterious effects of that drug, then if that factor is present in the patient, less drug should be given. This principle has three parts: a goal, a domain rationale, and a prototype method.

The goal tells the writer what it is that the principle can do. In this case, the principle tells how to anticipate toxicity. The domain rationale is a pattern that is matched against the domain model to provide further information necessary to achieve the goal. In Figure 16-7, arrows represent causality, just as they do in the domain model. Thus the system will look in the domain model to match a **Finding** (e.g., increased Ca) that causes some sort of a **Dangerous Deviation** (e.g., change to ventricular fibrillation) that is also caused by an increased level of the drug. By looking at the domain model, we can see both increased Ca and decreased K will match as findings, since both can cause a change to ventricular fibrillation.

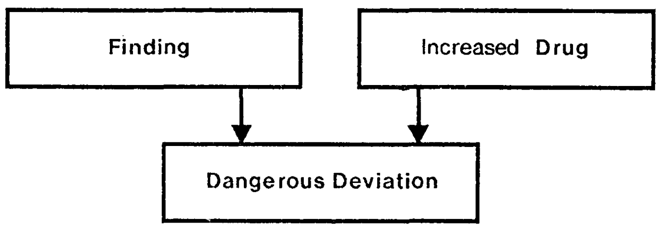
The prototype method is an abstract method that tells the system how to accomplish the goal. The steps of the prototype method are annotated to distinguish implementation details (such as record-keeping) from steps that are significant in medical problem solving. These annotations are used by the explanation routines to filter out implementation details when presenting explanations to medical personnel.

---

[6]Domain principles are composed of variables and constants. Variables appear in boldface in Figure 16-7. When the writer is matching, a variable in a pattern will match anything that is of the same kind as itself. Thus the variable **Finding** would match increased serum Ca or decreased K, since increased serum Ca and decreased K are both kinds of findings.

Goal: Anticipate Drug Toxicity

Domain Rationale:



Prototype Method:

If the **Finding** exists

then: reduce the **drug dose**

else: maintain the **drug dose**

FIGURE 16-7 An example of a domain principle.

After the domain rationale has been matched against the domain model, the prototype method is instantiated for each match of the domain rationale. When we say that we instantiate the prototype method, we mean that we create a new structure where the variables in the prototype method have been replaced by the things they matched. In this case, two structures would be created. In the first, **Finding** would be replaced by increased serum Ca, and **drug** would be replaced by digitalis. In the second, **Finding** would be replaced by decreased serum K, and **drug** would again be replaced by digitalis. Note that now, with these new structures, we have changed the single abstract problem of how to anticipate toxicity into several more specific ones, such as how to determine whether decreased serum K exists, how to reduce the dose, and how to maintain it.

After instantiation, the more specific goals of the prototype method are placed in the refinement structure as offspring of the goal being resolved. If we look at Figure 16-5, we can see that the instantiated prototype method that checks for decreased serum K has been placed below the anticipate toxicity goal. Once they have been placed in the refinement structure, the newly instantiated goals become goals for the writer to resolve. For example, after the writer applied this domain principle, it would have to find ways of determining whether increased Ca existed in the patient, whether decreased K existed, and ways of reducing and maintaining the dose. The system continues in this fashion, refining goals at the bottom of the structure and growing the tree down until eventually the level of system primitives is reached.

Please enter the value of serum-k: why?

The system is anticipating digitalis toxicity. Decreased serum-k causes increased automaticity, which may cause a change to ventricular fibrillation. Increased digitalis also causes increased automaticity. Thus, if the system observes decreased serum-k, it reduces the dose of digitalis due to decreased serum-k.

Please enter the value of serum-k: 3.7

Please enter the value of serum-ca: why?

(The system produces a shortened explanation, reflecting the fact that it has already explained several of the causal relationships in the previous explanation. Also, since the system remembers that it has already told the user about serum-K, it suggests the analogy between the two here.)

The system is anticipating digitalis toxicity. Increased serum-ca also causes increased automaticity. Thus, (as with decreased serum-k) if the system observes increased serum-ca, it reduces the dose of digitalis due to increased serum-ca.

Please enter the value of serum-ca: 9

FIGURE 16-8 A sample interaction providing justifications.

The system must also take into account interactions between the actions it takes. For example, while the individual instantiations above say that if increased serum Ca exists the dose should be reduced and if decreased serum K exists the dose should be reduced, they do not give any indication of what should happen if *both* increased serum Ca *and* decreased serum K occur. Exactly what should happen depends on the characteristics of the domain. It could be that the occurrence of either sensitivity "covers" for the other, so that only one reduction should be made and the predicate of the If should be made into a disjunction. Or (as is actually the case) it could be that when multiple sensitivities appear, multiple reductions should be made. The way to resolve that is to serialize these two program fragments, connecting the outputs of the first to the inputs of the second. The automatic programmer handles this situation by setting it up as something to be refined. The domain principle used in the refinement of this problem may further constrain the way in which other goals may be refined. The details of this operation will not be presented here. The interested reader should see Swartout (1981).

Once the refinement process is complete, we have a working expert system. A sample interaction with the system is given in Figure 16-8. The first sentence of the explanation was produced by stating the higher goal (that is, anticipate toxicity). Next, the explanation routines located the domain principle that caused the step in question to appear in the program. The domain rationale associated with that principle was then converted to English (with pattern variables replaced by the facts they matched in the domain model). That step produced the next two sentences of the explanation. The last sentence is just the instantiated version of the prototype method of the domain principle. These explanations should be compared

```
(describe-method [(check sensitivities)])

TO CHECK SENSITIVITIES I DO THE FOLLOWING STEPS:

1. I CHECK SENSITIVITY DUE TO CALCIUM.
2. I CHECK SENSITIVITY DUE TO POTASSIUM.
3. I CHECK SENSITIVITY DUE TO CARDIOMYOPATHY-MI.
4. I CHECK SENSITIVITY DUE TO HYPOXEMIA.
5. I CHECK SENSITIVITY DUE TO THYROID-FUNCTION.
6. I CHECK SENSITIVITY DUE TO ADVANCED AGE.
7. I COMPUTE THE FACTOR OF ALTERATION.
```

FIGURE 16-9 An explanation from the old Digitalis Advisor.

with those presented in Figure 16-3 to appreciate the improvement that is possible with this approach. [The generation routines are described in detail in Swartout (1981).]

16.5.2 Explanations of Domain Principles

In the old Digitalis Advisor, when we wanted to give a more abstract view of what was going on, we just described a higher-level procedure (Swartout, 1977). In this regard, we were following the principles of structured programming. While this approach often gave reasonable explanations, there were times when it was considerably less than illuminating. The general method for anticipating digitalis toxicity was called "check sensitivities" in the old Digitalis Advisor. An explanation of it appears in Figure 16-9. While this explanation does tell the user what sensitivities are being checked,[7] it does not say what will be done if sensitivities are discovered, nor does it say why the system considers these particular factors to be sensitivities. Finally, it is much too redundant and verbose. The first objection can be dealt with by removing the calls to lower procedures and substituting the code of those procedures in-line. This results in the somewhat improved explanation produced by XPLAIN when it is asked to describe the method for anticipating digitalis toxicity (see Figure 16-10). However, while this explanation shows what the system does, it does not say why things like increased calcium, cardiomyopathy, and decreased potassium are sensitivities, and if anything, it is even more verbose than the original explanation.

The reason we cannot get the sorts of explanations we want by producing explanations directly from the code is that much of the sort of reasoning we want to explain has been "compiled out." Thus we are forced

[7]The reader may notice that there were more sensitivities checked in the original version of the program than in the current version. We now feel that some of these, such as thyroid function and advanced age, should not be treated as sensitivities *per se* because they tend to have an effect on reducing renal function and hence slowing excretion, rather than on increasing sensitivity to digitalis. The other sensitivities would be easy to add by including the appropriate causal links in the domain model.

```
(describe-method [((anticipate*o (toxicity*f digitalis))*i 1)])
```

To anticipate digitalis toxicity:

(1) If the system determines that cardiomyopathy exists, it reduces the dose of digitalis due to cardiomyopathy.

(2) If the system determines that decreased serum-k exists, it reduces the dose of digitalis due to decreased serum-k.

(3) If the system determines that increased serum-ca exists, it reduces the dose of digitalis due to increased serum-ca.

FIGURE 16-10 An explanation from the code for anticipating toxicity.

```
(describe-proto-method [(anticipate*o (toxicity*f digitalis))])
```

The system considers those cases where a finding causes a dangerous deviation and increased digitalis also causes the dangerous deviation. If the system determines that the finding exists, it reduces the dose of digitalis due to the finding.

The findings considered are increased calcium and decreased potassium.

FIGURE 16-11 Explanation of a domain principle.

into explaining at a level that is either too abstract or too specific. The intermediate reasoning that we would like to explain was done by a human programmer in the case of the old Digitalis Advisor. However, because the Digitalis Therapy Advisor performance program was produced by an automatic programmer, that reasoning is available in the domain principle. For example, if we were to use the English generator to explain the domain principle that produced the code for anticipating digitalis toxicity rather than the code itself, we would get the explanation that appears in Figure 16-11. Thus the use of an automatic programmer not only allows us to justify the performance program, it also allows us to give better descriptions of methods by making available intermediate levels of abstraction that were not previously available.

16.6 Is Automatic Programming Too Hard?

One possible objection to the whole approach to explanation advocated in this paper is that it is just too hard to get an automatic programmer to write the performance program. Our original plan for producing better explanations was to create structures detailing the development of the performance program, but these structures would be created by hand rather than automatically, because it was feared that automatic programming was just too hard. However, as the research progressed, it became clear that if we had sufficiently powerful representations available so that it could be said that explanations were being produced from an understanding of the program, then actually writing the program automatically would not be all that much more difficult. This seems to be true in general. It seems that the primary difficulty in both explanation and automatic programming is a knowledge representation problem, and that the kinds of knowledge to be represented in both cases are similar, so that a solution to one makes the other much easier. However, it must be pointed out that the field of automatic programming is still an active research area and a number of difficult problems remain to be solved in addition to the knowledge representation problem, so this conjecture still awaits a final resolution.

16.7 A Summary of Major Points

First, we have argued that to be acceptable, consultant programs must be able to explain what they do and why. Second, we have described the various ways that traditional approaches fail to provide adequate explanations and justifications. Major failings include (1) the inability of such approaches to justify what the system is doing because the knowledge required to produce justifications is not represented within the system, and (2) a lack of distinction between steps required just to get the computer-based implementation to work and those that are motivated by the application domain. Third, we have outlined an approach that captures the knowledge necessary to improve explanations. This involves using an automatic programmer to generate the performance program. As the program is generated, a refinement structure is created that gives the explanation routines access to decisions made during the creation of the program. The improvement in explanatory capabilities that is achieved is due more to the availability of this refinement structure than to the use of more sophisticated English generation functions, since the explanation routines used in this paper do not differ greatly from those used in the old Digitalis Advisor.

ACKNOWLEDGMENTS

This research was supported (in part) by a National Institutes of Health grant (no. 1 P01 LM 03374-01) from the National Library of Medicine. The author wishes to express his thanks to Peter Szolovits for his insightful comments and suggestions during the course of this research.