**Knowledge Systems Laboratory** **original October 1990**
**Technical Report KSL 91-17** **Revised February 1991**

# Interactive Acquisition of Justifications: Learning "Why" by Being Told "What"

**by**

**Thomas R. Gruber**

To appear in *IEEE Expert*, 1991. Based on a lecture presented at the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, Kyoto, Japan, October 26, 1990.

**KNOWLEDGE SYSTEMS LABORATORY**
**Computer Science Department**
**Stanford University**
**Stanford, California 94305**

# Interactive Acquisition of Justifications: Learning "Why" by Being Told "What"*

**Thomas Gruber**

Knowledge Systems Laboratory
Stanford University
701 Welch Road, Building C
Palo Alto, CA 94304
gruber@sumex-aim.stanford.edu

Original October, 1990
Revised February 1991

## ABSTRACT

In this paper I describe an approach to automated knowledge acquisition in which users specify desired system behavior by constructing *justifications* of examples. Justifications are explanations of why example behaviors are appropriate in given situations. I analyze the problem of acquiring justifications, showing how current knowledge acquisition techniques are best suited for asking what-questions while justifications are naturally viewed as answers to why-questions. I sketch a new approach for acquiring justifications that transforms why-questions into what-questions, borrowing the sources of power of existing techniques. In this approach, users construct justifications by selecting facts that specify what is relevant in a situation from a space of facts provided by the elicitation tool. Justifications are then used to create operational mappings from situations to intended outcomes. I show how the approach is applied to two different knowledge acquisition problems: the acquisition of diagnostic strategy and the acquisition of design rationale. I conclude by identifying common characteristics of the two applications and discuss how their design distributes the cognitive load between human and machine.

*To appear in *IEEE Expert*, April 1991. Based on a lecture presented at the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, Kyoto, Japan, October 26, 1990.

# 1. Introduction

The vast majority of machine learning and automated knowledge acquisition systems learn *what* to do or believe. It can also be useful to learn *why* something should be done or believed. Asking a domain specialist to give reasons for a prescribed action or belief often reveals knowledge not anticipated by the system designer. A common form for "why" knowledge is the *justification:* an explanation of why a particular behavior, such as a decision to take some action or choose some alternative, is appropriate in a given situation. In this paper I analyze the problem of acquiring justifications from humans, focusing on two types of justifications. The first type answers the question "Why is this action chosen in this situation?" I demonstrate how a technique for eliciting answers to this question is used to acquire diagnostic strategy, in Section 2. The second type answers the question "Why is this device designed this way?" I illustrate a scheme for answering this question as a method for capturing design rationale in Section 3. The techniques for acquiring both strategic knowledge and design rationale share some interesting abstract properties, which I summarize in Section 4. In Section 5, I conclude with a discussion of how the approach distributes the cognitive load among a system designer, a cooperative domain specialist, and an elicitation program.

## Automated Knowledge Acquisition

Knowledge acquisition is the process of constructing computational models that represent human knowledge in a form that can be used by knowledge-based systems. Knowledge acquisition is inherently difficult because of the conceptual gap, or representation mismatch, between the form in which knowledge is described in the natural discourse of the domain specialist and the form in which it is represented in knowledge-based programs [Buchanan et al., 1983; Gruber, 1989b; Gruber and Cohen, 1987]. To be effective, knowledge acquisition techniques must reduce the conceptual gap between the two representations. Manual approaches such as KADS [Breuker and Wielinga, 1985; Breuker et al., 1987] and ontological analysis [Alexander et al., 1988], and protocol analysis [Ericsson and Simon, 1984] help the system builder analyze and formalize domain knowledge—bringing the natural discourse closer to the operational form. Automated knowledge acquisition approaches help bridge the gap from the other direction—making a pre-designed, operational representation more accessible to the user.

The state of the art in automated knowledge acquisition is the paradigm of task- and method-specific tools. These interactive programs help users build computational models by *instantiating* existing, partial models rather than constructing them from scratch. An example of a successful task-specific tool is OPAL [Musen et al., 1987], which helps oncologists specify operational cancer treatment protocols that are managed by the ONCOCIN expert system. OPAL presents the user a predefined vocabulary of representation primitives, such as terms for representing drug and radiation treatment methods. It elicits instances of these terms with form-filling interfaces, and elicits simple procedures (composed using these terms as operations) with direct-manipulation flow-chart editors. The elicited knowledge is guaranteed to be operational, since the terms with which the knowledge is acquired are already implemented with respect to the skeletal plan refinement method of ONCOCIN.

Similarly, method-specific knowledge acquisition tools present to the user a language of predesigned terms that are the parameters of domain-independent problem-solving methods, each of which is designed to solve a particular class of tasks. (These algorithmic methods are called *generic task problem solvers* [Bylander and Chandrasekaran, 1987; Chandrasekaran, 1986] and the parameters are called *roles* of the problem-solving method [McDermott, 1988].) For well-

understood classes of tasks and problem-solving methods, the system designer can build a method-specific shell and associated knowledge acquisition tool that can elicit statements in these terms, using user-interface techniques such as prompted interviewing. Since the operational semantics of the elicited statements can be understood in the context of the problem-solving methods, these tools can perform consistency and completeness analysis, providing useful feedback to the user. SALT [Marcus, 1988], for example, can identify cycles in a plan for configuring a system by analyzing the graph of plan step proposing and revising rules elicited from the user; this analysis is based on an understanding of the propose-and-revise method that applies the rules at run time.

## Asking What and Asking Why

The task- and method-specific tools allow the user to specify *what* a knowledge-based system should do or believe, rather than *how* to achieve the desired behavior—effectively replacing programming with specification. The OPAL user specifies what drug therapies are appropriate for a given patient condition; OPAL knows how this advice should be incorporated into the skeletal plan that drives ONCOCIN. Similarly, the SALT user specifies what components to add next in a given state of the configuration, and the tool knows how proposed extensions are applied by the backtracking algorithm that does the configuration. These knowledge acquisition systems work because user interfaces can be designed for asking what-questions (e.g., form-filling and prompted interviewing interfaces) that nonprogrammers can answer.

In manual knowledge acquisition sessions, knowledge engineers ask not only what-questions, but also why-questions. The answers to why-questions can provide further insight into the domain and problem-solving expertise. The developer of a cancer treatment application might ask which drugs are appropriate in various situations, but also why they are appropriate. The domain specialist might reply to the why-question with new, relevant information, such as an assessment of drug efficacy. The builder of a configuration system might ask why one part should be included before another, to elucidate order dependencies among design or assembly plan steps.

The problem addressed in this paper is how one might design a human-machine dialog to ask a class of why-questions for acquiring knowledge in the form of justifications. We view justifications as a form in which domain specialists can answer the why-questions to provide knowledge relevant to the task. For example, the oncologist might justify the use of a particular drug in a particular therapeutic situation by explaining how the drug is expected to satisfy the goals of the protocol (i.e., to treat the patient's condition). The configuration expert might justify the strategy of determining the space and power requirements before choosing the power supply by explaining that the selection of power supplies is dependent on space and power requirements.

How could one design a knowledge acquisition dialog for eliciting justifications so that the knowledge is useful for some computational task? We can begin by adapting the design strategy used for the task- and method-specific tools described earlier: namely, to design a representation language of implemented, task-specific terms and build a structured interface for eliciting statements using these terms. For a given task, we need to determine the computational role of justifications and design an appropriate representation for them. Given such a representation, we also need a way to ask for justifications in a human-machine dialog. Here we face the problem that justifications are naturally stated as explanations that are *constructed* rather than *selected*. While what-questions can often be answered by selecting from pre-enumerated alternatives or by filling forms, why-questions are often answered with more complex constructions. For example,

one can ask which drug to use by eliciting its name with a form and selecting its type from a menu. A justification for why the drug is appropriate for a given patient in a particular state at a particular stage of treatment would be difficult to acquire with a menu, since the set of relevant combinations of possible drugs, patient states, and treatment stages would be hard to anticipate and harder to present to the user.

However, one can transform the why-question into a what-question by considering the nature of justifications. We can treat the justification as a declarative specification of what the situation is, what the possible choices are, and a set of reasons for a particular choice being appropriate. Furthermore, we can treat reasons as specifications of *relevant features* of the situation or the choice. For example, the reason for the drug choice is its efficacy, which is a function of the choice (the drug) and the situation (the protocol context, including patient state, disease, etc.). If "efficacy" is a term in the task-specific representation, then the efficacy of a particular drug in a particular protocol can be *selected* as a reason for choosing that drug. The fact that efficacy—among the many possible features—is relevant to a particular situation is the key knowledge provided by the domain specialist (i.e., not anticipated by the system builder). Furthermore, if "efficacy" is present as an implemented function in the environment of the target knowledge system, then the specification of the particular efficacy of the chosen drug in the given situation could be computed in the context of an example protocol in a running protocol management system. This avoids the need to pre-enumerate all possible reasons (i.e., all values of features in all situations). The intuition can be summarized as a heuristic for the design of an interaction dialog:

> *To ask why a choice is appropriate in a given situation, ask for a set of relevant features of the situation and the choice in the context of a specific example in a running system where features can be computed.*

A sketch of this approach to acquiring justifications is shown Figure 1.

Given:

- A partial domain model that defines a space of situations, possible alternatives for given situations, and features of situations and alternatives.
- An interactive environment in which situations, alternatives, and their features are explicitly represented and accessible by the user.

Elicit from the user:

- A training example specifying a situation and a preferred choice among the possible alternatives in that situation.
- A list of features (of the situation and the alternatives) that are deemed relevant to the decision favoring the chosen alternative.

Compute from the model:

- The values of the relevant features in the example.

Present to the user:

- The set of relevant features and their values as an explanation of why the choice was appropriate in the situation.

**Figure 1:** An approach to acquiring justifications.

This idea has been explored in two very different applications: the acquisition of strategic knowledge for medical diagnosis and the acquisition of design rationale for physical devices.

## 2. The Acquisition of Strategic Knowledge

Strategic knowledge is about deciding what to do next. More precisely, it is knowledge used by an agent to decide what actions to perform in given situations, where actions have consequences external to the agent, and situations change with time and the effects of actions. In the context of conventional knowledge systems, actions are observable outputs such as recommendations to people (e.g., "take a throat culture, then give antibiotics") or signals to physical systems (e.g., "close valve V now!"), and strategic knowledge is used to choose the best action to perform at each point of interaction between the system and the environment. Strategic knowledge is concerned with observable situations and actions at the knowledge level [Newell, 1982], rather than the internal symbol-level activity of a program (e.g., search control).

Domain-specific strategic knowledge is pervasive in many planning and control tasks, where actions with varying utilities must be chosen without knowing the consequences of the actions in advance. As an example, consider how medical diagnosis is performed in the real world. Physicians usually do not start out with feature vectors of all the possibly relevant data on a patient, and make a definitive classification. In clinical practice, evidence for a diagnosis is gathered over time with non-negligible costs. At each point in the workup, the physician re-

evaluates the current information on the patient's condition and strategically chooses the next action (e.g., question to ask, diagnostic test to run, or therapy to try), balancing factors such as diagnostic power, therapeutic value, and cost. In different cases the physician performs different sequences of actions, and order matters. This task is called prospective diagnosis, and domain-specific strategic knowledge is essential to expert performance. Therefore, to build a knowledge-based system for this task we cannot rely on general-purpose control strategies; we need a technique to acquire strategic knowledge from the experts.

To study the role of strategy in reasoning under uncertainty, Paul Cohen and his colleagues built a knowledge system for the prospective diagnosis of chest pain [Cohen et al., 1987]. In interviewing practicing physicians, they found that it was far easier for the experts to offer rules of evidential support, specifying which hypotheses to believe given various data, than to describe their diagnostic strategy [Gruber and Cohen, 1987]. In knowledge acquisition sessions for the initial version of the application, knowledge engineers asked a physician to walk them through the sequences of diagnostic and therapeutic actions taken for a number of patient cases. At each decision point in a case, they asked the physician *why* a particular action was taken next. The answers were statements like "this is the cheapest test for discriminating among these hypotheses" and "when I suspect a dangerous condition, I prescribe drugs that hedge against not treating the condition and could also provide evidence for the diagnosis." The knowledge engineers generalized from this kind of advice and wrote a planning algorithm that embodied what they thought were the underlying strategies. The conceptual gap between the statements elicited from the domain specialist and the corresponding procedures written by knowledge engineers meant that it would be difficult to acquire the strategic knowledge directly in terms of procedures.

## ASK: A Tool for Acquiring Strategic Knowledge

This experience motivated the design of ASK, an interactive program for acquiring strategic knowledge from experts [Gruber, 1989a, 1989b]. ASK is of interest here because it employs a technique for acquiring justifications that applies the idea of transforming why-questions into what-questions. Briefly, here is how it works.

ASK operates in a performance context in which strategic choices are made. The context is a partially implemented knowledge system for prospective diagnosis. In the scenario I will use for illustration, the knowledge system is equipped with diagnostic knowledge (relationships between evidence and hypotheses), and includes a library of possible diagnostic and therapeutic actions (e.g., lab tests, diagnostic procedures, drugs). At each iteration of a decision cycle, the knowledge system chooses an action, executes it, and propagates the results through an inference network to update its model of the patient condition. Strategic knowledge is used to choose among actions at each decision point.[1] ASK is used to acquire knowledge that will improve the ability of the system to choose appropriate actions.

The target knowledge system serves as the partial domain model mentioned in Figure 1; it defines the space of situations, choices, and features. The knowledge elicitation comes in two phases. First, ASK needs an example specifying a situation and a choice. The user provides this example by running the interactive knowledge system until it comes to an interesting choice

[1]The strategic knowledge is represented in the form of strategy rules that map states of working memory (specified with Horn clauses) to classes of appropriate actions (specified with operators for generating and filtering sets of candidate actions).

among actions.[2] The current situation is now operationally defined by the state of the application; in the chest pain application, the state includes the data already gathered, the set of working hypotheses, and other features supported in the working memory of the system. The user interrupts the knowledge system and indicates which action she would have taken in the situation (the positive example) and an example of an action that would not have been appropriate (the negative example).

Then ASK elicits justifications for the strategic decision, which are reasons for choosing one action over another. Following the approach outlined in Section 1, the interface presents features of the current situation—abstractions over the state of the working memory—and features of the positive and negative actions. The user selects from among these features a set of *relevant* features for choosing the positive example over the negative example. ASK computes the values of the specified features for the current state and the positive and negative actions, and presents these object / feature / value tuples as English sentences that are intended to explain why the chosen action is appropriate.

The interaction is a mixed initiative dialog, with ASK and the user both operating on a shared workspace. ASK uses heuristics to suggest possibly relevant objects and features, seeding the set of justifications. The user can edit this set and add more relevant features; ASK continually updates the corresponding English sentences. When the user is satisfied that the explanation mentions all relevant features, and ASK verifies that the feature values are sufficient to distinguish the positive and negative examples, then ASK generates a strategy rule that will cause the system to choose similarly in future situations.[3]

## Acquiring Strategic Knowledge for Prospective Diagnosis

I will now quickly demonstrate the look and feel of ASK's dialog to show how justifications can be acquired with a fairly simple selection interface. In this scenario, the user is running a knowledge system for the prospective diagnosis of chest pain, and is part-way through a session for a patient who has shown evidence of angina. The system already has a few strategy rules that tell it to do the cheap and easy actions first (which are simple questions that have already been answered in this session), and then to perform tests that gather evidence for current working hypotheses (i.e., those on the differential, the list of suspected diseases). The user will teach the system to refine this strategy for the case where *critical* hypotheses exist; in this situation, the system should concentrate on tests that confirm or rule out critical hypotheses first.

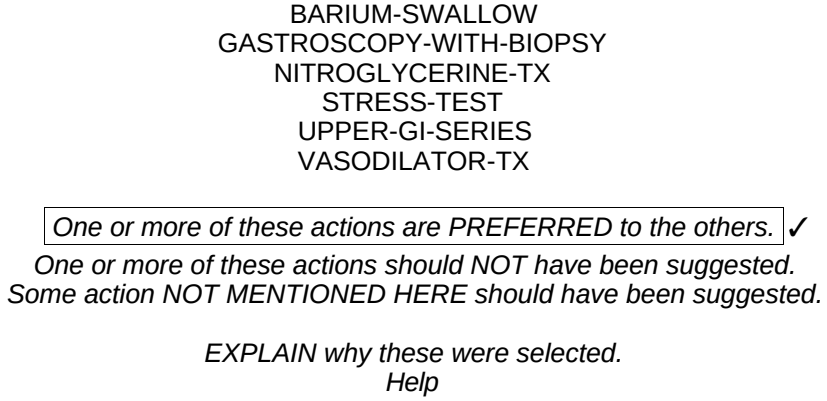
In the current state of the diagnosis, the system suggests six possible actions. (Whenever it lacks strategic knowledge to warrant selecting a single next action, the system leaves it to the application user to choose among the actions it has some reason to select.) The domain expert, playing the role of the application user, wants the system to be more discriminatory in selecting tests and treatments in the situation. Therefore the expert interrupts the normal data-gathering

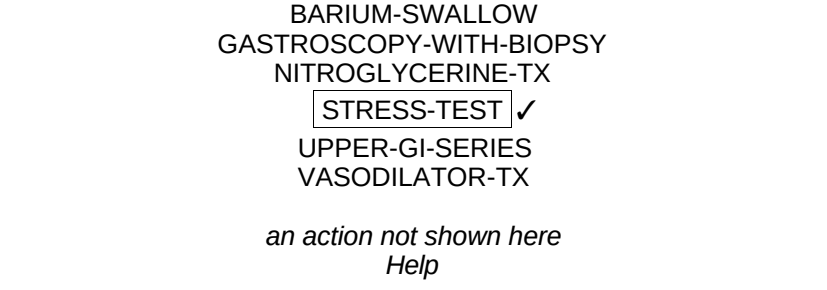
---

[2]The MU architecture, upon which ASK's target knowledge systems are built, allows the user to run the decision cycle by hand, choosing actions from the set of possibilities. Any strategic knowledge that the system already has been given is used to prune the space of possible next actions at each decision point. Therefore, the example-selection technique can work with any amount of prior strategic knowledge, although it is better constrained when the prior strategic knowledge narrows the choices in a given situation.

[3]ASK also has to verify that the new set of features is sufficient to produce a strategy rule that will cause the system to choose the same way as the user, given the same state and the existing body of strategy rules. The details of ASK's learning algorithms are available in [Gruber, 1989a].

cycle of the application, and offers a critique. ASK elicits positive and negative examples, actions that should and should not be chosen in this situation. In this case, the expert indicates that instead of recommending all six actions as equally appropriate, the system should have preferred the Stress-Test over alternative actions, such as Upper-GI-Series. This interaction is shown in Figure 2.



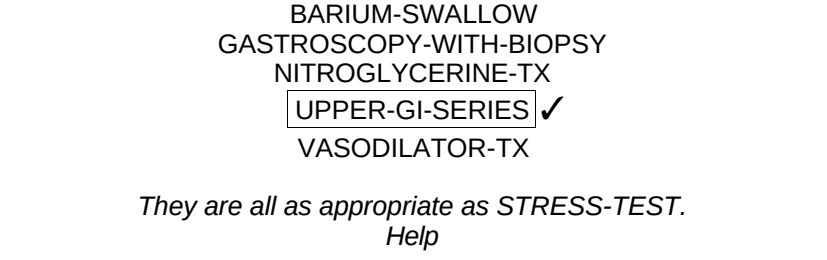




**Figure 2:** Eliciting a critique of the system's choice of actions. In the interaction shown in the top window, the user is presented with the set of actions that the system proposes for the current action cycle, and the user indicates that the system instead should have proposed a smaller set of preferred actions. In the second and third windows, the user picks training exemplars, indicating that actions like Stress-Test should have been preferred over actions like Upper-GI-Series in this diagnostic situation.

GATHER-EVIDENCE-FOR-DIFFERENTIAL is in the CURRENT-GOALS.
STRESS-TEST is in the potential-evidence of DIFFERENTIAL.
UPPER-GI-SERIES is in the potential-evidence of DIFFERENTIAL.

**CRITICAL-HYPOTHESES**
**potentially-conclusive-evidence:** Stress-Test. ✓
**value:** Classic-Angina.
**CURRENT-GOALS**
**value:** Gather-Evidence-For-Differential.
**DIFFERENTIAL**
**potential-evidence:** Barium-Swallow, Gastroscopy-With-Biopsy, Nitroglycerine-Tx, Str…
**potentially-conclusive-evidence:** Gastroscopy-With-Biopsy, Stress-Test, Upper-Gi-S…
**value:** Classic-Angina, Esophagitis, Esophageal-Reflux, Pericarditis.
**STRESS-TEST**
**applicability:** Applicable
**classes:** Diagnostic-Tests.
**cost:** Medium
**executed?:** No
**potentially-confirms:** Classic-Angina, Prinzmetal-Angina, Unstable-Angina.
**UPPER-GI-SERIES**
**applicability:** Applicable
**classes:** Diagnostic-Tests.
**cost:** Medium
**executed?:** No
**potentially-confirms:** Esophageal-Reflux, Esophagitis.

***more below***

GATHER-EVIDENCE-FOR-DIFFERENTIAL is in the CURRENT-GOALS.
STRESS-TEST is in the potential-evidence of DIFFERENTIAL.
UPPER-GI-SERIES is in the potential-evidence of DIFFERENTIAL.
*There are some CRITICAL-HYPOTHESES.*
*STRESS-TEST is in the potentially-conclusive-evidence of CRITICAL-HYPOTHESES.*

**Figure 3:** Eliciting justifications by selection of relevant features. The top window shows the initial set of justifications, generated by ASK to explain why the current six actions were chosen by the system. The middle window shows the set of objects and features that ASK guesses are potentially relevant to the current strategic decision. The user mouses on the fact that the action Stress-Test is potentially conclusive for the hypotheses on the Critical-Hypotheses list, which currently contains the single hypothesis Classic-Angina. A paraphrase of this fact is added to the contents of the justifications window, as shown in italics the bottom window. Only a portion of the logical contents of these windows will fit on the computer screen, as indicated by "…" and "*more below*". The interface makes it convenient to scroll to other portions of the contents, as well as edit the set of objects logically contained in the windows.

Then ASK elicits reasons for choosing the positive example over the negative example with the justification interface shown in Figure 3. The window shown at the top of Figure 3 is the justifications window. Each sentence in the window is a paraphrase of a pattern-matching expression over features of objects, instantiated on objects of the current situation. These justification sentences are intended to serve as an explanation of why one action is preferred over another.

ASK begins the elicitation dialog by seeding the justifications window with the *system's* reasons for choosing actions. In the current example, one of the reasons for selecting these six actions in the current situation is that they are all elements of the potential-evidence feature of the differential. We also see in the justification window the statements that Stress-Test and Upper-GI-Series are in the potential-evidence set. That is the reason why the system choose both actions in the current situation, which is specified by the fact that the goal gather-evidence-for-differential has been posted in this state.

The window shown in the middle of Figure 3 is the relevant objects window. It presents a view of objects in the knowledge base that might be relevant to the strategic decision, and their features. For example, ASK assumes (using domain-independent heuristics) that the current goal, the critical hypotheses, the hypotheses on the differential, and the positive and negative examples are all potentially relevant, so it presents these objects and their features. Features are attributes, functions, and relations whose values are dynamically maintained during a session (e.g., the set of hypotheses on the differential and the actions that are potential-evidence for them are updated with each new datum). The user can add other knowledge base objects to the relevant objects window if ASK failed to guess that they might be relevant.

The user creates additional justifications by pointing out relevant features of the current situation, as presented in the relevant objects window. In this example the user has indicated that two additional factors should be considered in this situation. The first is the fact that a critical hypothesis has been activated. The user indicates that this fact is relevant by mousing on "value: Classic-Angina" under "CRITICAL-HYPOTHESES." The second factor is the fact that Stress-Test has a special relationship to the critical hypothesis—it could potentially confirm or rule out the hypothesis. The user indicates that this fact is relevant by mousing as shown in the middle window of Figure 3.

As the user specifies relevant features, the justification window is updated accordingly. The subsequent state of the justification window is shown in the bottom of Figure 3, with the new justifications set in italics.

The user continues modifying the list of justifications until she is satisfied that it explains why the Stress-Test action should be chosen over actions like the Upper-GI-Series, and until ASK is satisfied that the given justifications can actually distinguish the two actions.

Then ASK operationalizes and generalizes the justifications, forming a strategy rule that becomes part of the knowledge base. In similar situations in the future, the knowledge system will use the new rule to choose among actions.

To summarize, ASK provides a way for humans to specify why an action should be chosen by selecting the relevant features of situations and actions in the context of a running knowledge-based system. A selection interface is feasible because the set of possibly relevant features can be

presented for a *specific* case.[4] ASK can transform the justifications into operational strategic knowledge because the list of relevant features can be generalized into pattern-matching expressions and incorporated into a knowledge base of strategy rules.

Now I will turn to the second knowledge acquisition problem, the acquisition of design rationale. This is work in progress at the Stanford Knowledge Systems Laboratory.

# 3. The Acquisition of Design Rationale

For many engineering tasks, including redesign, verification, and diagnosis, it is important to know why an artifact was designed the way it was. *Design rationale* is a general term referring to the knowledge or reasoning underlying a design. Although there is wide agreement that design rationale is valuable knowledge, there are few means for effectively acquiring it. To understand the rationale for a design often requires a broad range of knowledge, including the structure of the artifact, the reasons for choosing particular components or implementation approaches, the assumptions about the context in which the artifact is to be used, and the institutional experience with designing and manufacturing similar artifacts (e.g., design case histories).

One approach to design rationale capture is to provide electronic notebooks that encourage designers to record their design processes on line [Lakin et al., 1989]. Another is to elicit semistructured text about the design rationale, based on argument-style discussions among designers [Conklin and Begeman, 1988]. Another is to treat designs as parametric decisions, and to elicit reasons pro and con for parts of the design in a decision-theoretic or similar framework [Shema et al., 1990]. A fourth approach is to record the history of design choices and provide intelligent indexing into a shared design memory [Mark and Schlossberg, 1990]. A fifth approach is to acquire design knowledge in the form of engineering models [Baudin, Sivard, and Zweben, 1989; Gruber and Russell, 1990]. All of these approaches deal with the acquisition of different aspects or abstractions of design knowledge.

In this paper I frame design rationale capture as a knowledge acquisition problem, where the task is to elicit, from domain specialists (designers), knowledge that is sufficient to enable a program to generate explanations of how the designed artifact is intended to achieve its function. I treat this as a problem in acquiring justifications, where design justifications are explanations of why design choices are appropriate in the context of requirements, intended function, and assumed operating conditions.

Recall the basic approach to acquiring justifications summarized in Figure 1. To adapt this to the acquisition of design justifications, we need a partial domain theory with a representation for features of situations and choices, means for eliciting example decisions in context, an interface for eliciting relevant features of the situation and choice, a set of implemented functions for computing values of some features, and a way to present the elicited information as an explanation.

For a domain theory, we can borrow the representations and modelling methods of engineering. In engineering domains, designs are recorded—at the very least—as artifact descriptions: the components used, how they are connected, etc. Engineers also have techniques for modelling the behavior of their artifacts. Situations, then, can include the context in which

[4]The set of features often needs to be extended during knowledge acquisition, which is hard to automate in any general way. I discuss this limitation in Section 5.

artifacts exist—structurally, as parts of larger systems, and behaviorally, as black boxes that implement functions. In addition to achieving specified behaviors, designs must also meet other requirements specified as constraints, such as cost, weight, size, and reliability. Choices in design are alternatives considered during the design process, such as alternative components for achieving a function or alternative materials for building a structure. Some features of situations and choices can be easily retrieved, such as cost and weight. The values of other features can be computed by simulation and analysis tools. For example, a straightforward justification for a design is a verification that it achieves all of its requirements, including intended functions, which are specified in terms of measurable behaviors. Engineering verifications are typically performed by simulation and analysis (i.e., not by proof), which predict whether given structures produce behaviors within prescribed limits. The predicted behaviors can be thought of as the computed features, analogous to the predicted relationships between evidence and hypotheses in the prospective diagnosis application.

Given this framework, one can imagine how an ASK-style justification interface would appear. A justification window would contain the list of relevant requirements and properties of chosen components. A relevant objects window would display requirements, components, materials, etc. It might look like the mock up in Figure 4.

MINIMIZE-COST is in the CURRENT-REQUIREMENTS.
MAXIMIZE-RELIABILITY is in the CURRENT-REQUIREMENTS.
The RELIABILITY of 100A-DIODE-PAIR is HIGH.

**CURRENT-REQUIREMENTS**
**value:** Minimize-Cost, Maximize-Reliability.
**RELAY-XYZ**
**cost:** Low
**reliability:** High
**100A-DIODE-PAIR**
**cost:** Low
**reliability:** High
**behavior: ????**

**Figure 4:** How *not* to acquire design rationale.

However, adopting the ASK-style interface wholesale would be naive. Relevant features of design contexts, especially descriptions of behavior, are very difficult to formalize as properties of objects and logical relations among them. In the case of strategic knowledge for prospective diagnosis, it is possible to represent relevant diagnostic state in terms of relevant features, such as the set of critical hypotheses and their inferential relationships to evidence-gathering actions. In the case of design rationale, it is not as simple; most of the difficulty of capturing the knowledge is due to the difficulty of specifying intended behavior. If designs could be reduced to decisions among well-defined objects with comparable features, then a decision-theoretic technique such as that proposed in CANARD [Shema et al., 1990] would suffice. Where components can be annotated with properties such as cost, and requirements such as global cost ceilings can be enumerated, a rationale-capture tool should provide electronic means for recording this information. However, the knowledge of expected behavior and assumed operating conditions is

typically incomplete, implicit, or missing in design documentation today (electronic or otherwise), even though this knowledge is extremely valuable to capture in a reusable form.

To acquire these more elusive "features of design situations," we need a representation and interface metaphor other than menus, charts, and semistructured text. For a solution, we have turned to the technology for the interactive construction of engineering models and explainable simulations.

## Explainable Simulation as a Communication Medium

In engineering practice, simulation is used to predict the behavior of engineered artifacts. A simulation requires behavior models, typically in the form of equations, that describe the behavior of individual components and general physical processes. The simulation predicts the system behavior that arises from the interactions of components and the operating environment. Although the computation of behavior from a model is an automated process, the formulation of behavior models is an interactive activity involving human engineers making assumptions, setting up scenarios, and writing constraints to answer some class of questions about the behavior of an artifact.

We are exploring the idea of employing computer simulation as a communication medium for design rationale. The basic approach is to arrange for a knowledgeable user to set up simulation scenarios, configuring and guiding a simulation tool so that the predicted results account for the phenomenon to be communicated. Instead of writing text describing a design, one engages in a dialog with a knowledge-based modelling and simulation environment, telling the machine what *it* needs to know to generate a demonstration of some behavior of interest.

The basic insight in using simulation as a medium is that the information that the engineer enters to set up a simulation scenario captures knowledge in a machine-intelligible form, and this knowledge can be reused later by consumers of this knowledge. The specific models and input conditions that are specified document a behavior of interest and intended context of use. The author of such a document manipulates a simulation scenario until the simulation output corresponds to the intended message. Because the simulation output is generated from underlying models, it can be used to answer questions not typically anticipated by the author of static documentation, and can be effectively indexed with other on-line information about the device.

By acquiring this knowledge in an interactive environment, a tool can provide structured interfaces to help with elicitation and offer computational feedback on the implications of what is entered. For example, when writing free text documentation, it is easy to forget to specify details of the assumed operating conditions or to be imprecise about the expected behavior. A simulation program, like a task-specific problem-solving architecture, can determine when the set of initial conditions is complete and can compute expected behavior trajectories from models. Furthermore, it can be difficult to describe intended function in natural language when the behavior of interest is dynamic and complex, whereas simulation tools are designed to predict dynamic, complex behaviors from unambiguous models.

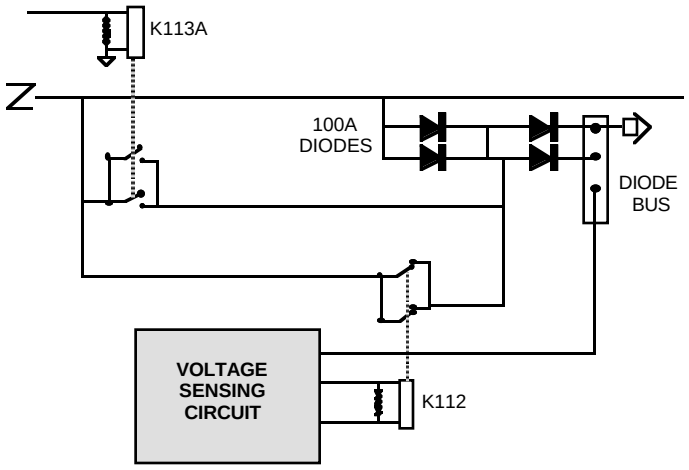
Given this view of simulation scenarios as a medium for specifying design intent, we can return to the problem of acquiring design justifications. What are the situations, choices, features, and explanations? The situations can be described in terms of the factors that go into design decisions: requirements such as desired behaviors (functions and behavioral constraints), structural contexts of the designed artifact (how a device is embedded in a system), and behavioral contexts (operating regions and initial conditions). The choices are design

alternatives, such as different selections of components. The explanations are accounts of how the parts of the design interact in the specified context to produce the desired behavior.

The interface for the elicitation of an example and its justification is provided by the interactive modelling and simulation environment. The user sets up the example by configuring a simulation for part of a designed system and some aspects of its behavior that illustrate why it was designed as it was. The user identifies the *relevant* structure and *intended* behavior by constraining the possibly relevant structure and behavior within the closed world defined by the modelling primitives. The machine generates explanations based on the outcomes of simulation. This captures a justification for a single choice. To distinguish one design alternative from another, the user configures a simulation scenario differing from the first by the structure of interest.

## Explaining the Design of an Electronic Circuit

Consider the text fragment from a NASA document explaining the purpose of a piece of a circuit in the electrical power system of the Hubble Space Telescope, shown in Figure 5. Note that the human explanation is i*mprecise* and *inconsistent* ("limit the voltage to 32.0 V" ≠ "keep it between 34.12 and 31.41 V") and *incomplete* ("how does inserting diodes accomplish the voltage-limiting function?", "what are the intended operating conditions?", "why 2 diodes?", "what are the other diodes doing there?").




5.1.5 Voltage Limiter

A voltage limiting circuit is provided in each diode bus feeder to limit the voltage at the using equipment to 32.0 V. With the SSM design change to six 23-cell, type 44 batteries the user input could exceed the 32 V user specification. To prevent this, an additional diode pair (parallel 100A diodes) was added in series with the load and bypassed by a relay that is operated by a voltage sensing circuit. When the diode bus voltage exceeds 34.12 ± 0.2 V the bypass relay opens and inserts the diode in the circuit as a voltage dropping element. When the voltage falls to 31.41 ± 0.2 V the relay will reclose, bypassing the diodes. The voltage sense circuit board used in this circuit is essentially the same as the circuit used in the charge current controller except for the set point voltage.

**Figure 5:** Human-generated explanation of how a voltage limiter works.

No system or document would be able answer all the questions one could think of concerning this circuit. However, an explanation generated from an underlying model of the device would have available considerably more complete and consistent detail than a static document. A fixed set of query types could be supported, organized by components, processes, etc. If the input models were acquired by assembling model fragments from a model library, then the information concerning the individual components, processes, and so forth, could be entered methodically prior to the interactive construction of a simulation scenario. The model library would be the place where static features of components such as cost, weight, size, etc. could be stored. This would allow the user to annotate the scenario by selecting features of components as relevant to the design decision, producing auxiliary justifications in the style of ASK, at little additional cost.

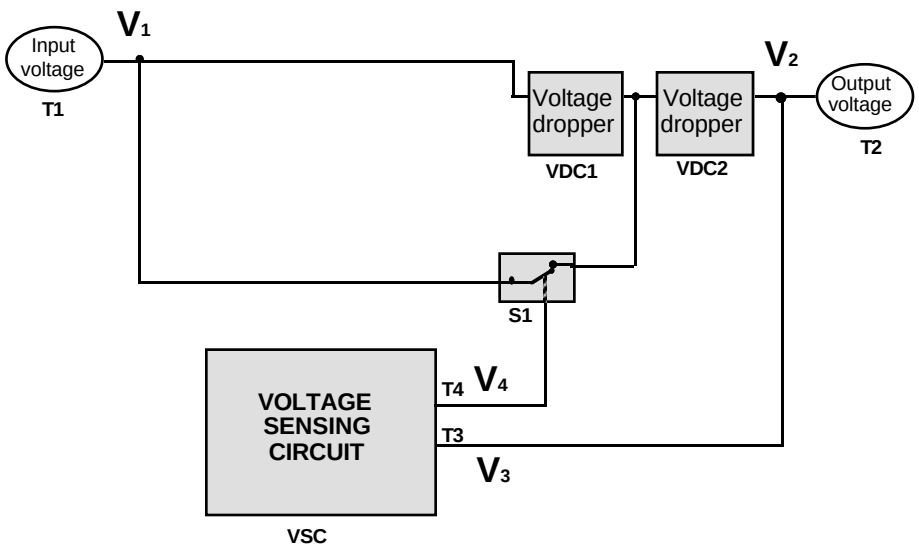
Consider a hypothetical dialog for acquiring the relevant structure and behavior to justify the design. First, the user identifies relevant structure, such as the parts to be considered in the scenario, treating some subsystems as as atomic components (black boxes). Assuming that the artifact structure is already stored in a CAD database, this interaction can be automated with direct manipulation on graphical presentations of device models and with selection from information in tables. Then the user specifies relevant behavior by setting up initial conditions for the simulation and dynamically specifying the values of underconstrained parameters. The simulation environment will need to query the user for this information; the user should not be

expected to supply it all in advance. For example, the user may have forgotten to specify whether the relay is initially open or closed, or whether the voltage sensing circuit begins with its output line on or off. A simulation system can identify inconsistent settings of initial conditions (e.g., voltages at various points that do not obey Kirchhoff's laws).

Then the program performs a simulation, producing state descriptions that can be used to generate a summary explanation of a causal path from structure to behavior. The explanation is presented in graphical and textual form, allowing for direct manipulation of presented objects. The user evaluates the explanation and changes inputs to the simulation as appropriate (i.e., specifying different operating assumptions). When the intended behavior is displayed, the user labels the scenario (e.g., "The function of the voltage limiter"), and repeats the process for alternate design scenarios.

For example, consider the hypothetical output from a qualitative simulation program shown in Figure 6. The explanation is organized around qualitative events such as the opening of the relay and the change in current path. By mousing on various parts of the text, the user could ask follow up questions, such as "What is the value of TV1?" and "What caused the current flow from T1 through VDC1 through VDC2 to T2?"

The technology for implementing this kind of environment is just beginning to emerge. Forbus and Falkenhainer [1990] have demonstrated the capability to generate interactive explanations of simulations based on qualitative processes. The ability to assemble models from modular component libraries by specifying relevant levels of abstraction and granularity is an ongoing research topic with some promising early results [Fishwick, 1989; Falkenhainer and Forbus, 1991; Iwasaki, 1990]. Causal explanations can be generated from an analysis of behavior equations [Iwasaki and Simon, 1986] and the engineering models (e.g, derived from qualitative events determined by the activation of qualitative processes, or following explicit functional labels [Keuneke, 1989]). Techniques for presenting text and allowing interactive followup questions have been demonstrated by Moore and Swartout [1989]. We are incorporating these ideas into the Device Modelling Environment [Iwasaki et al., 1989].




The voltage limiting circuit VLC

```
"A purpose of the voltage limiting circuit VLC is to prevent the
output voltage V2 at diode bus terminal T2 from exceeding 32.0 V."

The voltage limiter VLC works as follows:

When  the bypass relay S1 is closed,
there is current flow from input terminal T1 through relay S1
      through voltage dropping circuit VDC2 to terminal T2,
and the input voltage V1 = voltage V2 + voltage drop VD2 across
      voltage dropping circuit VDC2.
Since terminal T2 is electrically connected to the terminal
      T3 of the voltage sensing circuit VSC,
the voltage V3 at terminal T3 = voltage V2.
When  output voltage V2 > threshold voltage TV1,
      voltage V3 > threshold TV1,
then  the voltage sensing circuit VSC outputs voltage V4 > 0V at
      terminal T4,
which causes relay S1 to open.
There is now current flow from terminal T1 through voltage dropper
      VDC1 through voltage dropper VDC2 to terminal T2.

Assuming voltage V1 is constant or decreasing,
then  voltage V2 will decrease by at least voltage drop VD1 across
      voltage dropper VDC1.
```

**Figure 6:** Example explanation from a hypothetical simulation

# 4. Comparison of justification elicitation in the two applications

Although the two applications considered in this paper are quite different, their respective knowledge acquisition techniques share some abstract characteristics. In ASK, justifications are enumerations of those features of the current state of the diagnosis, such as cost and therapeutic value, that are relevant to choosing one action over others. In the design rationale technique, justifications are explanations showing how specific features of structure and behavior, such as the subset of components and physical processes involved, are relevant to achieving the intended function. Table 1 summarizes five general characteristics of the approach to justification elicitation, and show how each is applied to the two tasks.

## Abstract Characteristics of Justification Elicitation

| Characteristic | Knowledge Acquisition Task | |
|---|---|---|
| | Strategic Knowledge | Design Rationale |
| Justifications are represented in terms of situations, choices, and their features. | Situations are states of the diagnosis such as hypotheses. Choices are actions such as diagnostic tests. Features are attributes, functions, and relations over objects in a working memory. | Situations are physical structure, requirements, and assumed operating conditions. Choices are design alternatives. Features are models, initial conditions, and predicted behavior. |
| The representation is implemented in a task-specific architecture that can apply justifications to perform some task. | The architecture for prospective diagnosis uses strategy rules to choose among actions at each decision point. Strategy rules can be formed from justifications. | The architecture for model formulation and simulation is for explaining how devices work. Justifications are used to configure simulation scenarios. |
| Examples are elicited in a computational context of use where situations and choices are reflected in the state of the system. | The context is a running knowledge-based system that makes strategic decisions about actions. | The context is an interactive simulation environment that generates explanations. |
| Justifications are elicited by asking for relevant features, selected from a finite set of possible features provided by the system. | The user justifies an action by selecting relevant aspects of the diagnostic state. | The user indicates what is relevant by selecting aspects of structure, behavior, and initial conditions to include in the simulation. |
| Explanations are generated by mapping from relevant features to intended outcome. | Justification statements are generalized into pattern-matching strategy rules that describe similar situations. | An explanation of the simulation relates design features to intended behaviors. |

**Table 1:** Common characteristics of two applications of justification elicitation.

# 5. Discussion

A key source of power for effective automated knowledge acquisition is to divide the cognitive labor, assigning roles to players with appropriate expertise. The task- and method-specific tools reviewed in Section 1 divide the labor between the system architect, who designs the problem-solving method and task-specific representation, the domain specialist, who instantiates that architecture for a particular domain, and the elicitation tool, which provides the interactive medium for building the domain-specific knowledge base. For the approach to acquiring justifications that I have described, the system architect formulates the knowledge to acquire in terms of situations, choices, and features that can be implemented in the computational setting. The elicitation program provides the medium: a computational context of use, an interface for examining and manipulating the features, and a mechanism for generating explanations. The domain specialist supplies the justifications, not by answering why-questions, but by *selecting* what is *relevant* among the possibilities afforded by the computational model. The knowledge acquired using the justification technique is guaranteed to be *operational* because the user can only convey something to the machine by getting the machine to say it.

The major limitation to this approach to knowledge acquisition, which is shared by all automated approaches (including machine learning), is the limited expressiveness of the representation. The machine can only elicit knowledge in terms that have already been operationally defined. Thus a central research challenge is to invent interactive environments in which nonprogrammers can extend the language supported by the knowledge acquisition tool. For task-specific tools, there has been work on meta-tools such as PROTEGE that help developers define the task vocabulary needed by OPAL-class tools [Musen, 1989a, 1989b]. For ASK, there is a term-definition interface that uses the definitions of existing terms in the knowledge bases as exemplars [Gruber, 1989a, 1989b]. For the proposed approach to design knowledge capture, the problem of defining relevant terms is essentially the problem of formulating modular model fragments in a model library. Model formulation is gaining increasing attention in qualitative physics [Addanki, Cremonini, and Penberthy, 1989; Crawford, Farquhar, and Kuipers, 1990; Forbus and Falkenhainer, 1991; Iwasaki et al., 1989] and numerical simulation [Abelson et al., 1989; Fishwick, 1989; Zeigler, Elzas, and Oren, 1989].

# 6. Some of the Related Work

Research on the acquisition of strategic knowledge is thoroughly reviewed in [Gruber, 1989a]. The idea of using simulation to communicate knowledge of how things work is central in the simulation-based training field (e.g., [Brown, Burton, and deKleer, 1982; Forbus and Stevens, 1981; Hollan, Hutchins, and Weitzman, 1984; Towne and Munro, 1989]). An architecture for generating explainable simulation scenarios has been demonstrated in the SIMGEN system [Forbus and Falkenhainer, 1990]. Capturing the underlying design of an artifact in a form suitable for generating human-readable explanations of design intent is being explored in the software domain by the EES project [Neches, Swartout, and Moore, 1985]. The use of machine-generated explanations as an input medium for acquiring operational knowledge is inspired by work in automated knowledge acquisition [Bareiss, 1989; Gruber, 1989a] and human-computer interaction [Yen, Neches, and DeBellis, 1988; Williams, 1984]. The general idea of design knowledge capture by explanation was independently proposed by Bill Mark [Kellog, et al., 1989; Mark, 1988].

## For Further Reading

This was not intended as a survey article, although the ideas touch on several large areas of research. For a good introduction to the literature on Automated Knowledge Acquisition, see the special issue of *Machine Learning*, Volume 4, Numbers 3 / 4, 1989 (also published as *Knowledge Acquisition: Selected Research and Commentary*, edited by S. Marcus, Kluwer Academic, 1990) and recent issues of the journals *Knowledge Acquisition: An International Journal* and *International Journal of Man-Machine Studies*. A good sampling of relevant papers on qualitative physics may be found in *Readings in Qualitative Reasoning about Physical Systems*, edited by Daniel Weld and Johan de Kleer and published by Morgan Kaufmann in 1990, and a forthcoming special issue of *Artificial Intelligence* (expected for Fall 1991). The literature on design rationale is just beginning to emerge; look for a special issue of *Human Computer Interaction* in 1991.

## Acknowledgements

The ASK work was done at the University of Massachusetts with Paul Cohen. The design rationale work is a collaboration with Dan Russell at Xerox PARC and the How Things Work project at the Stanford KSL with contributions from Tilda Brown, Yumi Iwasaki, Bob Engelmore, Ed Feigenbaum, Chee Meng Low, Pandu Nayak, Daneel Pang, and James Rice. Funding was provided by NASA Grant NCC2-537, NASA Grant NAG 2-581 (under ARPA Order 6822), IBM Agreement No. 14780042, and NIH Grant No. LM05208. Brian Falkenhainer, Ken Forbus, Bill Mark, and Brian Williams have been very influential. Thanks to B. Chandrasekaran, Hania Gajewska, and Hiroshi Motoda for helpful reviews.

## References

Abelson, H., Eisenberg, M. Halfant, M. Katzenelson, J., Sacks, E., Sussman, J., Wisdom, J. and Yip, K. (1989). Intelligence in scientific computing. *Communications of the ACM*, 32(5):546-562.

Addanki, S., Cremonini, R., and Penberthy, J. S. (1989). Reasoning about assumptions in graphs of models. *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1432-1438, Detroit.

Alexander, J. H., Freiling, M. J., Shulman, S. J., Rehfuss, S., and Messick, S. L. (1988). Ontological analysis: An ongoing experiment. In J. Boose and B. Gaines (Eds.), *Knowledge Acquisition Tools for Expert Systems, Volume Two*. Boston: Academic Press.

Baudin, C. Sivard, C. and Zweben, M. (1989). A model-based approach to design rationale conservation. *IJCAI-89 Workshop on Model-based Reasoning*.

Bareiss, E. R. (1989). *Exemplar-based Knowledge Acquisition: A Unified Approach to Concept Representation, Classification, and Learning.* Boston: Academic Press.

Buchanan, B. G., Barstow, D. K. , Bechtel, R., Bennett, J., Clancey, W., Kulikowski, C., Mitchell, T., and Waterman, D. A. (1983). Constructing an expert system. In F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, (Eds.), *Building Expert Systems*. Reading, MA: Addison-Wesley.

Breuker, J. A. and Wielinga, B. J. (1985). KADS: Structured knowledge acquisition for expert systems. *Proceedings of the Fifth International Workshop on Expert Systems and their Applications,* Avignon, France.

Breuker, J., Wielinga, B., van Someren, M., de Hoog, R., Schreiber, G., de Greef, P., Bredeweg, B., Wielemaker, J., Billault, J.-P., Davoodi, M., and Kayward, S. (1987). *Model Driven Knowledge Acquisition: Interpretation Models, Esprit Project P1098*. Amsterdam: University of Amsterdam and STL, Ltd.

Brown, J. S., Burton, R., and deKleer, J. (1982). Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II, and III. in *Intelligent Tutoring Systems*. Sleeman and Brown (Eds.), Academic Press.

Bylander, T. and Chandrasekaran, B. (1987). Generic Tasks for knowledge-based reasoning: The "right" level of abstraction for knowledge acquisition. *International Journal of Man-Machine Studies*, *26*(2):231-244.

Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert, 1*(3), 23–30.

Clancey, W. J. (1989). Viewing knowledge bases as qualitative models. *IEEE Expert*, 2(4): 9-23.

Cohen, P. R., Day, D. S., Delisio, J., Greenberg, M., Kjeldsen, R., Suthers, D., and Berman, P. (1987). Management of uncertainty in medicine. *International Journal of Approximate Reasoning*, *1*(1), 103–116.

Conklin, J. and Begeman, M. L. (1988). gIBIS: A hypertext tool for exploratory policy discussion. *Proceedings of the 1988 Conference on Computer Supported Cooperative Work (CSCW-88),* Portland, Oregon.

Crawford, J. M., Farquhar, A., and Kuipers, B. (1990). QPC: A compiler from physical models into qualitative differential equations. *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, August.

Ericsson, K. A. and Simon, H. A. (1984). *Protocol Analysis: Verbal Reports as Data*. Cambridge, MA: MIT Press.

Falkenhainer, B. and Forbus, D. K. (1991). Compositional modeling: Finding the right model for the job. To appear in *Artificial Intelligence*, Fall. Special issue on qualitative physics.

Fishwick, P. A. (1989). Qualitative methodology in simulation model engineering. *Simulation Journal*, 52(3): 95-101.

Fishwick, P. A. (1989b). Abstraction level traversal in hierarchical modeling. In B. P. Zeigler, M. Elzas, and T. Oren (Eds.). *Modelling and Simulation Methodology: Knowledge Systems Paradigms*, Elsevier North Holland, 393–429.

Forbus, D. K. and Falkenhainer, B. (1990). Self-explanatory simulations: An integration of qualitative and quantitative knowledge. *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, July.

Forbus, K. D. and Stevens, A. (1981). Using qualitative simulation to generate explanations. *Proceedings of the Third Meeting of the Cognitive Science Society*, August.

Gruber, T. R. (1989a). *The Acquisition of Strategic Knowledge.* Academic Press.

Gruber, T. R. (1989b). Automated knowledge acquisition for strategic knowledge. *Machine Learning*, 4(3-4), 293–336.

Gruber, T. R., and Cohen, P. R. (1987). Design for acquisition: Principles of knowledge system design to facilitate knowledge acquisition. *International Journal of Man-Machine Studies*, 26(2), 143–159.

Gruber, T. R. and Russell, D. M. (1990). Design knowledge and design rationale: A framework for representation, capture, and use. Stanford Knowledge Systems Laboratory, Technical report KSL 90–45.

Hollan, J. D., Hutchins, E. L., and Weitzman, L. (1984). STEAMER: An interactive inspectable simulation-based training system. *AI Magazine*, 2.

Iwasaki, Y. (1989). Two model abstraction techniques based on temporal grain size: aggregation and mixed models. *Proceedings of the Third Workshop on Qualitative Physics.* Available from Stanford University, Knowledge System Laboratory, report KSL 90-63.

Iwasaki , Y. (1990). Reasoning with multiple abstraction models. *Proceedings of the Fourth International Workshop on Qualitative Physics.* Available from Stanford University, Knowledge System Laboratory, report KSL 90-52.

Iwasaki, Y., Doshi, K., Gruber, T.,Keller, R., and Low, C. M. (1989). Equation model generation: Where do equations come from? *Proceedings of the IJCAI-89 Workshop on Model-Based Reasoning.*

Iwasaki, Y. and Simon, H. (1986). Causality in device behavior. *Artificial Intelligence*, 29:63-72.

Kellog, C., Gargan, R., Mark, W., McGuire, J., Pontecorvo, M., Schlossberg, J., Sullivan, J. Genesereth, M., and Singh, N. (1989). The acquisition, verification, and explanation of design knowledge. SIGART Newsletter, 108, April 1989, 163-165.

Keuneke, A. (1989). *Machine Understanding of Devices: Causal Explanation of Diagnostic Conclusions.* Doctoral dissertation, The Ohio State University, Columbus Ohio.

Lakin, F., Wambaugh, J., Leifer, L., Cannon, D., and Sivard, C. (1989). The electronic design notebook: Performing medium and processing medium. *Visual Computer: International Journal of Computer Graphics*.

Marcus, S. (1988). SALT: A knowledge acquisition tool for propose-and-refine systems. In S. Marcus (Ed.), *Automating Knowledge Acquisition for Expert Systems.* Boston: Kluwer Academic.

Mark, W. (1988). Explanation and interactive knowledge acquisition. *AAAI-88 Workshop on Explanation*.

Mark, W. and Schlossberg, J. (1990). Design Memory. *Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, November.

McDermott, J. (1988). Preliminary steps toward a taxonomy of problem-solving methods. In S. Marcus (Ed.), *Automating Knowledge Acquisition for Expert Systems.* Boston: Kluwer Academic.

Moore, J. D. and Swartout, W. R. (1989). A reactive approach to explanation. *IJCAI-89*, 1504-1510.

Musen, M. A. (1989a). *Automated Generation of of Model-based Knowledge-Acquisition Tools,* London: Pitman.

Musen, M. A. (1989b). Automated support for building and extending expert models. *Machine Learning*, 4(3-4), 347–377.

Musen, M. A., Fagan, L. M., Combs, D. M., and Shortliffe, E. H. (1987). Use of a domain model to drive an interactive knowledge editing tool. *International Journal of Man-Machine Studies*, *26*(1):105-121.

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

Newell, A. (1982). The knowledge level. *Artificial Intelligence, 18*, 87–127.

Shema, D., Bradshaw, J., Covington, S., and Boose, J. (1990). Design knowledge capture and alternative generation using possibility tables in CANARD. *Knowledge Acquisition*, *2*(4):345-364.

Towne, D. M. and Munro, A. (1989). Tools for Simulation-Based Training. Technical report 113, Behavioral Technology Laboratories, USC.

Williams, M. (1984). What makes RABBIT run? *International Journal of Man-machine Studies*, 21:333-352.

Yen, J., Neches, R., and DeBellis, M. (1988). Specification by reformulation: A paradigm for building integrated user support environments. *AAAI-88.*

Zeigler, B. P., Elzas, M. and Oren, T. (Eds.), (1989). *Modelling and Simulation Methodology: Knowledge Systems Paradigms*, Elsevier North Holland.