The Knowledge Engineering Review, Vol. 8:1, 1993, 5–25

Knowledge-acquisition tools with explicit problem-solving models

WILLIAM BIRMINGHAM[1] and GEORG KLINKER[2]

[1]*University of Michigan, Department of Electrical Engineering and Computer Science, Ann Arbor, MI 48109, USA (e.mail: wpb@crim.eecs.umich.edu)*
[2]*Digital Equipment Corporation, 111 Locke Drive, Marlboro, MA 01752, USA (e.mail: klinker@airg.dec.com)*

Abstract

In the past decade, expert systems have been applied to a wide variety of application tasks. A central problem of expert system development and maintenance is the demand placed on knowledge engineers and domain experts. A commonly proposed solution is knowledge-acquisition tools. This paper reviews a class of knowledge-acquisition tools that presuppose the problem-solving method, as well as the structure of the knowledge base. These explicit problem-solving models are exploited by the tools during knowledge-acquisition, knowledge generalization, error checking and code generation.

1 Introduction

Existing expert systems have shown that artificial intelligence (AI) techniques can be used to solve tasks whose solutions depend on substantial, specialized knowledge. For example, Prospector (Duda, 1984) assists geologists in mineral exploration; R1 (McDermott, 1981) (now called XCON) configures computer systems automatically; and Dendral (Buchanan, 1978) identifies organic compounds from mass-spectroscopy data.

Engineering of an expert system poses a number of issues, such as:

- Expert systems tackle problems that are knowledge intensive. Acquiring, organizing, and developing representational schemes for knowledge is a lengthy and tedious process. Great demands are placed on knowledge engineers to develop knowledge bases of sufficient depth to enable expert systems to achieve high performance.
- A communication gap exists between the knowledge engineer and the domain expert. To build an expert system, the knowledge engineer must learn a great deal about how the domain expert performs his tasks. On the other hand, the domain expert often finds it difficult, or even impossible, to explain her reasoning process. The transfer of knowledge, therefore, from expert to knowledge engineer often is slow and is susceptible to errors.
- Expert systems are generally implemented in specialized languages that use nontraditional programming paradigms. Mastering these languages requires extensive training.
- Expert systems are often used in areas where knowledge is constantly unfolding. Thus, maintaining the high performance of an expert system requires continually updating the knowledge base.

Knowledge-acquisition tools address these issues. They provide a high-level interface that abstracts from the implementation details of the expert system. In fact, some tools have sufficient knowledge of a domain to allow a domain expert to build a knowledge base without the assistance of a knowledge engineer. Furthermore, knowledge-acquisition tools can facilitate maintenance by allowing easy access to a knowledge base for adding new, and deleting outdated, knowledge.

To date, numerous knowledge-acquisition tools have been developed. They cover a wide range of undertakings, ranging from general-purpose tools to those that are specialized for one aspect of the knowledge-engineering task.

In this paper, we describe a class of knowledge-acquisition tools with explicit problem-solving models. We call these tools *specialized knowledge-acquisition tools*. Specialized knowledge-acquisition tools exploit a model of the expert system for which they are acquiring knowledge. The model represents both the problem-solving behaviour, or problem-solving method, and the structure of the expert system's knowledge base. A problem-solving method defines the processes for solving a problem. It relies on the knowledge base to provide the specific knowledge needed to solve the problem at hand. Based on the model, knowledge-acquisition tools can engage in dialogue with an expert to acquire knowledge, to generalize this knowledge, to check it for errors, and then to operationalize it for use by the expert system.

In section 2, we relate specialized knowledge-acquisition tools to tools resulting from other approaches to knowledge-acquisition. The characteristics of specialized knowledge-acquisition tools are summarized in section 3. Section 4 describes the problem-solving models of these tools. Section 5 explicates their knowledge-acquisition process. Section 6 presents data on scope, performance, and actual uses. Finally, section 7 discusses alternative approaches to knowledge acquisition.

2 Use of specialized knowledge-acquisition tools in the lifecycle of expert systems

Developing a knowledge-based system includes performing a task analysis, designing a problem-solving approach and knowledge base, and populating that knowledge base with information required by the problem-solving approach. Knowledge-acquisition tools developed to date support only a subset of these activities.

In sections 2.1–2.3, we concisely characterize existing knowledge-acquisition tools and approaches with respect to the knowledge-engineering activities they support. Our intention is not to provide a complete taxonomy.[1] See Boose (1989) and Westphal (1989) for a comprehensive compilation of existing knowledge-acquisition tools. The characterizations given here outline current approaches with the goal of showing when specialized knowledge-acquisition tools can be useful in the lifecycle of expert systems.

2.1 Performance of a task analysis

When presented with a new task, the knowledge engineer must perform a task analysis to determine the relevant task-specific characteristics. The characteristics include a domain's essential concepts and vocabulary, the activities involved in the task, the resources consumed and the products produced, and the agents performing the activities. Conceptual frameworks for performing such analysis include Newell's knowledge level (Newell, 1982). The objective of a task analysis is to determine the essential knowledge required for a task, without considering the way in which the task will be performed. For example, Balkany et al. (1991) identify the functions needed to perform design (constructive) tasks by performing successive decomposition. The decomposition proceeds until a set of operators are found that operate on a predetermined, albeit general, knowledge structure. By predefining the knowledge structure, the grain size of the operators is uniform. The authors indicate that performing knowledge-level analyses can reduce system

[1] We consider examples of those tools developed by the knowledge-acquisition community with the explicit goal of supporting the development of knowledge-based systems. Also, we do not consider here tools for and approaches to providing a support environment for integrating all of a knowledge engineer's activities. See section 7 for a brief description of some of those support environments.

development by identifying knowledge that can be reused from existing systems and combined in different ways.

Several tools assist a user in uncovering initial concepts, as well as in identifying structural and functional relationships for the domain. These tools can be used as part of a comprehensive task analysis. For example, ETS (Boose, 1984) and KSS0 (Shaw et al., 1987) are tools based on personal-construct theory. They assist users in determining structural relationships and relevant terms and concepts using the repertory-grid technique. They place terms on a grid and interact with a user to identify clusters. As clusters are found, concepts and relationships among them are determined. Kriton (Diederich et al., 1987) combines a set of knowledge-elicitation techniques, such as text analysis, to uncover initial concepts and structural relationships; interview techniques to refine the results of the text analysis; and protocol analysis to determine functional relationships. SOK (Linster, 1989) extends Kriton with integration of additional elicitation techniques, and with a modelling component that describes the problem-solving process.

2.2 Design of a problem-solving approach and knowledge base

Based on the task analysis, the knowledge engineer designs an approach to perform the task and a representation scheme for the knowledge required by the problem solver. Some work in knowledge-acquisition supports a knowledge engineer with this design task. That work is mainly concerned with developing conceptual frameworks to guide the implementation process. For example, Steels (1990) introduces a comprehensive framework for translating a knowledge-level description to code, i.e., his framework describes how to design a knowledge-based system from a knowledge-level analysis. Design activities are performed at a level in between the knowledge level and the code level; Steels calls this level the *knowledge-use* level. His framework integrates different aspects of the design task: decomposing a task into subtasks, categorizing problem-solving approaches and the kinds of knowledge they require, and organizing domain-specific knowledge.

Other researchers have developed frameworks that focus on specific aspects of the design task. For example, the generic-tasks approach (Chandrasekaran, 1990) describes how to decompose an application into general task categories. McDermott (1988) emphasizes the problem-solving approach of an application, and describes a variety of problem-solving methods and knowledge types required to support the method. Balkany et al. (1991) take a similar approach; they have reported an initial set of building blocks for problem-solving methods that are relevant for configuration design. They derived this set by analysing the operation of existing design tools. About 50 functions have been identified to date.

Another group of researchers has made progress characterizing individual, problem-specific problem-solving approaches and the tasks that such approaches can solve. Examples are approaches to solve reporting tasks (Klinker et al., 1989), skeletal-plan refinement tasks (Friedland et al., 1989), classification tasks (Clancey, 1985), and configuration-design tasks (Runkel et al., 1992).

2.3 Population of a knowledge base

Once the problem-solving approach and the structure of the knowledge base are defined, the knowledge base needs to be populated. Specialized knowledge-acquisition tools assist users with this task. Examples of specialized knowledge-acquisition tools are Mole (Eshelman et al., 1987) for selective tasks; Burn (Marques et al., 1988) for computer-system sizing tasks; Salt (Marcus, 1988) and CGEN (Birmingham, 1988) for constructive tasks; Knack (Klinker, 1988) for reporting tasks; and Opal (Musen, 1989a) for the task of entering cancer-therapy protocols. All of the tools that are listed above have the explicit goal of interacting directly with a domain expert. Sections 3–6 describe these tools in detail.

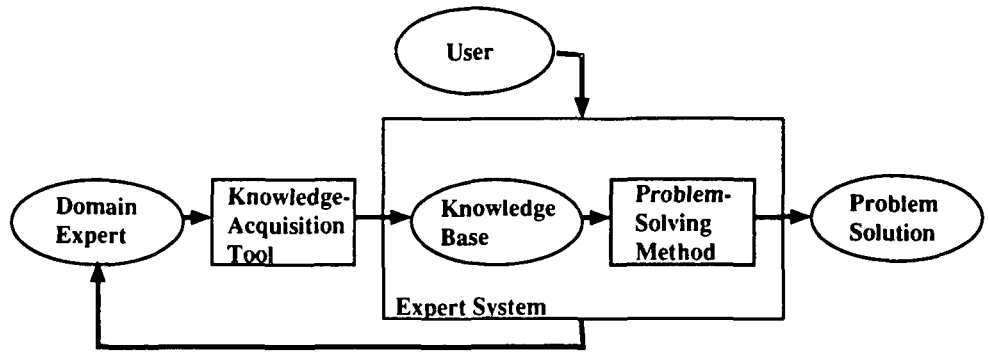

Figure 1 Model for knowledge-acquisition

3 Overview of specialized knowledge-acquisition tools

Figure 1 characterizes the knowledge-acquisition process. It presents an overview of the relationship between a knowledge-acquisition tool and an expert system. A domain expert, or knowledge engineer, or both, use a knowledge-acquisition tool to generate a knowledge base. A problem-solving method uses the knowledge base and additional information from a user to produce a solution. Since the performance of an expert system is dependent on its knowledge base, the quality of solutions can be used to assess the quality of the knowledge base. Whenever an expert system reveals inadequacies, the knowledge engineer, or domain expert, uses the knowledge-acquisition tool to improve the knowledge base. In those cases where the knowledge-acquisition tool cannot be used, the knowledge engineer manually enhances the expert system.

Specialized knowledge-acquisition tools exhibit the following additional characteristics:

- They presuppose a problem-solving method, as well as the structure of a knowledge base; i.e., they exploit a model of the expert system they generate.
- They acquire knowledge, generalize the acquired knowledge, check it for errors, and then generate code.

Presupposing the problem-solving method and the structure of the associated knowledge base provides the tools with significant power to interface directly with the domain expert by using a task-dependent terminology, to guide the dialogue with the domain expert by exploiting the predefined structure of the knowledge base, and identify and remedy problems within the knowledge base by exploiting its structure.

The following sections describe specialized knowledge-acquisition tools in more detail. We use six existing tools to demonstrate common characteristics. The tools have all been used to assist with the creation of expert systems, and have been documented in the literature.

- **Mole** (Eshelman, 1987) was used to generate expert systems to diagnose:
  - — problems in a car engine
  - — problems in a steel-rolling mill
  - — inefficiencies in a coal-burning power plant.
- **Burn** (Marques, 1988; 1991) was used to generate expert systems to:
  - — estimate the pieces of various hardware needed to satisfy the computational demands of a prospective client's organization
  - — select software tools to meet a client's computer needs
    - — estimate the pieces of various host and desktop hardware resources for a prospective client's organization
  - — determine the pieces of various hardware resources of a prospective client for a specific software system
  - — recommend application software for a client's organization.

- **Salt** (Marcus, 1988) was used to generate expert systems to:
  - — configure elevator systems
  - — schedule tasks in an engineering department.
- **CGEN** (Birmingham, 1988) was used to create expert systems to design:
  - — workstation and PC-class computers
  - — simple mechanical devices, window regulators, to raise and lower the windows in a car door (Birmingham, 1989).
- **Knack** (Klinker, 1988) was used to generate expert systems to:
  - — report on designs of electromechanical systems that may be suboptimal from a specified perspective (three expert systems for different stages in the design of electromechanical systems)
  - — assist a project leader with the creation of project proposals (three expert systems for different tasks of a project leader)
  - — assist analysts with the definition of requirements for software systems
  - — assist an entrepreneur in preparing marketing plans for a proposed business.
- **Opal** (Musen, 1989 a) was used to generate expert systems to:
  - — assist physicians caring for cancer patients being treated under clinical protocols (Shortliffe, 1986).

Table 1 summarizes the characteristics of the example tools. Sections 4 and 5 expand the table entries.

Presupposition of a problem-solving method and associated knowledge roles

Both the problem-solving method and the structure of the knowledge base are the result of a detailed task analysis. Specialized knowledge-acquisition tools do not assist a user with performing that analysis. They presuppose the problem-solving method of the expert systems that they generate or extend (McDermott, 1988; Gruber, 1987).

A problem-solving method establishes and controls the sequences of actions required to perform some task. This control knowledge dynamically defines the order in which subtasks have to solve the overall task. It also defines the kind of domain-specific knowledge that is applicable within each subtask, thereby making explicit the different roles knowledge plays. Thus, a problem-solving method embodies the control knowledge needed to apply the domain knowledge. Considered from a different perspective, the time at which domain knowledge will be brought to bear is determined solely by that knowledge's role.

A problem-solving method represents computational processes, and defining these processes is the chore of programmers. If domain experts are nonprogrammers, they do not know how to define computational processes. By presupposing the problem solver, the knowledge-acquisition tool can generate an expert system that embodies the problem solver, thereby freeing the domain expert from having to implement it. Of course, a programmer must define the problem-solving method for the knowledge-acquisition tool. Once he has done that, many expert systems can be constructed using the defined method.

The knowledge roles determine the structure of the knowledge base, and establish a vocabulary that can be used during knowledge-acquisition to guide a domain expert or knowledge engineer in defining, analysing and testing a knowledge base. By defining a vocabulary familiar to the expert, knowledge roles help her in the knowledge-elicitation process. Furthermore, these roles allow the knowledge-acquisition tool to develop expectations about the knowledge the expert might provide, and thus can guide the elicitation session, i.e., the tool can ask specific questions.

The problem-solving methods and their associated knowledge roles presupposed by the tools introduced in the previous section are described next.

4.1 Mole

1Mole is a tool for building diagnostic expert systems. For example, systems for diagnosing car engine problems or diagnosing problems in a steel-rolling mill. A Mole-generated expert system

Table 1 Overview of example tools' characteristics

Problem-solving method	*Knowledge roles*	*Knowledge gathering*	*Generalization*	*Error checking*	*Code generation*
Mole: Ask the user for symptoms that are present, propose a set of alternatives or explanations for each symptom, and then query the user about information that will help to differentiate these alternatives.	Complaints, explanations, differentiation knowledge	Questions are used to elicit complaints first, then explanations for the complaints, and finally information differentiating among explanations.		The user indicates than a result is wrong, and Mole suggests remedies.	Data structures that are interpreted by OPS5 code
Burn: Elicit information about a particular sizing problem from the user; identify other, similar, already solved problems in a knowledge base of cases; and use the differences between the solved and unsolved cases to extrapolate from a known solution to a new one.	Case library, indexing knowledge, extrapolation knowledge	Given an incorrect solution, questions and forms are used to add cases, indexing knowledge, and extrapolation knowledge.		The user indicates that a result is wrong, and updates the case, indexing, or extrapolation knowledge.	Data structures that are interpreted by OPS5 code
Salt: Construct an approximate plan or design by proposing a value for one parameter of the design at a time, check whether each parameter satisfies all constraint on it, and revise past decisions whenever constraint violations are detected.	Design-extension knowledge, identifying-a-constraint knowledge, fix knowledge	Forms are used first to acquire design and constraint knowledge. Salt then uses forms to elicit fix knowledge for each defined constraint.		Salt checks for incompleteness and inconsistencies while acquiring knowledge, and asks the user to resolve any such findings.	OPS5 code
CGEN: Query the user for functional, cost, and performance goals of the system to be designed; refine these specifications using a hierarchical, stepwise-refinement process, where user-given specifications are successively decomposed until components can be selected from a library, and appropriate structures are selected to integrate these components.	Part-function knowledge, selection knowledge, structural knowledge, calculation knowledge	Give part-function knowledge, questions are used to acquire selection and calculation knowledge. Structural knowledge is defined through a graphical interface.	CGEN generalizes structural knowledge.	CGEN checks whether selection, structural and calculation knowledge was provided for each part. If none was, it asks the user to provide that knowledge.	OPS83 code
Knack: Identify all pieces of information that are appropriate to acquire, select one piece to gather and a strategy for acquiring it, apply the selected strategy, integrate that information with whatever information is already there, and produce a report documenting the acquired information.	Report knowledge, strategy knowledge, domain model	Free text is used to define report knowledge, forms are used to acquire strategies, and a graphical interface is used to elicit the domain model.	Knack generalizes report and strategy knowledge.	The user indicates that a result is wrong, and updates the report and strategy knowledge and the domain model.	OPS5 code
Opal: Follow the procedures specified in a standard plan, specialize that plan with parameters that are specific to the current case, and modify the plan periodically in response to the observed reactions to the earlier actions.	Standard plan actions, sequencing knowledge, refinement knowledge	Given standard plan actions, a graphical interface is used to define sequencing knowledge. Forms are used to elicit refinement knowledge for the relevant plan actions.		Opal checks for incompleteness and inconsistencies, and asks the user to resolve any such findings.	OZONE code

asks the user for symptoms that are present, proposes a set of alternatives or explanations for each symptom, and then queries the user about information that will help to differentiate these alternatives.

The knowledge roles of a Mole-generated expert system are:

- Complaint knowledge tells a user that there is a problem to be diagnosed.
- Explanation knowledge identifies states or events that will explain or cover a complaint.
- Differentiation knowledge differentiates candidate explanations.

4.2 Burn

Burn is the successor of Sizzle (Offutt, 1988; Marques et al., 1988). It builds expert systems that reason quantitatively and use a case-based reasoning method to solve sizing problems, e.g., sizing the requirements for a computer system. A Burn-generated expert system elicits information about a particular sizing problem from the user; it identifies other, similar, already solved problems in a

knowledge base of cases; and it uses the differences between the solved and unsolved cases to extrapolate from a known solution to a new one.

The knowledge roles of a Burn-generated expert system are:

- Case-library knowledge describes previously sized problems, as well as their solutions.
- Indexing knowledge defines a similarity metric that rates the similarity of each case to the new sizing problem to be solved.
- Extrapolation knowledge indicates how to adjust the solution to the most similar case for the sizing problem to be solved.

4.3 Salt

Salt builds expert systems that perform constraint-satisfaction tasks, e.g., configuring an elevator system or scheduling tasks in an engineering department. A Salt-generated expert system constructs an approximate plan or design by proposing a value for one parameter of the design at a time, checks whether each parameter satisfies all constraints on it, and revises past decisions whenever constraint violations are detected.

The knowledge roles of a Salt-generated expert system are:

- Design-extension knowledge contains procedures for obtaining initial values for pieces of the design.
- Identify-a-constraint knowledge contains procedures for obtaining any constraints on those values.
- Fix knowledge contains local remedies for violated constraints.

4.4 CGEN

CGEN constructs expert systems for hierarchical configuration tasks, where it is necessary to choose appropriate components and structures to integrate those components. CGEN-generated expert systems query the user for functional, cost and performance goals of the system to be designed. These specifications are then refined using a hierarchical, stepwise-refinement process, where user-given specifications are decomposed successively until components can be selected from a library, and appropriate structures are selected to integrate these components.

The knowledge roles of a CGEN-generated expert system are:

- Part-function knowledge describes the functional and other attributes (cost, power, reliability, etc.) of the parts.
- Selection knowledge provides constraints for choosing among parts.
- Structural knowledge defines the possible connections among parts.
- Calculation knowledge supplies the formulae to create new part specifications, and to propagate constraints.

4.5 Knack

Knack generates expert systems that assist a user in gathering information from a variety of sources, and in combining that information into a document, e.g., writing proposals and documenting design decisions. A Knack-generated expert system identifies all pieces of information that are appropriate to acquire, selects one piece to gather and a strategy for acquiring it, applies the selected strategy, and integrates that information with whatever information it already has. A Knack-generated expert system produces a report documenting the acquired information.

The knowledge roles of a Knack-generated expert system are:

- Report knowledge contains knowledge that determines the chapters, sections and subsections that are part of a document, and their order. If further defines the report fragments that may be

included into the report. Finally, report knowledge contains information about placement of those fragments within the report.
- Strategy knowledge supplies knowledge that associates a set of acquisition strategies with each variable text component. This knowledge further describes how to integrate the newly acquired piece of information with the existing information.
- Domain-model knowledge defines the concepts and terms relevant to the applications, as well as their interdependencies.

4.6 Opal

Opal is a tool for building expert systems that perform skeletal-plan refinement tasks, e.g., assisting physicians with the treatment according to protocols of patients who have cancer. An Opal-generated expert system follows the procedures specified in a standard plan, specializes that plan with parameters that are specific to the current case, and modifies the plan periodically in response to the observed patient reactions to the earlier actions.

The knowledge roles of an Opal-generated expert system are (Tu, 1989):

- Standard plan actions contain knowledge describing the components and structural relationships of cancer-treatment plans.
- Sequencing knowledge specifies that ordering of the cancer-treatment plan actions.
- Refinement knowledge specializes cancer-treatment plan actions for the current case.

5 Knowledge-acquisition

Specialized knowledge-acquisition tools perform the following functions: (1) acquire knowledge; (2) generalize the acquired knowledge; (3) check it for errors; and (4) generate code. Typically, a knowledge-acquisition tool iterates through the functions of knowledge gathering, generalization, and error checking until it is satisfied with the acquired knowledge, or until the user quits. For some tools, the knowledge gathering and error checking functions may be so closely related that they are virtually indistinguishable. Once the knowledge base is acquired, generalized and corrected, the tool operationalizes it, i.e., the tool generates code. The remainder of this section discusses these functions in more detail.

5.1 Knowledge gathering

The objective of a knowledge-acquisition tool during the knowledge-gathering phase is to obtain information from the user. To reduce the burden placed on the user, the tool must abstract the details of the problem solver and knowledge base, and present an interface that allows efficient transfer of knowledge. It meets these goals by communicating with the user using task-dependent terminology. Thus, specialized knowledge-acquisition tools have interfaces specific to their task to provide the most assistance to the user.

The process of gathering knowledge is directed by the knowledge-acquisition tool. Depending on several factors, including the level of experience of the user and the amount of knowledge to be acquired, the tool may select among different strategies to acquire knowledge from the user. For example, the tool might engage in a dialogue with a user, or might automatically infer additional information from information already available to it. In either case, the gathering process is structured by the knowledge roles. The knowledge-acquisition tool will direct its questions, or inferences, specifically for these roles. The domain primitives chosen for communication, therefore, relate specifically to the knowledge roles. In the examples presented later in this section, notice how the answers to questions map directly to the knowledge roles listed in section 4. The result of this phase of the knowledge-acquisition process is a representation of the newly acquired knowledge in the form of implementation primitives.

```
List possible complaints or symptoms that might need to be
diagnosed

Complaint:
>> [ NONE ] loss-in-gas
```

Figure 2 Mole: acquiring initial symptoms

```
What is a better value for megabytes of
system disk space? [ 60.0 ]: 100

What is a better value for pages per minute of
printing? [ 12.0 ]: 20
```

Figure 3 Burn: correcting a sizing solution

```
1 Name:            CAR-JAMB-RETURN
2 Precondition:    DOOR-OPENING = CENTER
3 Procedure:       CALCULATION
4 Formula:         [ PLATFORM-WIDTH - OPENING-WIDTH ] / 2
5 Justification:   CENTER-OPENING DOORS LOOK BEST WHEN CENTERED ON
                   THE PLATFORM.
```

Figure 4 Salt: Providing design-extension knowledge

In the following we provide an impression of the user interfaces of the example knowledge-acquisition tools.

5.1.1 Mole

Mole begins a new knowledge-acquisition session by asking an expert to list a few complaints or symptoms that would tell a potential user that there is a problem to be diagnosed, such as, 'loss-in-gas' or 'high-fly-ash-flow' (complaints). Mole presents itself to the user via a list of questions providing appropriate default answers. Figure 2 gives an impression of Mole's interface; it demonstrates Mole creating a complaint.

After the initial symptoms have been defined, Mole asks for states or events that will explain the symptoms; for example 'low-heat-transfer' explains 'loss-in-gas' (explanations). This process continues for higher-level explanations. Once the initial explanation space has been built, Mole asks for information to determine how to differentiate among candidate explanations for cases where there are multiple explanations for a symptom (differentiation knowledge).

5.1.2 Burn

Burn uses a dynamic approach to extend its knowledge base. A user poses sizing tasks to an existing sizer, and evaluates its proposed solution. If the sizer performs inappropriately, then the user can add knowledge to the sizer's knowledge base (case-library, indexing, and extrapolation knowledge). For this purpose, Burn presents its solution to the user in the form of questions like those in Figure 3.

Burn attaches the corrected solution to the current sizing case, and adds that solution to its knowledge base, thus, creating a new case.

5.1.3 Salt

In acquiring knowledge, Salt uses a bottom-up strategy, asking the user to define the pieces of knowledge that are determined by Salt's knowledge roles. For example, Figure 4 demonstrates a user providing design-extension knowledge.

```
calculation_type: calculation
calculation: MEMORY_ACCESS_TIME = 4 * BUS_CLOCK_RATE -
             DRIVER_DELAY - PROCESSOR_SET_UP_TIME -
             MEMORY_DEVICE_SET_UP_TIME - SKEW_DELAY
```

Figure 5 CGEN: example calculation for memory-access time

From the acquired knowledge, Salt builds a parameter network. The network consists of variables, which are interrelated with procedures, constraints and fixes. Procedures describe how to determine values for variables; constraints limit the values that variables can take; and fixes describe how values can be changed in case constraints are violated.

Salt uses a similar, form-based interface to acquire identify-a-constraint and fix knowledge.

5.1.4 CGEN

CGEN's task is to acquire hardware-design knowledge. The requisite knowledge is obtained from designers who have no experience in knowledge engineering. CGEN abstracts the details of the expert system's implementation by providing an interface that is natural for a designer to use. All knowledge is acquired via schematic drawings and a simple programming language.

The design domain, especially in the computer area, is intricately tied to technology. Design techniques must be upgraded constantly to incorporate new integrated circuits (IC). Therefore, CGEN must build an initial knowledge base with today's IC technology, and must make incremental additions to that knowledge base as technology improves over many years. The need for incremental addition forms the basis for the organization of the knowledge base, and for the way knowledge is acquired from the domain expert.

Before describing CGEN's knowledge-gathering operation, we will discuss its knowledge-acquisition process. The process is initiated by the expert system, since the latter can determine whether the knowledge base is inadequate for a particular design situation. Other systems, such as Teiresias (Davis, 1981), use a similar type of *in situ* knowledge-acquisition. Once a part is chosen, the expert system will attempt to find corresponding structural knowledge for it. If it cannot find appropriate structures, the expert system creates an initial set of preconditions for the structural knowledge to be acquired, tells the user that CGEN is being invoked, suspends itself, and waits for new knowledge to be compiled into its knowledge base. At this point, the domain expert interacts with CGEN to generate the necessary inputs (schematics and equations).

The designer interfaces with CGEN via two different strategies, depending on the type of knowledge base acquired:

1. Schematic drawings: structural knowledge is conveniently described via schematic drawings produced on a companion schematic-drawing tool.[1]
2. Equation language: equations are written in a simple language, as illustrated in Figure 5. The equation language is used to acquire selection and calculation knowledge.

Part-function and structural knowledge is stored in a database. A separate interface, not associated with CGEN, is used to acquire that information.

5.1.5 Knack

Knack uses a sample-driven approach to acquiring knowledge. This approach is summarized in Figure 6.

Knack expects the expert to communicate a portion of her knowledge as a sample report and a domain model. The sample report is a document that exemplifies the report that a Knack-generated expert system (a WRINGER) is expected to produce (report knowledge). Figure 7 illustrates a small part of an actual sample report.

[1] CGEN actually uses a net list (a list of connections without topological information) that is generated automatically by the schematic-drawing tool.

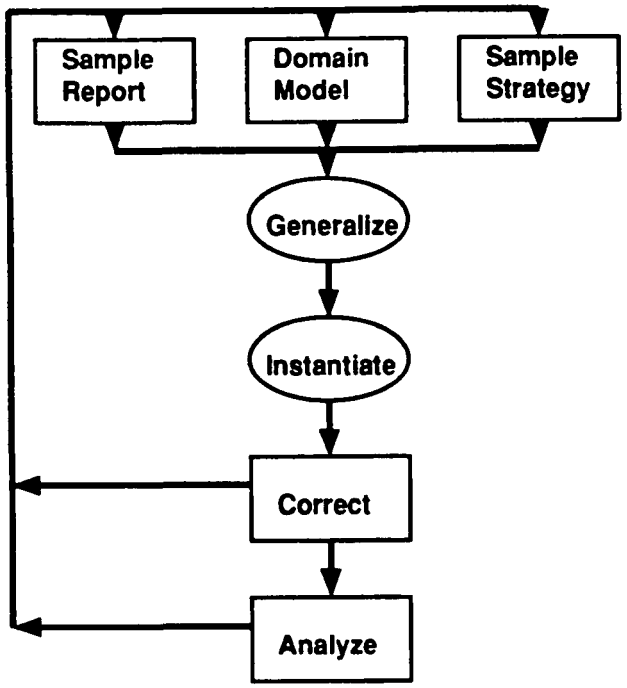

Figure 6 Knack's approach to knowledge-acquisition

```
The system Communications Unit consists of a Computer, a Modem, a
Radio, and a Motor  Generator. Power is supplied from the Motor
Generator to the Computer, Modem, and Radio by the Power Cable.
```

Figure 7 Knack: a fragment of a sample report

The sample report describes a particular application in terms familiar to the expert. To generalize the sample report (see section 5.2), thereby making it applicable to other applications, Knack needs a model of the new domain. The model custom-tailors Knack for a particular domain by using the proper parlance when communicating with the domain expert. The expert uses a graphical editor to define the domain model. Figure 8 demonstrates a part of the domain model needed to generalize the piece of the sample report shown in Figure 7. Knack's acquisition of strategy knowledge is described in section 5.3.

5.1.6 Opal

Opal contains an explicit model of its episodic skeletal-plan refinement problem-solving method, as well as the components of a cancer-therapy protocol, i.e., Opal's model includes domain-specific concepts, such as chemotherapy, drug, toxic reaction, and laboratory test. Thus, those terms and relationships form the basis for a language that is used by both the program and the expert to describe cancer-therapy instances.

Opal's user interface is exemplified in Figure 9. It provides icons representing the standard processes of cancer care. That is, the knowledge about the standard plan actions is predefined in Opal.[2] To define the sequencing actions (sequencing knowledge) for the standard plan actions, the domain expert—a medical doctor, in this case—describes various oncological protocols by manipulating these icons.

[2] It is defined by knowledge engineers using the Protégé tool (Musen, 1989b).

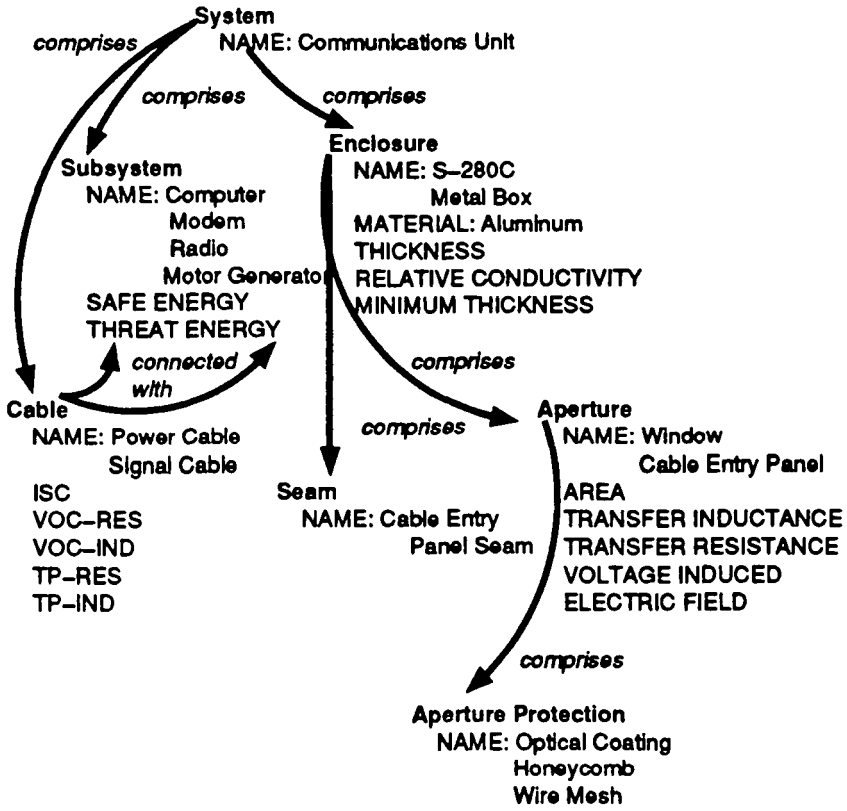

Figure 8 Knack: part of a domain model

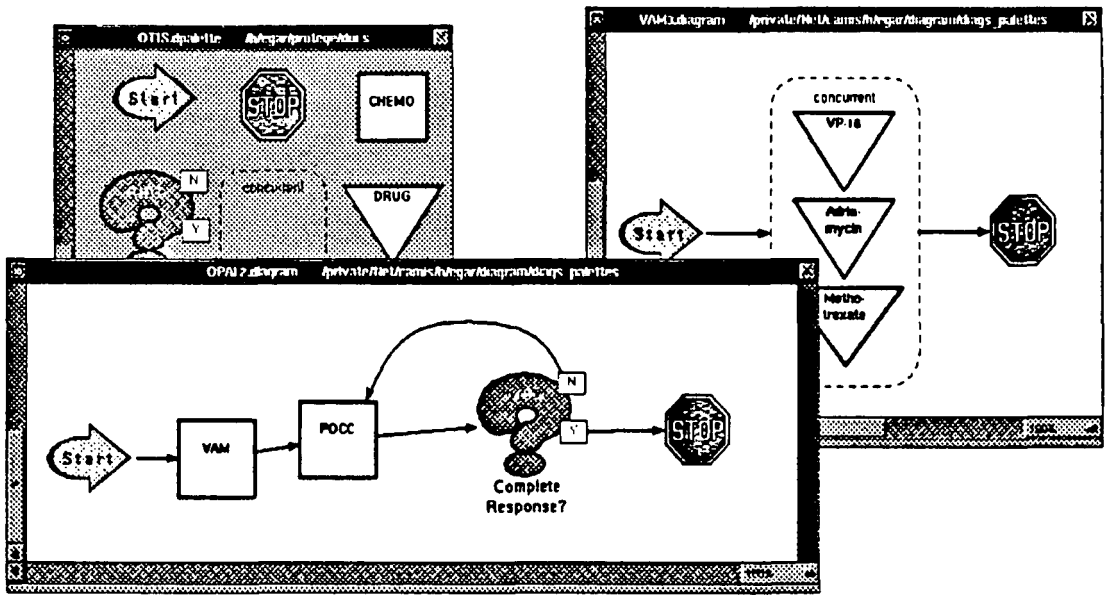

Figure 9 Opal's icon editor

To specialize plan actions for a given situation, Opal provides predefined forms that the user fills in (refinement knowledge).[3]

5.2 Generalization

Often, specific instances of knowledge can be generalized to apply to new situations. To generalize the acquired knowledge, certain knowledge-acquisition tools employ techniques borrowed from

[3]Again, the forms are defined by the knowledge engineer using Protégé.

machine learning; for example, the Vexed system used explanation-based generalization (Mitchell et al., 1985). Other tools use techniques specific to their knowledge-acquisition task; for example, Knack replaces specific terms with variables denoting the terms's concepts.

Among the tools discussed in this paper, CGEN and Knack have an explicit generalization step as part of their approach to acquiring knowledge.

5.2.1 CGEN

The structural knowledge acquired by CGEN is characterized by a set of preconditions, which describe when that structure should be used relative to others. Often, these preconditions are too constraining, thereby preventing the knowledge from being applied in situations where it is perfectly valid. This increases the knowledge-acquisition burden, drastically reducing the effectiveness of CGEN.

The key to CGEN's generalization process is identifying overly constraining preconditions and relaxing them. The generalization process requires its own knowledge base, which cites certain preconditions, the methods for relaxing them, and the conditions when relaxation is allowed. These methods are then applied to newly acquired knowledge. CGEN's generalization knowledge base contains about 100 rules.

5.2.2 Knack

Once the sample report is entered and an initial domain model is defined (see section 5.1), Knack interacts with a domain expert to generalize the sample report. It creates a generalized report by replacing specific terms with the concepts that they represent. The process employs simple heuristics to infer the concepts used. The heuristics are based on keywords, representatives of concepts, and knowledge of relations between candidate concepts contained in the domain model. Figure 10 shows the generalized fragment from Figure 7.

In the example of Figure 7, 'Communications Unit' is recognized as representative of <SYSTEM.NAME>; 'Power Cable' in Figure 7 corresponds to <CABLE.NAME>. Knack replaces the representatives with variables, as shown in Figure 10. A list of representatives for a concept is replaced with text that contains a variable for that concept, surrounded by a LOOP structure. For example, 'a Computer, a Modem, a Radio, and a Motor Generator' is assumed to be a list <SUBSYSTEM.NAME>. The LOOP structure identifies the beginning and end of the text to be repeated, and gives the name of the variable over which the WRINGER should loop. In a WRINGER report, the text outside a LOOP structure will be printed once, whereas the text within the structure will be repeated for each instantiation of the variable that the WRINGER user supplies.

5.3 Error checking

During error checking, the newly acquired knowledge is translated into a declarative representation, an intermediate form used by the knowledge-acquisition tool. In other words, the domain primitives resulting from the knowledge-gathering phase are transformed into implementation primitives understood by the knowledge-acquisition tool. To ensure the integrity of the knowledge, the tool may resume a dialogue with the knowledge engineer or domain expert if the acquired knowledge is found to be incomplete or incorrect. The types of errors that can occur are:

```
The system <SYSTEM.NAME> consists of *LOOPOVER* <SUBSYSTEM.NAME> a
<SUBSYSTEM.NAME> ,*ENDLOOP*. Power is supplied from the <SUBSYS-
TEM.NAME> to the *LOOPOVER* <SUBSYSTEM.NAME> <SUBSYSTEM.NAME>,
*ENDLOOP* by the <CABLE.NAME>.
```

Figure 10 Knack: A generalized report fragment

- Incorrect knowledge: the knowledge provides an incorrect solution to the domain expert. For example, in the domain of blood-disease diagnosis, the expert system could wrongly identify a disease from the data provided. The problem needs to be corrected by the expert.
- Incomplete knowledge: the knowledge provides only a partial solution for a task, and thereby prevents the problem solver from achieving a complete solution. For example, a design system may require knowledge to select both a component and to interconnect it. This requirement is defined by the system's knowledge roles. If the expert provides only selection knowledge, the design system will fail to find a complete configuration, since the component will never be integrated into the design.
- Inconsistent knowledge: the new knowledge is not consistent with previously acquired knowledge. Inconsistent knowledge is not necessarily incorrect; it can indicate, among other things, that underlying assumptions are no longer valid. To illustrate this type of error, let us suppose that the domain is computer design, and that the knowledge-acquisition tool understands memory and processors to exist in physically separate subsystems. When an expert tries to describe a new type of integrated circuit that integrates both functions, a contradiction in the tool's model arises. The confusion arises from a technological change in the domain that violates one of the system's basic assumptions.

A specialized knowledge-acquisition tool uses knowledge roles and feedback from the domain expert to determine whether its knowledge base is correct. The following examples demonstrate that.

5.3.1 Mole

Mole uses a dynamic approach to discover whether the knowledge base is missing relevant knowledge: it refines its knowledge base in the context of providing diagnoses. The user gives Mole a test case, and tells Mole the correct diagnosis. If Mole computes an incorrect result, it tries to determine the source of the errors and recommends possible remedies. The knowledge base can be incomplete because events or states are missing that would explain a symptom, or because Mole cannot differentiate among candidate explanations. The user's test case and feedback provide Mole with the context to analyze and remedy the problem.

5.3.2 Burn

Burn's dynamic approach to extend its knowledge base has been described in section 5.1.2.

5.3.3 Salt

Salt simultaneously acquires knowledge and checks for incompleteness and inconsistencies. When problems are found, the domain expert is asked to help resolve them. For example, Salt works to create a complete knowledge base by requesting a user to define a procedure for each parameter in its network, to supply constraints for any nonconstraint value, and to specify fixes for a violation of any constraint identified.

Inconsistencies can arise, for example, when one formula must have the results of another formula before it can be evaluated, and *vice versa*. Salt detects this situation, which is indicated by a cycle in the constraint network, and asks the user to specify reasonable initial values for some of the variables. Problems might also occur when two procedures provide values for the same variable. Again, the domain expert is asked to help solve the dilemma by providing criteria to disambiguate the procedures.

5.3.4 CGEN

During integrity checking, CGEN exploits its knowledge roles to maintain the consistent behaviour in the expert system. Whenever structural knowledge introduces new parts, specifications for those parts must be provided. In addition, formulae for generating information needed later in the

design process must be provided. A part's design cannot begin until the part's specifications are filled, and a design is not complete until all 'abstract' parts are driven to 'physical' parts. Therefore, the following types of knowledge must be acquired at each session: selection, structural and calculation knowledge.

CGEN ensures that each newly introduced part has a corresponding set of specifications. If CGEN determines that certain specifications have no equations, it will inform the expert that additional knowledge is necessary. CGEN will then restart the knowledge-gathering process.

5.3.5 Knack

After creating a report, Knack predicts and exemplifies the performance that an expert can expect from the WRINGER being created. Knack instantiates the concepts of the generalized fragments with known concept representatives taken from the domain model, and displays several differently instantiated examples of each generalized report fragment. The expert edits any examples that make implausible statements about the domain. Knack identifies such events as incorrect knowledge-base use, the interprets the corrections as new knowledge for updating the generalization and for improving the domain model.

This process of abstraction and completion results in a knowledge base containing a large collection of generalized report fragments applicable more broadly than is the sample report. This collection is only part of a WRINGER's knowledge base. A WRINGER must be able to custom tailor the generalized report for a particular application. It uses strategies to determine values for the identified variables. The third kind of knowledge that Knack asks the expert to provide, therefore, comprises those strategies (strategy knowledge). Again, the expert types in samples, and Knack generalizes them using the domain model.

5.3.6 Opal

Consistency is maintained in Opal by a user interface that allows users to enter knowledge only if that knowledge is consistent with previously entered values. This check is performed by *ad hoc* code associated with each of Opal's form blanks. A second level of checking occurs using *ad hoc* code when the intermediate representation of the acquired knowledge is transformed into operational code.

5.4 Code generation

The final phase of the knowledge-acquisition process is generating code automatically from the intermediate format. This process frees the expert from having to know the implementation details of the expert system, which are often complex. When code generation is complete, the newly acquired knowledge is fully operational.

A common technique for generating code is based on instantiating code templates. This technique is used by all six tools reviewed in this paper. A code template is associated with a knowledge role, and it describes how to generate code for that role. The template describes a role as typical actions performed, necessary bookkeeping operations, and data-structure referencing. The template is then custom-tailored with the knowledge acquired from the user.

5.4.1 Mole

Mole represents its knowledge in the form of data. The code realizing the problem-solving method operates on those data. The problem-solving method is implemented in OPS5 (Forgy, 1981).

5.4.2 Burn

Burn represents its knowledge in the form of production rules written in OPS5. These are combined with code realizing the problem-solving method, also written in OPS5.

5.4.3 Salt

Salt operationalizes its domain-specific knowledge base into production rules written in OPS5. These rules are combined with code realizing the problem-solving method, also written in OPS5. The production rule implementing the procedure from Figure 4 follows in an English translation:

```
Rule 1
if   values are available for DOOR-OPENING,
     PLATFORM-WIDTH, and OPENING-WIDTH, and if
     the value of DOOR-OPENING is CENTER, and if
     there is no value for CAR-JAMB-RETURN,
then Calculate the result of the formula
     [PLATFORM-WIDTH–OPENING-WIDTH]/2.
     Assign the result of this calculation as the value of
          CAR-JAMB-RETURN.
```

5.4.4 CGEN

CGEN generates OPS83 code (Forgy, 1984). A specification rule corresponding to the input shown in Figure 5 follows:

```
RULE_G120293829
If   the current goal is specification, and if
     the current abstract part is CLOCK_GENERATOR_0, and if
     CLOCK_GENERATOR_0 has specification CLOCK_FREQUENCY
then mark goal as complete, and
     specification CLOCK_FREQUENCY = TX_BAUD * 16
```

5.4.5 Knack

Knack stores the knowledge that it acquires in a declarative form in its knowledge base. To create an expert system, Knack operationalizes the knowledge into OPS5 production rules using a simple parser written in Lisp. The following exemplifies rule generation for the report knowledge role.

Report rules represent the outline and the content of a WRINGER's report. Knack will insert a new chapter, section or subsection whenever it discovers a keyword such as 'chapter', 'section' or 'subsection' in a report fragment. An OPS5 rule representing a part of the skeletal report is created for each chapter, section and subsection heading. The skeletal report rule for a section heading follows:

```
if   the goal is to print the report,
then create Section 2 of Chapter 1 with the heading "Shielding
        Requirements for the Enclosures," and
     establish that fragment 9 can be selected, and
     establish that fragment 10 can be selected, and
     establish that fragment 11 can be selected, and
     . . .
```

The rule defines the section heading as it appears in the table of contents of the report that the WRINGER is producing. It also specifies the fragments that can be included in the section, and determines the order of those fragments. Other report rules represent the content of a WRINGER's report, and the pieces of information that need to be collected to instantiate the report.

5.4.6 Opal

The Opal plan components are represented with frames, and are implemented in Ozone, an object-oriented language created by the Opal developers. The sequencing information is implemented via augmented transition networks. Finally, refinement knowledge is realized by Mycin-style production rules and tables that are indexed by the contexts in which they apply, and by the parameters whose values they conclude.

6 Some data

This section presents data on the scope, performance and actual use of the six specialized knowledge-acquisition tools reviewed in this paper.

6.1 Mole

Mole has been used by knowledge engineers to build nine prototype systems. These experiments have resulted in two insights regarding Mole's scope:

- A task must have individual solutions. If a task has combinations of independent solutions, it would be impractical to preenumerate all possible combinations.
- A task must have enough structure to describe the solutions in terms of explanations.

6.2 Burn

Burn has been used successfully by knowledge engineers to build five prototype systems. One of the system was also built by a domain expert who had no programming experience. Building the full-scale prototypes took between 4 and 17 days.

6.3 Salt

Salt was used by a domain expert to generate an initial version of a system, called VT, that configures elevators. It is now being used by domain experts to maintain and extend VT.

Salt was also used to generate a prototype for a scheduling application. For this purpose, some of Salt's knowledge roles, as well as its problem-solving method, were redefined to accommodate this new application. The changes included extensions to the knowledge roles to account for time relations as part of design-extension knowledge, and functions as part of fix knowledge.

6.4 CGEN

CGEN is limited to hierarchical-configuration tasks with the following characteristics: the component functions can be organized in a hierarchy, the connections between components can be enumerated, and the constraints among parts in the design can be explicated at the start of the design process.

CGEN, and its expert system Micon, have seen considerable use (Birmingham, 1992). The number of users, outside of the development group, is over 50, with installation in a dozen sites. CGEN has acquired over 5000 rules. Knowledge bases created by CGEN have produced a variety of working computer systems; the most complex system is a medium-performance work-station with graphics and network capabilities.

As mentioned previously, CGEN has had moderate success in generating prototype systems in two other domains; software and mechanical design. The tasks to which CGEN was applied were relatively simple. For example, the mechanical design task did not require any reasoning about shape.

6.5 Knack

Extensive experiments analysed Knack's scope and performance. The details of the results are reported by Klinker et al. (1989). Knack's scope is reporting tasks with the following characteristics:

- The report to be produced can be standardized or nonstandardized (if it is nonstandardized, the expert must standardize it before interacting with the Knack tool), and must be structured and explicit.
- The task-specific information presented in a report can be standardized or nonstandardized (if it is nonstandardized, the expert must standardize it before interacting with the Knack tool), must be structured, and can be explicit or implicit.
- Most of the strategies used to acquire task-specific information must be interactive.

The analysis of Knack's performance showed that the tool could be used by knowledge engineers to create most of the knowledge bases for nine prototype systems in widely differing domains. In each case, however, the knowledge base had to be changed or extended to accommodate the specific requirements of a given domain.

Currently, Knack is being used to maintain the report generators that create reports on designs that may be suboptimal from a specified perspective.

6.6 Opal

Encoding one cancer-treatment protocol manually takes about six weeks. With Opal, a knowledge engineer can perform the same task in about one week. In 1986, Opal was used by knowledge engineers to encode 36 cancer-treatment protocols during a 12-month period.

7 Discussion

The facts presented in the previous section indicate that specialized knowledge-acquisition tools are powerful, if they meet the requirements of the domain. Their narrow scopes, however, results in two major drawbacks:

- It is difficult to determine whether a tool is appropriate for the given application.
- The tools break once they encounter situations that cannot be solved by their presupposed problem-solving methods and accompanying knowledge representations, i.e., the tools are brittle.

The following sections describe current research aimed at addressing these issues.

7.1 Determination of whether a knowledge-acquisition tools is appropriate

The problem of mapping application tasks onto the problem-solving models presupposed by the tools described in this paper is a serious one. It is difficult for people to understand a somewhat generic, abstract model of their task unless they have participated in the creation of that model. People performing a task are buried in the task details—details that make the task unique. These details are exactly those by which a person would recognize her task, and yet these are the same details that an abstract model has removed. We do not have a language for abstract models of application tasks. And we do not have a language to describe the tools that might be used to solve application tasks. For example, application tasks are 'Write a project proposal' or 'Present electromechanical systems such that they can be evaluated for conformance to safety regulations'. These application tasks are not labelled as reporting tasks. Thus, it is not immediately obvious what problem-solving method is appropriate for them.

A major focus in current knowledge-acquisition research is to characterize application tasks and to define mechanisms that can be used to solve them. The scopes of existing expert systems serve as data points. Although some progress has been made, the problem of how to characterize application tasks is still poorly understood. As of today, no taxonomy exists that characterizes real-life tasks, and that maps their characteristics onto problem-solving methods.

This comes as no surprise. We do not have enough data points yet. Compared to the number of real-life tasks, only a few specialized knowledge-acquisition tools have been developed. Only a few of these systems have been tested extensively for different tasks. We believe that using knowledge-acquisition tools for numerous tasks is the only well-grounded method to determine that tool's scope.

7.2 Reduction of brittleness in knowledge-acquisition tools

As described in this paper, current specialized knowledge-acquisition tools are targeted for the specific knowledge engineering function of extending an initial knowledge base. This goal leads to tools that break whenever it is necessary to change the structure of the knowledge base or the problem-solving behaviour. The tools can be used only to extend an existing knowledge base, not to design a new one, and not to design a problem-solving method. Since specialized knowledge-acquisition tools are difficult to build, this problem is a serious drawback. The tools are complex as, if not more complex, than, the expert systems that they construct. The skill needed to build these tools is scarce. There are a number of approaches working on providing assistance with these problems.

One approach is to start from programming languages and to create programming constructs that are more easily understood than are traditional language constructs, and therefore more easily configured. Rich et al. (1990) use constructs (cliches, in their terminology) that are functionally larger than those of traditional programming languages, but are still fairly small and have broad, general applicability. It takes many of these constructs to make an application. Thus, the task of determining how to combine the parts into a working program is easier than it is when traditional languages are used, but it still requires significant programming skill.

Another approach is taken by the SBF framework (Marques et al., 1992; Dallemagne et al., 1992). This project is defining building blocks for problem-solving methods, called *mechanisms*, that can be reused across problem-solving methods. The approach is to build mechanisms that are specific enough to allow the use of powerful knowledge-acquisition tools for instantiating them, yet are small enough to allow recombination, increasing their applicability. The SBF framework presupposes a library of such mechanisms and associated knowledge-acquisition tools, as well as a framework configuring and exploiting them. The framework consists of tools that interpret between the application developer (who is not necessarily a programmer) and the mechanisms, and tools that configure the mechanisms for the developer.

The DIDS project (Balkany et al., 1991; Runkel et al., 1992) takes an approach similar to that of SBF. DIDS is developing a graphics-based programming environment for developing configuration-design tools. The environment provides a tool to build a problem-solving method using mechanisms. A knowledge-acquisition tool is then automatically generated. The mechanisms in DIDS are the result of an extensive, and ongoing, analysis of the tasks that compose configuration design. The mechanisms have the features of reusability and combinability, which makes programming easier. Reusability reduces the amount of programming needed to build a system. Combinability ensures that mechanisms can be pieced together. The knowledge-acquisition tools produced by DIDS are specialized to each problem-solving method developed. These tools understand the relationships between knowledge roles, as well as the roles themselves. This understanding allows the tool to develop a good strategy for acquiring knowledge. The order in which knowledge is acquired can have a significant influence on the effectiveness of the knowledge-acquisition tool. For example, if a constraint network is used for a problem, the best ordering is to build the network completely before acquiring ways to backtrack through it. This order is better because, depending on the structure or the network, backtracking may not be required. Whether it will be required is impossible to determine, however, if backtracking knowledge is acquired on a constraint-by-constraint basis.

Protégé II (Tu, 1991) is a framework for knowledge engineers to generate knowledge-acquisition tools. It includes an architecture to assemble problem-solving methods from lower-

level building blocks. This work emphasizes a uniform view of problem-solving at different levels of granularity, an explicit data model to construct new datatypes from predefined datatypes, and the inclusion of domain-dependent control information.

The KADS and KADS II projects (Wielinga et al., 1992) are developing a library of building blocks (knowledge sources in their terminology) that appear to be of a scope similar to that of SBF's and DIDS's mechanisms. The researchers have built a formal framework of interrelated models to guide the process of building applications. The biggest difference among these systems is that the KADS framework is not intended to be operational. It defines a methodology aimed to help a programmer to design an application.

Acknowledgements

We thank Mark Musen and an anonymous reviewer for their exceptionally useful comments on earlier drafts of this paper.

References

Balkany, A, Birmingham, W and Tommelein, I, 1991. "A knowledge-level analysis of several design tools". In: *Proceedings AI and Design 1991*, Edinburgh, UK.

Birmingham, WP, Gupta, A and Siewiorek, D, 1992. *Automating the Design of Computer Systems: The Micon Project*, Jones and Barlett.

Birmingham, WP, Gupta, AP and Siewiorek, DP, 1989. "The micon system for computer design". In: *26th Design Automation Conference Proceedings*, IEEE Computer Society.

Birmingham, W, 1988. "Automated knowledge acquisition for a computer hardware synthesis system". In: *Proceedings 3rd Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada.

Boose, J, 1984. "Personal construct theory and the transfer of human expertise". In: *Proceedings 4th National Conference on Artificial Intelligence*, Austin, TX.

Boose, J, 1989. "A survey of knowledge acquisition techniques and tools" *Knowledge acquisition* **1** (1) 3–38.

Buchanan, B and Feigenbaum, E, 1978. "Dendral and metadendral: their applications dimensions" *Artificial Intelligence* **11**.

Chandrasekaran, B, 1990. "Design problem solving: A task analysis" *AI Magazine* **11** (4) 54–71.

Clancey, W, 1984. "Heuristic classification" *Artificial Intelligence* **27** 289–350.

Dallemagne, G, Klinker, G, Marques, D, McDermott, J and Tung, D, 1992. "Making application programming more worthwhile". In: F Schmalhofer, G Strube and Th Wetter, (eds.), *Knowledge Engineering and Cognition*, Springer-Verlag.

Davis, R, 1981. "Interactive transfer of expertise: acquisition of new inference rules" *Artificial Intelligence* **12** (2) 121–157.

Diederich, J, Ruhmann, I and May, M, 1987. "Kriton: a knowledge acquisition tool for export systems" *International Journal of Man-Machine Studies* **6** (1) 29–40.

Duda, R and Reboh, R, 1984. "AI and decision making: the prospector experience". In: W Reitman (ed.), *Artificial Intelligence Applications for Business*, Ablex.

Eshelman, L, Ehret, D, McDermott, J and Tan, M, 1987. "MOLE: a tenacious knowledge acquisition tool" *International Journal of Man-Machine Studies* **26** (1) 41–54.

Forgy, CL, 1984. "The OPS83 Report". Technical Report, CMU-CS-84-133, School of Computer Science, Carnegie Mellon University.

Forgy, CL, 1981. "OPS5 User's Manual". Technical Report, CMU-CS-81-135, School of Computer Science, Carnegie Mellon University.

Friedland, PE and Iwasaki, Y, 1989. "The concept and implementation of skeletal plans" *Journal of Automated Reasoning* **1** (2) 161–208.

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

Klinker, G, Boyd, C, Dong, D, Maiman, J, McDermott, J and Schnelbach, R, 1989. "Building expert systems with KNACK" *Knowledge Acquisition* **1** (3) 299–320.

Klinker, G, 1988. "KNACK: sample-driven knowledge acquisition for reporting systems". In: S Marcus, (ed.), *Automating Knowledge Acquisition for Expert Systems*, Kluwer.

Linster, M, 1989. "Towards a second generation knowledge-acquisition tool" *Knowledge Acquisition* **1** (2) 163–184.

Marcus, S, 1988. SALT: a knowledge-acquisition tool for purpose-and-revise systems". In: S Marcus, (ed.), *Automating Knowledge Acquisition for Expert Systems,* Kluwer.

Marques, D, Dallemagne, G, Klinker, G, McDermott, J and Tung, D, 1992. "Easy programming: Empowering people to build their own applications" *IEEE Expert* **7** (3) 16–29.

Marques, D, Klinker, G, Dallemagne, G, Gautier, P, McDermott, J and Tung, D, 1991. "More data on usable and reusable programming constructs". In: *Proceedings 6th Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada.

Marques, D, Latto, A and McDermott, J, 1988 "A comparison of case based reasoning with few large vs many small features". In: *Proceedings of the Workshop on Case Based Reasoning*, St Paul.

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

McDermott, J, 1982. "R1: a rule-based configurer of computer systems" *Artificial Intelligence* **19** (1).

Mitchell, T, Mahadevan, S and Steinberg, L, 1985. "LEAP: A learning apprentice for VLSI design". In: *Proceedings of IJCAI 85*, Morgan Kaufman.

Musen, M, 1989(a). "An editor for the conceptual models of interactive knowledge acquisition tools" *International Journal of Man-Machine Studies* **31** (6) 673–699.

Musen, M, 1989(b). *Automated Generation of Model-Based Knowledge-Acquisition Tools*, Pitman.

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

Offutt, D, 1988. "SIZZLE: A knowledge-acquisition tool specialized for the sizing task". In: S Marcus, (ed.), *Automating Knowledge Acquisition for Expert Systems*, Kluwer.

Rich, C and Waters, R, 1990. *The Programmer's Apprentice*, Addison-Wesley.

Runkel, JT, Birmingham, WP, Darr, TP, Maxim, BR and Tommelein, ID, 1992. "Domain independent design system: Environment for rapid development of configuration design systems". In: *2nd International Conference on AI in Design,* Butterworth-Heinemann Ltd., UK.

Shaw, MLG and Gaines, BR, 1987. "Techniques for knowledge acquisition and transfer" *International Journal of Man-Machine Studies* **27** (3) 251–280.

Shortliffe, EH, 1986. "Medical expert systems—Knowledge tools for physicians" *West Journal of Medicine* **145** 830–839.

Steels, L, 1990. "Components of expertise" *AI Magazine* **11** (2).

Tu, S, Shahar, Y, Dawes, J, Winkles, J, Puerta, A and Musen, M, 1991. "A problem-solving model for episodic skeletal-plan refinement". In: *Proceedings 6th Knowledge Acquisition of Knowledge-based Systems Workshop*, Banff, Canada.

Tu, S, Kahn, M, Musen, M, Ferguson, J, Shortliffe, E and Fagan, L, 1989. "Episodic skeletal-plan refinement based on temporal data" *Communications of the ACM* **32** (12) 1439–1455.

Westphal, C and McGraw, K, (eds.), 1989. Special Issue on Knowledge Acquisition, *ACM SIGART* **108**.

Wielinga, B, Schreiber, ATh and Breuker, JA, 1992. "KADS: a modeling approach to knowledge engineering" *Knowledge Acquisition* **4** 5–54.