Specification and Generation of Custom-Tailored Knowledge-Acquisition Tools*

Henrik Eriksson
Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University School of Medicine
Stanford, California 94305-5479
U.S.A.

Abstract

Domain-oriented knowledge-acquisition tools provide efficient support for the design of knowledge-based systems. However, the cost of developing such tools is high, especially when their restricted scope is taken into account. Developers can use metalevel tools to generate domain-oriented knowledge-acquisition tools that are custom tailored for a small group of experts, with considerably less effort than is required for manual tool development. An epistemic obstacle to creating such *metatools* is the specification model for target knowledge-acquisition tools. The metatool DOTS is based on an abstract-architecture approach to the specification and generation of knowledge-acquisition tools. DOTS is domain and method independent, because it is based on an architectural model of the target knowledge-acquisition tool.

1 Introduction

Many knowledge-acquisition tools are unsuitable for their tasks because they are adapted neither to the application domain, not to the requirements of individuals, such as developers and experts. Researchers in knowledge acquisition are experimenting with knowledge-acquisition tools custom tailored for specific domains [Gale, 1987; Musen *et al,* 1987]. Usually, such domain-oriented knowledge-acquisition tools are more useful than are general knowledge-acquisition tools, because custom-tailored tools can meet the requirements of the particular knowledge-acquisition situation.

Simultaneously, traditional knowledge engineering and expertise transfer is being replaced gradually by methodologies where developers assemble problem solvers for knowledge-based systems from reusable method components that accomplish subtasks [Chandrasekaran, 1986; Steels, 1990]. In McDermott's [1988] approach, developers use method-specific knowledge-acquisition tools to acquire the domain knowledge required by the methods. Method-oriented knowledge-acquisition tools, however, are not domain oriented per se; they must be adapted to specific domains and individuals.

Developers of knowledge-based systems wishing to use domain-oriented knowledge-acquisition tools face several barriers: It is difficult and laborious for developers of knowledge-based systems to adapt existing knowledge-acquisition tools, and to implement new domain-oriented knowledge-acquisition tools for new domains. Another barrier is that the investment of developing and maintaining domain-oriented tools cannot always be justified within the budget of a single application project. Many of these barriers can be eliminated by tools that enable developers to generate new domain-oriented knowledge-acquisition tools from high-level descriptions. Such *metatools* can simplify the task of developing domain-oriented knowledge-acquisition tools, and can reduce significantly the work required to implement these tools. Thus, metatools can make domain-oriented knowledge-acquisition tools feasible in situations where these tools could not be used previously. The knowledge-acquisition tools generated can then support the development of the target knowledge-based systems.

The design of metatools presents several epistemological and technical challenges. The way developers view knowledge acquisition affects the way that they design knowledge-acquisition tools. The developer's view of target knowledge-acquisition tools determines the appropriate specification strategy for target knowledge-acquisition tools in metatools. Automatic generation of such knowledge-acquisition tools requires a high-level description—or *metaview*—of the target tools [Eriksson and Musen, in press]. For example, PROTEGE [Musen, 1989] is a metatool that generates a domain-oriented knowledge-acquisition tool from an instantiation of a generic problem-solving method. The drawbacks of method-oriented metatools, such as PROTEGE, are that the problem-solving method supported cannot be replaced easily, and that such metatools do not handle combined methods well.

We have formulated a metaview, the *abstract-*

*The preparation of this manuscript has been supported in part by grants LM05157 and LM05305 from the National Library of Medicine, by gifts from Digital Equipment Corporation, and by scholarships from the Swedish Institute, from the Fulbright Commission, and from Stanford University. The development of DOTS has been supported by the Swedish National Board for Industrial and Technical Development (NUTEK).

architecture view, for specification of target knowledge-acquisition tools in metatools. The abstract-architecture view is based on a decomposition of the major functions in target knowledge-acquisition tools. In this approach, developers instantiate and combine subcomponents into specifications of target knowledge-acquisition tools, which are then used to instantiate target tools. DOTS[1] is a metatool that implements the abstract-architecture view [Eriksson, 1992]. DOTS allows developers to custom tailor knowledge-acquisition tools for new domains with minimal effort. DOTS is domain and method independent in the sense that it does not assume any particular domain, or problem-solving method, for the knowledge-acquisition tools it generates. DOTS assumes that the target knowledge-acquisition tools are based on graphical knowledge editing; the target knowledge-acquisition tools comprise several knowledge editors in which the experts enter their knowledge *actively* according to their conceptual model of the domain.

2 Design of Knowledge-Acquisition Tools

Knowledge-acquisition tools can be based on several models—for example, models of cognition, models of knowledge representations, and models of problem-solving methods. Musen [1989] describes three basic conceptual models for interactive knowledge-acquisition tools: *symbol-level, task-oriented,* and *method-oriented* conceptual models. The supportive power and the scope of a knowledge-acquisition tool follow from the model supported. Moreover, we can conceive many models for metatools. The essence of a metatool is the model for the target knowledge acquisition tools that the metatool supports. We use the term *metaview* for such specification models for knowledge-acquisition tools. We divide the development process for knowledge-acquisition tools into three major stages:

1. *Knowledge-acquisition analysis:* In this stage, the developer analyzes the domain and the task, and outlines the requirements for the knowledge-acquisition tool. The developer must acquire domain knowledge manually to determine the features required by the knowledge-acquisition tool.
2. *Tool specification:* In this stage, the developer models the knowledge-acquisition tool. When a metatool is used, the result of this stage is a specification that conforms to the metaview supported by the metatool; otherwise, the result is a regular software specification.
3. *Tool implementation:* In this stage, the developer—or the metatool—implements the knowledge-acquisition tool. Metatools transform the tool specification developed in step 2 to an operational program.

Developers can model domains independent of the problem-solving method through specification approaches, such as *knowledge-level analysis* [Newell, 1982] and *ontological analysis* [Alexander *et* al., 1987]. However, in the general case, the transformation from such models into a high-quality knowledge-acquisition tool is nontrivial. Target knowledge-acquisition tools must be *designed* by developers in cooperation with the tool users (e.g., domain experts). We seek appropriate knowledge-level descriptions for domain-oriented knowledge-acquisition tools (rather than descriptions of domains or descriptions of problem-solving methods), because knowledge-acquisition *tools* are different from target systems. Our research objective is to develop metaviews for domain-oriented knowledge-acquisition tools that are general; that is, they are not restricted, for instance, to a particular problem-solving method.

We have developed a rnetaview for target knowledge-acquisition tools that comprises generic building blocks for knowledge-acquisition tools—for instance, generic user-interface components for interactive knowledge editing, generic knowledge representation structures for internal use in the knowledge-acquisition tool, and generic knowledge-base generators that produce target knowledge bases. This *abstract-architecture* view incorporates architectural components of knowledge-acquisition tools at an abstract level. The abstract-architecture view provides the developer with a conceptual model of the target knowledge-acquisition tool that is based on the tool's architecture. Developers can specify a broad variety of knowledge-acquisition tools by specifying such building blocks and by defining the relationships among them. A metatool can automate stage 3 by transforming such specifications into an implementation of the knowledge-acquisition tool.

3 Generation of Knowledge-Acquisition Tools

DOTS is a metatool that supports the abstract-architecture view. DOTS allows its users to edit interactively an abstract-architecture specification of the target knowledge-acquisition tool. From this specification, DOTS produces an operational knowledge-acquisition tool that experts can use to develop knowledge bases. We shall discuss briefly the specification of knowledge-acquisition tools in DOTS. The details of the DOTS implementation in described in [Eriksson, 1992].

3.1 Specification Model

The abstract-architecture view comprises four component types, each of which constitutes a stage in the acquisition and generation of knowledge bases in knowledge-acquisition tools (see Figure 1). In this model, knowledge editors handle the user dialog; for instance, there are form-based and graph-oriented knowledge editors. Also, there are knowledge modules that represent the knowledge acquired by these knowledge editors. The relationships among the knowledge editors and the knowledge modules are defined by update rules. Finally, there is a description language of transformation rules that allows the developer to specify the knowledge-base generator of the target knowledge-acquisition tool. Specifically, the four component types in the abstract-architecture view

[1] DOTS is an acronym for Domain-Oriented Tool Support.

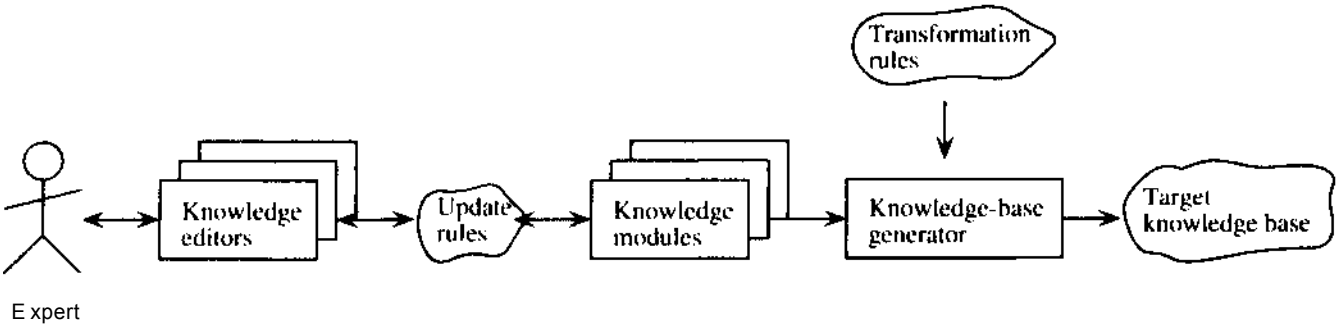

Figure 1: The model of the target knowledge-acquisition tools. The expert interacts with knowledge editors. Update rules map the contents of the knowledge editors to an internal representation—the knowledge modules. A knowledge-base generator driven by transformation rules transforms the knowledge modules to a target knowledge base. DOTS allows developers to edit the knowledge-acquisition-tool specification according to this abstract-architecture view.

are defined as follows:

1. *Knowledge editors:* The knowledge editors allow experts to *enter* and *edit* domain knowledge according to the experts' conceptual domain model. Examples of knowledge editors are domain-specific forms and graph editors. The knowledge editors are part of the user interface of the target knowledge- acquisition tool; they operate on the internal knowledge representation in the knowledge-acquisition tool, and provide views of the representation.

   In DOTS, developers can custom tailor the user interface of target knowledge-acquisition tools by specifying and refining knowledge editors, such as menu layouts, window-system behavior, and editor properties (see Figure 2). DOTS provides predefined types of knowledge editors for various types of knowledge, and also can allow the introduction of user-defined knowledge editors. Figure 3 shows a form-layout editor provided by DOTS. Developers use this tool to design layouts for form-based knowledge editors. The resulting form-based knowledge editor is shown in Figure 4. Note that this example is consistent with the fixes in the Sisyphus VT task.[2]

2. *Knowledge modules:* The knowledge modules provide *encapsulation* and *manipulation* of the internal knowledge representation of the target knowledge-acquisition tools. The information that the expert enters in the knowledge editors is stored in knowledge modules. Furthermore, the knowledge modules serve as an intermediate representation in the transformation of the knowledge acquired into the target knowledge base (see Figure 1).

3. *Update rules:* Update rules preserve consistency among the knowledge editors and the knowledge modules in the target knowledge-acquisition tools. The specification of the update rules defines the mapping between the knowledge presented and edited in the knowledge editors and the knowledge modules.

   Transformation rules: The knowledge-base generator of the target knowledge-acquisition tool generates the knowledge bases. In the DOTS approach, the knowledge-base generator is based on *transformation rules* that map knowledge represented by the knowledge modules into the appropriate knowledge-base structures (e.g., classes, instances, and rules). Generally, the rule preconditions of the transformation rules refer to the contents of knowledge modules, whereas the rule conclusions refer to structures in target knowledge bases. In other words, the transformation rules define the denota-

[2]The Sisyphus experiment is an attempt by the knowledge-acquisition community to provide a set of standard problems that researchers can use to compare their approaches to knowledge acquisition and problem solving. The Sisyphus VT task is based on the VT system for elevator configuration [Marcus *et* al., 1988],

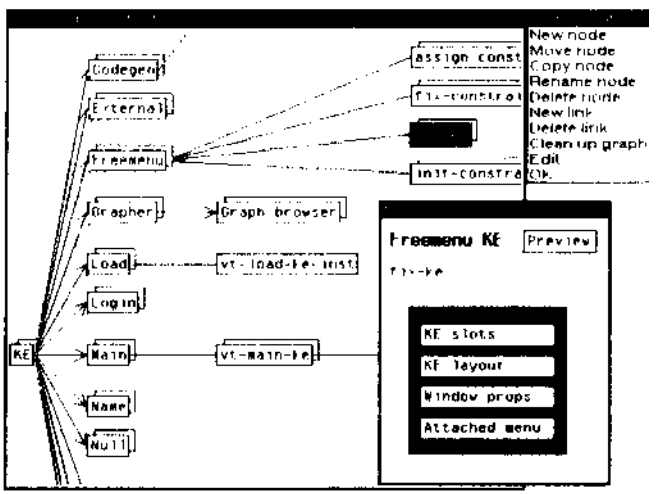

Figure 2: The specification of knowledge editors in DOTS. A sample user-defined knowledge editor (fix-ke) is highlighted in the knowledge-editor hierarchy. The developer uses an editor comprising subeditors for slots, menu layout, window properties, and attached menu to define the details of the knowledge editor (lower right).

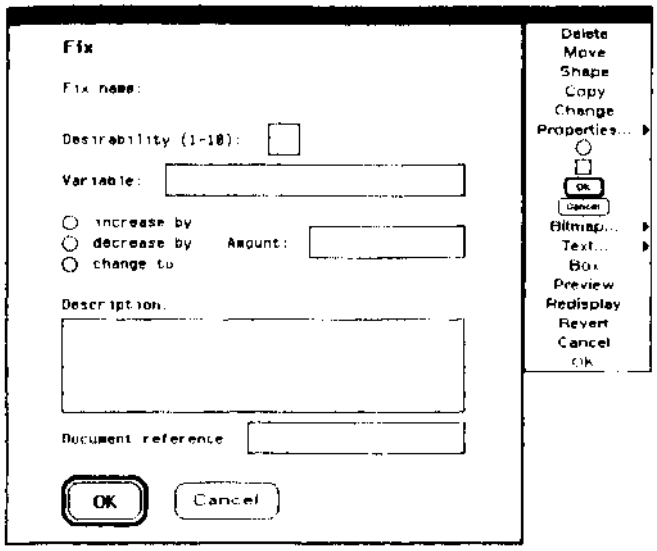

Figure 3: The editing of the layout for a form-based knowledge editor. DOTS provides this graphical tool, which developers can use to custom tailor forms for the domain—in this case, elevator configuration [Marcus *el at.,* 1988].

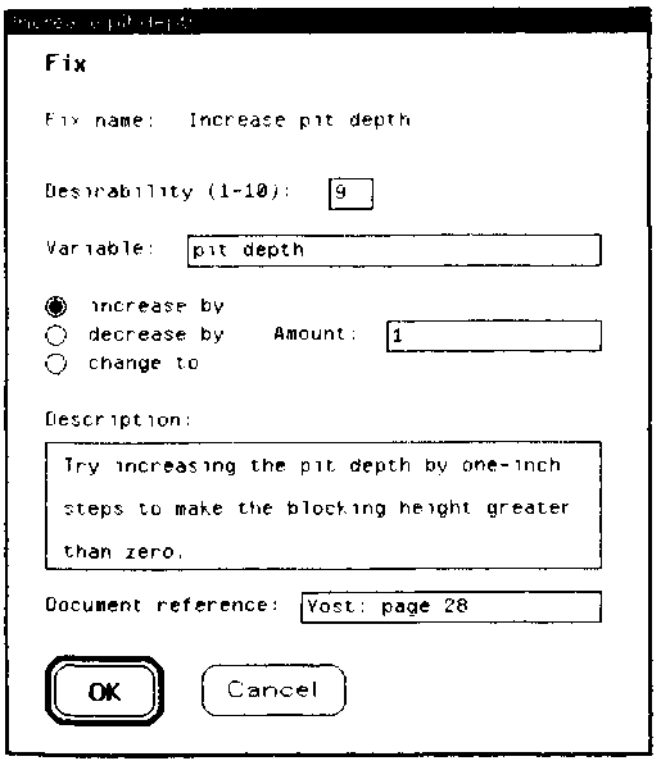

Figure 4: The resulting form-based knowledge editor in the target knowledge-acquisition tool. In this example, the experts use this form to specify fixes for constraint violations.

tional semantics for the knowledge that the target knowledge-acquisition tools acquire, because, currently, the knowledge-acquisition tools do not interpret the knowledge modules.

We have found the concepts of knowledge editors, knowledge modules, update rules, and transformation rules to be sufficient building blocks for interactive knowledge-acquisition tools based on graphical knowledge editing. Other types of knowledge-acquisition tools, such as interview-oriented tools that elicit knowledge through a textual dialog with the expert, might require other sets of components [Kawaguchi *et a/.*, 1991].

3.2 Generation of Target Tools from DOTS

The DOTS code generator takes as input the user's specification of the target knowledge-acquisition tool and produces code for the target knowledge-acquisition tool. The code generator is based on a set of transformation rules that maps abstract-architecture specifications (e.g., components descriptions) into constructs constituting the target knowledge-acquisition tool. The transformation rules are similar to those used by the target knowledge-acquisition tools for knowledge-base generation. The code generated is intended to run with a run-time library consisting of core functions required by the target knowledge-acquisition tool—for example, functions for window management, internal bookkeeping, persistent storage, and knowledge-base generation.

3.3 Generation of Knowledge Bases from Target Tools

The generic knowledge-base generator is an important component of the target knowledge-acquisition tools. We shall provide an example that illustrates how target knowledge bases are generated from the contents of knowledge modules. This example is loosely based on the Sisyphus VT task. Suppose that we want to generate rules from the information entered in the fix form (Figure 4). We assume that the developer has already defined the knowledge editor and the appropriate knowledge module for representing the information acquired. The task of the knowledge-base generator is to produce appropriate rules from such knowledge modules. For the form shown in Figure 4, we wish to generate an instance of the *fix* class for the knowledge base, in this example:

```
(definstance increase_pit_depthl of fix
  (desirability 9)
  (variable pit_depth)
  (action increase)
  (amount 1)
  (description "Try increasing..."))
```

In DOTS, transformation rules map the knowledge modules to the target knowledge base. Figure 5 shows a sample transformation rule (generate-1) that produces target instances from the knowledge modules. The clause **($km :whichis fix-km)** states that the transformation rule is applicable to all knowledge modules of type fix-km, and that the variable **$km** is bound to a knowledge-module instance during the generation process (i.e., $\forall x$ where $x \in$ **fix-km**). The precondition of the transformation rule tests whether the user has filled in

```
(define-transformation-rule generate-1
  ($km :whichis fix-km)
  IF (not-empty (get-value $km variable))
  THEN
   (build-instance
     :name (target-id $km)
     :class 'fix
     :initargs
     `((desirability ,(get-value $km desirability))
       (variable ,(get-value $km variable))
       (action ,(get-value $km action))
       (amount ,(get-value $km amount))
       (description ,(get-value $km description)))))
```

Figure 5: A transformation rule that maps the contents of the fix form to a an instance of the *fix* class in the target knowledge base.

the variable field in the fix form (Figure 4). If the precondition is satisfied, the conclusion of the transformation rule builds an instance of the class *fix* in the target knowledge base. If the expert has entered several fix-km into the knowledge-acquisition tool, several instances of the generate-1 transformation rule may generate multiple target instances. The purpose of the build-instance clause is to provide a convenient syntax for instance generation, and to allow the knowledge-acquisition tool to maintain the target knowledge base as the knowledge modules change, by adding and deleting rules from the knowledge base.

An alternative to using instances of the *fix* class to represent fixes in the knowledge base, is to use production rules to represent the fixes. In this case, the production rules would implement fix operations (e.g., increase a state variable). By using a build-rule clause, we can define readily a transformation rule that generates production rules from the contents of the fix-km knowledge module. In this approach, the user interface of the knowledge-acquisition tool and the fix form (Figure 4) would remain the same after we modified the transformation rule. By using this technique, we can defer design decisions about the knowledge representation in the knowledge base until we have acquired a significant body of knowledge.

4 Related Work

The principal work with which the abstract-architecture approach and its implementation in DOTS should be compared is PROTEGE [Musen, 1989]. PROTEGE implements a metaview that is based on a generic problem-solving method: *skeletal-plan refinement* [Tu *et* al., 1989]. However, this *method-oriented* view is restricted to a single problem-solving method. A subsequent project, PROTEGE-II, generalizes PROTEGE, and removes some of these restrictions [Puerta *et* al., 1992]. Spark, Burn, and FireFighter (SBF) [Marques *et ai,* 1992] form a set of tools designed to make programming easier by providing reusable programming constructs, or *mechanisms.* Spark helps the developer to identify and combine relevant mechanisms from a library. In the SBF approach, each mechanism in the library is supported by a corresponding knowledge-acquisition tool. Sis [Kawaguchi *et* al., 1991] is a metatool that supports a metaview similar to the abstract-architecture view. Sis, however, generates knowledge-acquisition tools that interview experts through a textual question-and-answer dialog.

PROTEGE, SBF, and other metatools that support method-oriented views require less modeling than DOTS does, because the former metatools draw their power from assumptions on knowledge acquisition for the problem-solving method supported, and because these metatools use a priori knowledge-acquisition tool designs. Compared to PROTEGE and SBF, DOTS trades supportive power for generality, and requires the developer to perform a knowledge-acquisition analysis that results in an overall tool design. DOTS differs from toolboxes that support programmers in the implementation of knowledge-acquisition tools in conventional programming languages [Gappa, 1991] by providing an abstract and coherent architectural view, which hides from the developer the details of the run-time library of the knowledge-acquisition tools.

5 Summary and Discussion

Although domain-oriented knowledge-acquisition tools provide effective support for the development of knowledge-based systems, the task of implementing such domain-oriented tools is laborious [Eriksson, 1991; Gale, 1987; Musen *et ai,* 1987]. Previous approaches to automatic generation of knowledge-acquisition tools, such as PROTEGE, have relied mainly on method-oriented views for the specification, which, unfortunately, make the metatools specific to the problem-solving methods of the target knowledge-based systems. The abstract-architecture view and DOTS contribute to knowledge acquisition by enabling developers to construct domain-oriented knowledge-acquisition tools.

The DOTS project demonstrated that the abstract-architecture view can be used for method-independent specification of knowledge-acquisition tools, and that the tools generated automatically from such specifications are comparable to hand-crafted tools. For instance, we have used DOTS to generate a basic knowledge-acquisition tool for the Sisyphus VT task. Given a domain ontology, the development time for this knowledge-acquisition tool was about 9 hours. Because the problem definition is based on the VT system and the associated domain, the knowledge-acquisition tool generated performs a knowledge-acquisition task similar to that of SALT [Marcus and McDermott, 1989], a knowledge-acquisition tool developed for the original VT system [Marcus *et ai,* 1988].

Another general conclusion is that the abstract-architecture view is *sufficient* for specification of graphical knowledge-acquisition tools that provide knowledge-editing support for domain experts. In our approach, a generic knowledge-base generator, which is parameterized by transformation rules, provides flexibility in terms of output knowledge bases from the knowledge-acquisition tools.

Although we can remove most domain and method restrictions by using the abstract-architecture view, this approach uncovers other obstacles. Although DOTS is method independent, it is restricted in terms of the types of knowledge-acquisition tools that it can generate. In DOTS, the design space of the target knowledge-acquisition tools is restricted to knowledge-acquisition tools with a common basic architecture. DOTS cannot generate easily other knowledge-acquisition tools, such as repertory-grid based tools and machine-learning tools. However, this restriction has not been a severe hindrance in practical development, because we are mainly interested in generating domain-oriented knowledge-acquisition tools that support the experts' domain models.

We have applied DOTS to several domain tasks. For instance, we have used DOTS to develop a knowledge-acquisition tool for a knowledge-based system that troubleshoots DNA sequencing machines[3] [Eriksson and Larses, 1992]. Also, we are currently formulating metaviews that bridge the PROTEGE and DOTS approaches by providing multiple perspectives. We are developing a metatool, DASH, that supports knowledge-acquisition analysis, and that helps developers to design target knowledge-acquisition tools from domain ontologies (i.e., class definitions).

Acknowledgments

I thank Sture Hagglund, Mark Musen, and Kristian Sandahl for valuable discussions about DOTS, and I am grateful to Lyn Dupre for editorial assistance. The examples based on the Sisyphus VT task benefited from discussions with John Gennari and Thomas Rothen flub.

References

[Alexander et ai, 1987] James II. Alexander, Michael J. Freiling, Sheryl J. Shulman, Steven Rehfuss, and Steven L. Messick. Ontological analysis: An ongoing experiment. International Journal of Man- Machine Studies, 26(4):473 485, 1987.

[Chandrasekaran, 1986] B. Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. IEEE Expert, I(3):23-30, 1986.

[Eriksson and Larses, 1992] Henrik Eriksson and Per Larses. ALFA: A knowledge acquisition tool for troubleshooting of laboratory equipment. Journal of Chemical Information and Computer Sciences, 32(2):139-144, 1992.

[Eriksson and Musen, in press] Henrik Eriksson and Mark A. Musen. Conceptual models for automatic generation of knowledge-acquisition tools. Knowledge Engineering Review, in press.

[Eriksson, 1991] Henrik Eriksson. Specialized knowledge acquisition tool support compared to manual development: A case study. In Proceedings of the Seventh IEEE Conference on Artificial Intelligence Applications, Miami, FL, February 1991.

[Eriksson, 1992] Henrik Eriksson. Metatool support for custom-tailored domain-oriented knowledge acquisition. Knowledge Acquisition, 4(4):445-476, 1992.

[Gale, 1987] William A. Gale. Knowledge-based knowledge acquisition for a statistical consulting system. International Journal of Man-Machine Studies, 26(I):55-64, 1987.

[Gappa, 1991] Ute Gappa. A tool-box for generating graphical knowledge acquisition environments. In Proceedings of the World Congress on Expert Systems, pages 797-810, Orlando, FL, December 1991.

[Kawaguchi et ai, 199l] Atsuo Kawaguchi, Hiroshi Motoda, and Riichiro Mizoguchi. Interview-based knowledge acquisition using dynamic analysis. IEEE Expert, 6(5):47-60, October 1991.

[Marcus and McDermott, 1989] Sandra Marcus and John McDermott. SALT: A knowledge acquisition language for propose-and-revise systems. Artificial Intelligence, 39(I):I-37, 1989.

[Marcus et ai, 1988] Sandra Marcus, Jeffrey Stout, and John McDermott. VT: An expert elevator designer that uses knowledge-based backtracking. AI Magazine, 9(1):95- 112, Spring 1988.

[Marques et al, 1992] David Marques, GeofTroy Dallemange, Georg Klinker, John McDermott, and David Tung. Easy programming: Empowering people to build their own applications. IEEE Expert, 7(3): 16-29,June 1992.

[McDermott, 1988] John McDermott. Preliminary steps toward a taxonomy of problem-solving methods. In Sandra Marcus, editor, Automating Knowledge Acquisition for Expert Systems, chapter 8, pages 225-256. Kluwer Academic Publishers, Boston, MA, 1988.

[Musen et ai, 1987] Mark A. Musen, Lawrence M. Fagan, David M. Combs, and Edward H. Shortliffe. Use of a domain model to drive an interactive knowledge-editing tool. International Journal of Man-Machine Studies, 26(1):105-121, 1987.

[Musen, 1989] Mark A. Musen. Automated Generation of Model-Based Knowledge-Acquisition Tools. Morgan-Kaufmann, San Mateo, CA, 1989.

[Newell, 1982] Allen Newell. The knowledge level. Artificial Intelligence, 18(1):87 127, 1982.

[Puerta et ai, 1992] Angel H. Puerta, John W. Egar, Samson W. Tu, and Mark A. Musen. A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools. Knowledge Acquisition, 4(2):171-196, 1992.

[Steels, 1990] Luc Steels. Components of expertise. AI Magazine, II(2):28-49, 1990.

[Tu et ai, 1989] Samson W. Tu, Michael G. Kahn, Mark A. Musen, Jay C. Ferguson, Edward H. Shortliffe, and Lawrence M. Fagan. Episodic skeletal-plan refinement based on temporal data. Communications of the ACM, 32(12):1439-1455, 1989.

[3] DNA sequencing machines are pieces of laboratory equipment that analyze DNA molecules to determine the latter's base-pair sequences.