Int. J. Human–Computer Studies (1998) **49,** 315–338
Article No. hc980209

A competence theory approach to problem solving method construction†

B. J. Wielinga
University of Amsterdam, Social Science Informatics, Roetersstraat 15, NL-1018 WB Amsterdam, The Netherlands. email: {wielinga}@swi.psy.uva.nl

J. M. Akkermans
Free University Amsterdam, Computer Science Department, De Boelelaan 10812, 1081 HV Amsterdam, The Netherlands. email: HansAkkermans@cs.ucc.nl

A. Th. Schreiber
University of Amsterdam, Social Science Informatics, Roetersstraat 15, NL-1018 WB Amsterdam, The Netherlands. email: {schreiber}@swi.psy.uva.nl

This paper presents a theory of the construction process of problem-solving methods (PSMs) on the basis of the competence theory approach. This approach describes the refinement process of an initial, abstract formalization of the required competence of a PSM, towards an operational version of the PSM. Three major steps in this process are identified: specification of the required competence theory, refinement of the theory into a form that fits a PSM paradigm and the operationalization of the theory into a form that is close to an executable specification. As an example, the ontological commitments and assumptions underlying some problem-solving methods for classification problems are investigated and their operational forms are presented.

1. Introduction

Over the years the knowledge-based system (KBS) community has developed a large number of knowledge-based problem solvers, employing a wide variety of knowledge structures and reasoning methods. In spite of the success in developing useful applications, KBS development has been a tedious and expensive process. Not only the acquisition of the knowledge about the application domain is a painstaking job, the design, implementation and testing of the reasoning methods employed in KBSs is difficult and time-consuming. During the last decade, research efforts have aimed at the development of models of generic reasoning processes—problem solving methods

†The work reported here was partially carried out in the course of the KACTUS and the GRASP projects, with partial funding of the Commission of the European Union. The partners in the KACTUS project were: Cap Gemini Innovation (F), LABEIN (Sp), Lloyd's Register (UK), Statoil (N), CAP Programmator (S), University of Amsterdam (NL), University of Karlsruhe (D), IBERDROLA (Sp), DELOS (I), FINCANTIERI (I) and SINTEF (N). The partners in the GRASP project are: Metropolitan Police Service (UK), ACSE (BE), AI ENGINEERING (NL), University of Amsterdam (NL), ASTRA (UK), LISITT (SP), Ministerio de Cultura (SP), and ARTTIC (D). This article expresses the opinions of the authors and not necessarily those of the consortia. JMA carried out the reported work at the University of Twente.

(PSMs)—which are not application-specific and can be reused and can be made available in libraries. This had led to a number of descriptions of PSMs which are generic, but which are poorly understood in terms of their applicability, scope and performance properties. This poor understanding had led to what we would like to call the PSM enterprise: an effort to achieve a better understanding of the competence, scope and applicability of a PSM and to design systematic methods for developing new methods and for adapting existing ones.

A problem-solving method (PSM) is a formal or operational account of how a class of knowledge-intensive problems can be solved in a computationally tractable way. A problem is knowledge intensive if its solution requires a reasoning process involving a number of non-trivial and often non-deterministic inference steps that make use of knowledge of the application domain. A central assumption underlying the PSM enterprise is that PSMs can be formulated as a *generic* description of a reasoning process such that it is largely independent of the specific application domain. Given such a generic PSM, two important questions have to be answered if a PSM is considered as a candidate for solving a particular problem: "Will the PSM solve the problem in the required manner?" and "Will the PSM solve the problem in an efficient manner?". The answer to the first question will (loosely) correspond to the notion of *competence* of a PSM that will be developed in this paper and the second question addresses issues of *performance*. In this paper, we will explore some formal aspects of PSMs and develop a method of stepwise construction of a PSM that allows a relation between competence and performance to be established.

The process of selection or construction of a PSM usually starts out with an informal statement of a goal to be reached from some initial situation in which some facts are known or can be obtained through interaction with the environment. The goal specification should describe the information structures that constitute a solution and some criteria that solutions should satisfy. Often, but not always, the informal problem statement will also contain statements about the nature of the knowledge to be used in the problem-solving process. In addition, the problem statement may indicate requirements of a pragmatic nature such as the amount of questions asked to the user, reliability of the input, the solution time or the number of solutions allowed or required.

The problem of finding a suitable PSM from an informal problem statement can be viewed as a process of successive refinement. Starting point is a formalization of the problem statement in an abstract form: the initial *competence theory* which defines the basic ontological commitments and the solution criteria. Through a process of conceptual refinement, the initial competence theory is further elaborated through the introduction of new vocabulary and assumptions. A second refinement process transforms the refined competence theory into an operational form: a specification of a set of basic reasoning steps together with a control structure. Although the approach we present in this paper has a strong top-down flavor, the method can also be applied in an incremental way. The main advantage of making the competence theories at different levels of abstraction explicit is to analyse the ontological commitments and background assumptions that a PSM makes with regard to the domain theory and the required reasoning.

In this paper, we apply the competence theory approach to a number of variants of classification problems to illustrate the concepts and process of PSM construction.

Section 2 gives a brief overview of current approaches to the study of PSMs and their construction. In Section 3, we give a general survey of the competence theory analysis of PSMs. The approach will be illustrated using a simple method for classification problems. This initial description of the approach will gloss over many issues that arise when more complex problem contexts are tackled and will focus on the nature of the various theories rather than on the process of PSM construction. In Section 4 we introduce an application domain for classification and a more demanding problem class. We discuss each of the steps in the construction process in more detail in the context of the more complex application domain and task.

2. Approaches to the study of PSMs

There are currently different lines of AI research in which the theory underlying problem-solving in knowledge-based systems is studied. A first area is the research on *reusable problem-solving methods* (McDermott, 1988; Musen, 1989; Steels, 1990; Klinker *et al*., 1991; Chandrasekaran, 1988; Clancey, 1992; Chandrasekaran *et al*., 1992; Wielinga *et al*., 1992; Schreiber *et al*., 1994). This research originates from the need for explicating the strategy behind the reasoning process in large-scale knowledge-based systems. A central topic in this research concerns the nature of decompositions of the inference process and the knowledge-level (Newell, 1982) characterization of the types of support knowledge used in this process. The aim here is to identify and index generic problem-solving methods (PSMs) that can be used to build an application problem solver in a domain, by supporting configuration-based or compositional KBS modelling and design. Ultimately, the idea is that one can provide well-indexed libraries of PSMs that have proven useful and reusable so that application system building reduces to combining and configuring PSMs out of such libraries in a compositional way (Breuker & Van de Velde, 1994).

In other areas, one focuses on a specific task or domain and provides definitions of problem-solving ingredients like problem, solution and explanation, while their interdependencies are defined through an abstract specification of the derivability relation between the solution and the problem and the knowledge base. This work may be viewed as yielding clear, but top-level, goal specifications for a task. A good example of this line of research is the work on logic-based diagnosis (Reiter, 1987; Poole, 1988; Console & Torasso, 1990). This type of work is important since it provides specific PSMs for certain tasks.

While the focus of the above-mentioned work is mostly ‘product-oriented’—it provides problem solving methods and describes their structure in detail, but leaves the underlying construction process largely implicit—, more recent work emphasizes the process of PSM construction. Several approaches have been taken: decomposition operators for PSM construction (Benjamins, 1993), rewrite grammars for PSMs (van Heijst *et al*., 1992), the formalization of generic inference components for PSMs (Aben, 1995), parametric design of PSMs (ten Teije *et al*., 1996; ten Teije, 1997), assumption identification of PSMs (Fensel, 1995*a*) and executable formalizations of PSMs (Fensel, 1995*b*; Fensel & Groendoom, 1996; Fensel *et al*., 1996). These various approaches highlight two important ideas: the use of formal methods for the precise specification of PSMs and the use of transformational techniques for the construction of an operational

PSM for a top-level goal. The work described in this paper is also 'process-oriented' and builds upon these two ideas, but is focused on the integration of the formalization and transformational approaches. The aim is to identify the generic elements in the process of constructing problem-solving methods using formal methods. A framework is sketched that shows how to construct in a generic and principled way an operational configuration of a PSM starting from a top-level goal specification for a problem.

3. PSM analysis: the competence theory approach

As discussed in our previous work (Akkermans *et al.*, 1998*b*, 1994; Wielinga *et al.*, 1995), the general starting point of the competence-theory approach to PSMs is the view taken by Newell (1982) that knowledge is a competence-like notion and that rationality means to apply this competence in order to achieve a given goal, that is, to solve a problem. A problem-solving method can be ascribed a certain competence with respect to the class of problems it has been designed to solve. Therefore, as a key element in our approach we will employ the notion of a *competence theory* of a PSM (Van de Velde, 1988): a generic description of the ability that a PSM requires in order to solve a certain class of problems. The rational justification of a PSM lies in demonstrating its competence with respect to the given problem. This is done by outlining how the PSM construction process is guided all along by the conceptual refinement and operationalization of an initial, required, competence theory that is immediately linked to a top-level goal specification.

3.1. OVERVIEW OF THE STEPS IN PSM CONSTRUCTION

An overview of the "competence-theory" approach is given in Figure 1. The process contains the following three main steps.

1. *Specification* of the problem space and of the requirements for the solution. This yields a required competence theory $\mathbf{T}_0$ for the PSM.
2. *Conceptual refinement* of this competence theory introducing the intermediate task and domain vocabulary based on assumptions regarding the available domain theory and the ways in which the task goal is achieved by the method. This leads to a refined competence theory $\mathbf{T}_1$ of a PSM.
3. *Operationalization* of this refined competence theory to inference structures and associated control regimes that are sufficiently detailed and practical to act as a basis for KBS design and implementation. This entails further assumptions of an operational nature and yields an operational specification $\mathbf{T}_2$ of a PSM.

This process can be seen as a special form of step-wise refinement, in which in each step a specific type of refinement is introduced, e.g. the structure of the domain theory. The three theories are not necessarily logically equivalent. In particular, $\mathbf{T}_2$ is typically weaker than the $\mathbf{T}_1$, because the operational assumptions usually limit in some way the competence of the method. The same holds, although to a somewhat lesser extent for the relation between $\mathbf{T}_0$ and $\mathbf{T}_1$. The specific conceptualizations introduced in the domain theory limit the competence. For example, in a configuration problem, the conceptualization of design preferences may actually limit the solution space. Thus, when going from

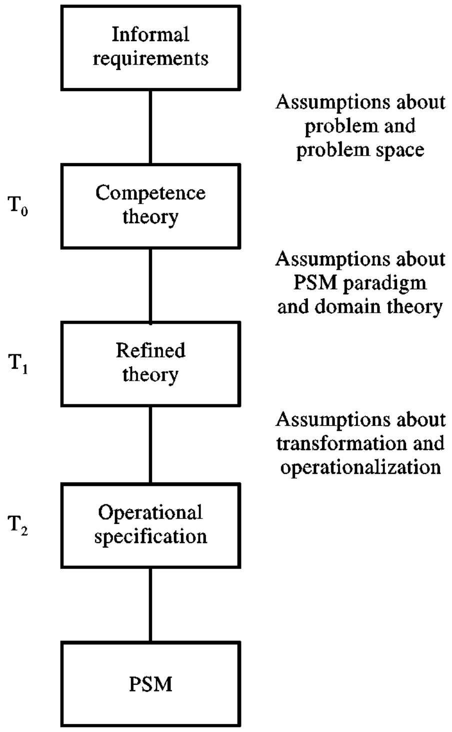
Informal requirements
Assumptions about problem and problem space
T_0
Competence theory
Assumptions about PSM paradigm and domain theory
T_1
Refined theory
Assumptions about transformation and operationalization
T_2
Operational specification
PSM

FIGURE 1. Competence theory approach to PSM analysis and construction.

one theory to a next we are typically working with a weakened version of the previous theory.

We discuss each of these steps in some more detail in the rest of this section.

3.2. FORMALIZATION OF THE COMPETENCE THEORY T_0.

The first step in the construction of a PSM is the specification of the task and the problem space.

Van de Velde (1988) defines the notion of *problem space*, Ω, as a triple

$$\Omega = \langle P, S, solution \rangle \tag{1}$$

where P is the set of problems in Ω, S is the set of solutions in Ω and *solution* is a relation between P and S, that is, $solution \subset P \times S$. A solution for a specific problem $p \in P$ in a problem space Ω then is any $s \in S$ such that $solution(p, s)$ holds. In the simplest case, the solution relation can be fully specified directly; then, the problem-solving method boils down to a direct match based on table look-up. However, since we are dealing with knowledge-intensive problems that require several non-trivial reasoning steps this will not be the case and we have to assume knowledge of the domain of application to be present. The competence of a PSM is therefore relative to some theory of the domain. Therefore, we extend the definition of the problem space Ω with an additional element:

the domain theory DT.

$$\Omega = \langle P, S, DT, solution \rangle \tag{2}$$

Based upon this abstract definition, the process of *problem specification* means to analyse in more detail what the ingredients and ontological commitments of the problem space Ω are, leading to the definition of the first version of the competence theory: $\mathbf{T}_0$.

The informal problem statement usually is not precise enough to specify what competence is required from a PSM. There are several issues to be determined before a begin can be made with the specification of an adequate PSM. The specification process needs to address the following issues.

- What is the information structure of a problem?
- What is the information structure of a solution?
- What is the nature of the domain knowledge?
- Assumptions: what are the epistemological and ontological commitments that apply to the task?

This process will be illustrated for a simple type of classification problem. This problem class is characterized by a set of class definitions in terms of sets of attribute–value pairs and observations which are also a set of attribute–value pairs. This type of problem often is the target problem solver in machine learning applications and is discussed in many AI textbooks such as in Stefik (1995). Even in this simple case, we have various choices of how to formalize the notion of classification and we have numerous options for operationalization of the competence theory as we will see below.

Table 1 defines the space of problems as the power set of observations, where an observation is a single valued attribute-value pair (a feature). We use the symbol $\exists!$ to indicate the uniqueness quantifer meaning that there exists a unique object satisfying the quantified formula. The notation $\{\ \cdots\ x_i\ \cdots\ \}$ is used represent sets. For a particular problem not all attributes need to have been observed. This definition already contains a number of commitments. Observations can only assign a single value to an attribute, values are confined to a predefined set and cannot be "unknown". Also, the set of observations is assumed to be known before the problem-solving process starts.

TABLE 1
Problem definition for simple classification

Attribute set:	$\{\ \cdots a_i \cdots\ \} = A$	
Value set:	$\forall a_i \colon \exists\{\ \cdots v_{ij} \cdots\ \} = V_i$	
Feature set:	$\{\ \cdots f_{ij} \cdots\ \} = F$ *where* $f_{ij} = \langle a_i, v_{ij} \rangle$	(3)
Observation set:	$\{\ \cdots o_i \cdots\ \} = O$	
Single-value observations:	$o_i \rightarrow \exists! k$ such that $o_i = f_{ik}$	
Problem set:	$\{\ \cdots p_i \cdots\ \} = P = \mathscr{P}(O)$	

TABLE 2
Solution and domain-theory definition for simple classification

Set of classes:	$\{\cdots c_i \cdots\} = C$	
Set of class definitions:	$\{\cdots d_{ij} \cdots\} = DT$ where $d_{ij} \equiv c_i \rightarrow \bigvee_k f_{jk},\ c_i \in C, f_{jk} \in F$	(4)
Solution set:	$\{\cdots s_i \cdots\} = S,\ s_i \in C$	

TABLE 3
Solution criteria for simple classification

Solution predicate:	$\forall p \in P,\ s \in S\ solution(p, s) \leftrightarrow \exists s \forall o_i \in p\ \ s \cup DT \models o_i$	(5)

Table 2 defines the solution space and the domain theory. Classes are represented as simple propositions. A solution simply corresponds to a class. The domain theory consists of class definitions in terms of a disjunction of features implied by the class. For each class a set of such definitions can exist, corresponding to the knowledge that links the class to the various attribute–value combinations.

An alternative choice for the representation of the class definitions—one that is often used in KBSs—could have been a set of separate implications, such that more than one feature for a specific attribute could be implied by a class definition. Although this type of knowledge could be valid for a class, it cannot be valid for a particular object that is being classified because the description of a single object in terms of observations is assumed to have a unique value for an attribute. Since a solution is a piece of knowledge about an individual object and not about a class, the latter representation would not constitute a consistent competence theory.

Apart from the class names, the solution and domain ontologies in our simple example do not introduce any new vocabulary beyond that of the problem vocabulary. The domain theory in the example makes rather strong commitments about the structure of the knowledge. In general, the competence theory $\mathbf{T}_0$ will make much weaker assumptions about the knowledge in the domain theory. The problem, solution and domain-theory definitions are obviously abstractions of the application domain. No application-specific terms are used, the mapping between the abstract ontology in $\mathbf{T}_0$ and the domain knowledge is achieved partially in $\mathbf{T}_1$ and finally in $\mathbf{T}_2$.

Table 3 defines the solution predicate as the requirement to cover all observations. This is a strong solution criterion, referred to as *complete converge* (Stefik, 1995). The underlying assumption is that all observations are relevant with respect to the classification knowledge base. Other variants of the solution criterion for classification will be discussed in Section 4.

Table 1–3 together specify the initial competence theory $\mathbf{T}_0$. Already in this stage, several observations can be made about $\mathbf{T}_0$.

- There is no guarantee that for a particular problem a solution exists that satisfies the solution criteria. If such a guarantee would be required, additional properties of the domain theory should be enforced. For all possible sets of observations there should be at least one class definition that covers them.
- The competence theory defined above does not enforce a solution to be unique: more than one class can satisfy a given problem. Of course, a *single solution* criterion could be added to the solution predicate. This will be shown in Section 5.
- Class definitions are not required to be unique: there can be identical class definitions or classes can overlap.
- For classes to be viable as solutions, they should include all features that a problem contains as observations. This implies that a DT structured as a decision tree would only be able to produce a solution when the observed features correspond exactly to the set of decisions in the tree that led to the solution.
- If the knowledge about the domain is incomplete, i.e. certain features are unknown for certain classes, these could be included in the class definitions as a disjunction of all possible features or they could be omitted from DT. In the first case observations in the area of incomplete knowledge are accepted; in the latter case only those problems will be solved in which the unknown attribute in the class has no corresponding observation in the problem.

These observations indicate that even in the every simple example, the construction of the initial competence theory $\mathbf{T}_0$ already involves many choices and assumptions concerning the PSM and the DT required.

3.3. CONCEPTUAL REFINEMENT: $\mathbf{T}_1$

The competence theory $\mathbf{T}_0$ is still a rather abstract specification of what the PSM should achieve and is by no means suitable for operationalization. For example, the entailment relation in Equation 5 needs to be refined such that individual reasoning steps can be identified.

The *complete coverage* solution criterion (Stefik, 1995) imposes strong requirements on the domain theory: for a class to be admissible as a solution, its definition should include implications of all features that occur in the problem. This is a consequence of the ontological commitment that was made with respect to the structure of the domain theory. We can formalize this as follows. Since observations can only assign a single value to an attribute, we can infer from an observation that all other features are false (Equation 6):

$$o_i = f_{ik} \rightarrow \bigwedge_{j \neq k} \neg f_{ij}. \tag{6}$$

This means that if a feature f_{ik} is observed and it occurs in the disjunction side of a class definition, it is implied by that class definition (Equation 7). Using a shorthand notation F_i for the set of terms occurring in a disjunction of a class definition for attribute i, we can derive that an observation in the disjunction of a class definition implies that

TABLE 4
Introduction of two (meta-)predicates in $\mathbf{T}_1$

$$class(c_i) \leftarrow \quad c_i \in C \tag{9}$$

$$\begin{aligned} has\_features(c_i, a_j, V_j) \leftarrow \quad & class(c_i) \wedge \\ & V_j = \left\{ v_{jk} \mid isdefined\left(c_i \rightarrow \bigvee_k f_{jk} \right) \wedge f_{jk} = \langle a_j, v_{jk} \rangle \right\} \end{aligned} \tag{10}$$

observation:

$$\left(s \rightarrow \bigvee F_i \wedge f_{ik} \in F_i \wedge o_i = f_{ik} \right) \rightarrow (s \rightarrow o_i). \tag{7}$$

In order to be able to test whether a certain formula is part of the domain theory we introduce a meta-predicate *isdefined* that is true when the formula that is its argument is in the domain theory. We can now derive

$$solution(p, c_i) \leftrightarrow \forall o_j = f_{jk} \in p \wedge isdefined \left(c_i \rightarrow \bigvee F_j \right) \wedge f_{jk} \in F_j. \tag{8}$$

Strictly speaking, Equation 8 is not a first-order statement since it contains a meta-level predicate. Although this problem can be solved by simple techniques, it is an indication of a more general phenomenon that will be discussed in depth later: the need to make statements about the structure and content of the domain theory.

Given the rewritten solution predicate (Equation 8), we can now start to design the structure of the knowledge base and define a number of auxiliary predicates that bring us closer to the operational from of a PSM. We first introduce two predicates: *class* and *has features* that make some of the logical machinery (e.g. membership of domain terms) easier to handle (Table 4).

The reason for the introduction of the predicate *class* is to rewrite the set membership of the solution space in clausal form, such that we will be able to link it to the knowledge base where we assume that the names of classes are stored.

The predicate *has features* is used to hide the meta-level character of Equation 8 from the solution predicate definition. In fact, we will assume later that *has features* is explicity defined in the domain knowledge base. It is important to note that where the domain ontology in $\mathbf{T}_0$ represents the class definitions as logical implications, $\mathbf{T}_1$ assumes a much more specific representation of the domain knowledge. However, the required competence of the initial theory is still logically fulfilled by $\mathbf{T}_1$. This illustrates a very important aspect of the competence theory approach: the transformation from a general logical theory—which is computationally intractable—to a much more specific form of the theory, reduces the computational complexity through the restriction of the use of the domain knowledge expressions. This provides the essential power of the knowledge-based approach, without losing the logical rigor.

In addition to the two predicates which link the knowledge base to the competence theory, we define a predicate *explains* that is true when a class entails a given feature.

TABLE 5
Refinement of the solution criteria in $\mathbf{T}_1$

$explains(s, f_{ik}) \leftarrow$	$\exists f_{ik} = \langle a_i, v_{ik} \rangle \wedge$	
	has features$(s, a_i, V_i) \wedge v_{ik} \in V_i$	(11)
$solution(O, s) \leftarrow$	$class(s) \wedge$	
	$\forall o_i = f_{ik} \in O \; explains(s, f_{ik})$	(12)

Given this new predicate and assuming that a problem is essentially a set of observations, we can write Equation 8 as Equation 12 in Table 5.

Table 4 and 5 represent the *refined competence theory* $\mathbf{T}_1$. The form of Equation 12 as a conjunction of a number (i.c. two) predicates is typical for theories of type $\mathbf{T}_1$. The original solution criteria are transformed and decomposed in a number of terms which can be viewed as inference steps. The structure of the conjunction of terms that make up the solution predicate usually reflects a certain approach to the decomposition of the problem solving process: the *PSM paradigm*. The conjunction in Equation 12 reflects the Generate-and-Test PSM paradigm.

3.4. OPERATIONALIZATION $\mathbf{T}_2$

The third step in the PSM construction process is concerned with the transformation of $\mathbf{T}_1$ into an operational form. With "operational" we mean that we specify the procedure (or strategy) for solving a problem in practice. This requires that we define reasoning steps and the way in which these steps need to be ordered to solve the problem.

The objective of the operationalization is to define a feasible strategy that satisfies the application requirements. Typically, there are multiple ways in which we can operationalize a competence theory. We show a few simple examples of the operationalization of the refined competence theory in the previous section. Further on, we discuss the reasons that can underly the choices that need to be made in this process.

Strategy 1: simple generate and test. A first simple strategy that comes to mind when looking at Table 5 is the following.

1. Take a class (at random).
2. Check whether all the observations are explained by the feature definitions of this class.
3. If the check succeeds, a solution has been found, else return to step 1.

We can specify this strategy through a CommonKADS inference structure and control structure. Two basic reasoning steps ("inferences") are required that use the domain predicates in the refined competence theory:

Generate a class as candidate solution using the *class* predicate.

Test a candidate solution using the *explains* definition and *has features* predicates.

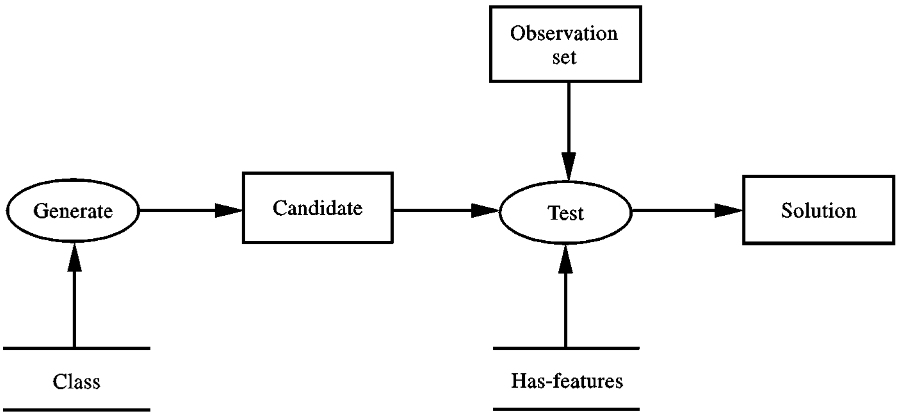

FIGURE 2. Inference structure for the simple generate-test strategy. Legend: ovals are reasoning steps ("inferences"); boxes indicate dynamic inputs of reasoning steps ("dynamic roles"); the open boxes denote the underlying knowledge used ("static roles").

The two inferences are shown in Figure 2. The control flow of the strategy can be expressed in pseudo-code as follows:

```
repeat
  generate( $\rightarrow$ candidate);
until test(candidate + observations $\rightarrow$ solution);
```

The depth first, solution driven search strategy implemented here is simple and effective if just one solution needs to be found and if the number of classes is limited. Other search regimes can be used if the pragmatics of the task so require. For example, if all solutions are required and the number of classes is limited, a solution driven, candidate set pruning method is appropriate. The next strategy is an example of such a pruning method.

Strategy 2: pure candidate set. An alternative strategy is to begin with the full set of candidates and prune this set by eliminating candidates that are inconsistent with observations found. For this strategy we need a slightly revised version of the inference structure

1. We need an additional inference for selecting an observation from the observation-set.
2. The "test" inference should not produce a solution, but just deliver a truth value, indicating whether the candidate is explained by the observation and the domain theory.

Figure 3 shows this adapted inference structure. Also, the control flow is different and more complicated:

```
while more-solutions (generate) do
  generate( $\rightarrow$ candidate);
    candidate-set := candidate union candidate-set
while more-solutions(select) and |candidate-set| > 1 do
  select(observation-set $\rightarrow$ observation);
  foreach candidate in candidate-set;
```

```
    if test(candidate + observation → false);
    then candidate-set := candidate-set/candidate;
```

Strategy 3: data-driven candidate generation. In Strategy 2, the candidate set is generated blindly: all potential solutions are placed in the candidate set. Experts often use more refined methods for generating the initial candidate set. This third strategy is such an example: the generate step takes as input an observation and generates only those candidates that have this observation as a feature. For the "test" part, the strategy is identical to the previous strategy. Two small changes are required to the inference structure.

1. An observation needs to be input to the "generate" inference.
2. The underlying knowledge used by "generate" in changed from the *class* predicate to the *has features* predicate.

Figure 4 shows the adapted inference structure. The control structure only slightly differs from Strategy 2, namely in the first part:

```
select(observation-set → observation)
while more-solutions(generate) do
  generate(observation → candidate);
  candidate-set := candidate union candidate-set

while more-solutions (select) and |candidate-set| > 1 do
  select(observation-set → observation);
  foreach candidate in candidate-set;
    if test(candidate + observation → false);
    then candidate-set := candidate-set/candidate;
```

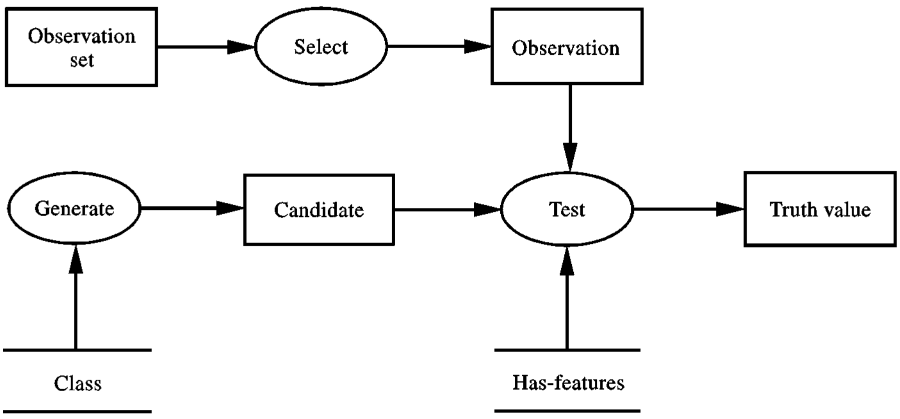

FIGURE 3. Adapted inferences structure for the "prune" strategy.

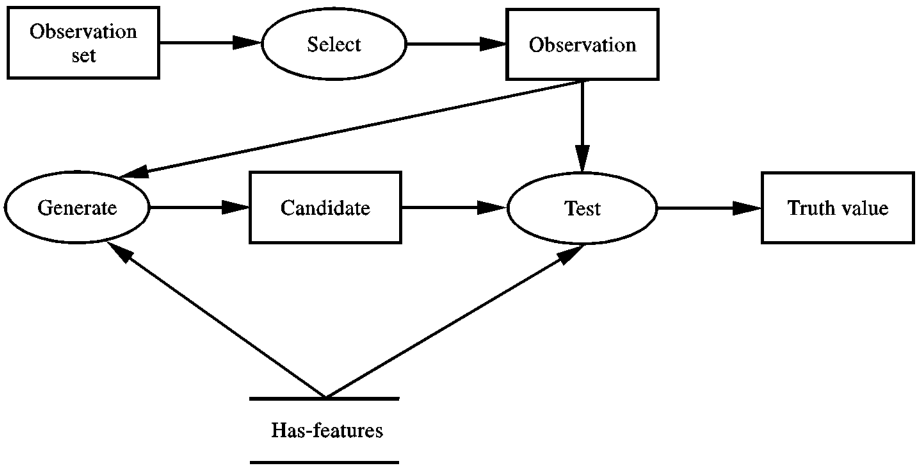

FIGURE 4. Inference structure for strategy 3: data-directed candidate generation.

4. A model elaborate example

This section illustrates the approach through an example. This example is in fact an extension of the classification examples shown in the previous section, applied in the context of the apple-classification domain. The example does not cover the full PSM construction process for the application, but rather focuses on additional interesting issues.

4.1. APPLE CLASSIFICATION-DOMAIN

In this section, the domain of classifying apples is used as the application domain. At a first glance this domain is simple: a few properties are usually sufficient to classify an apple as a member of a particular varietal class. A bright green, firm, medium-sized apple usually is the common "Granny Smith". However, this simplification is deceptive: it may hold for standard apples bought in the average European supermarket, but the task becomes more difficult if we allow a wider range of contexts, such as apples which have been stored for a long period of time, apples picked from wild trees, immature apples or apples from far away places. In addition, if we widen the range of classes to include the many tens of thousands of apple varieties that exist, the task becomes difficult indeed.

The starting point of our journey through apple land will be the domain knowledge: we assume a description of the object (the apple) in terms of easily observable attributes, each with a fixed and limited set of possible values. Table 6 gives a the set of observable attributes and their admissible value sets. Such a table could have been obtained through an initial knowledge acquisition session with an expert on apples.

In addition to the observable attributes, context information is needed. The maturity, cultivation context, handling history and age since harvest, can influence the properties of an apple. Table 7 shows the relevant attributes of this type.

TABLE 6
Observable attributes in the apple domain

Attribute	Values
Background colour	[yellow, yellowish green, pale green, green, bright green]
Foreground colour	[red, pink, bright red, cherry red, dark red, crimson]
Foreground pattern	[patchy, blush, striped, even]
Foreground coverage	[barely, significant, half, largely, full]
Lenticels	[none, small, clearly visible]
Stem length	[short, medium, long]
Texture	[smooth, rough, firm, granular, gritty]
Greasiness	[none, medium, strong]
Calyx	[open, semi-closed, closed]
Sepal angle	[horizontal, vertical, diagonal]
Russet	[none, obscure, near stem only, medium, strong]
Flesh colour	[white, cream, yellowish, red]
Flesh oxidation colour	[white, brown, reddish]
Skin aging	[smooth, ripple]
Shape	[round, conical, oblong, asymmetric, flat]
Surface shape	[smooth, ways, bumps, ribbing, ribs at the crown, uneven]
Size	[small, medium, medium large, large, very large]
Taste	[crisp, sweet, juicy, tart, dry, nutty, aromatic, aniseed, tangy]

TABLE 7
Context attributes in the apple domain

Attribute	Values
Orchard	[commercial, garden, wild]
Crop location	[Northern Europe, Southern Europe, US, South Africa]
Maturity at harvest	[mature, immature, over ripe]
Time since harvest	Number of months
Time since acquisition	Number of weeks
Handling history	[careful, normal, rought]
Storage conditions	[cooled, cellar, kitchen]

The domain theory of the apple domain will contain class definitions of apple properties. An informal description of an apple class could be as follows.

> A typical Golden Delicious is a medium-sized dessert apple. It has a yellow or yellow–green background, sometimes with a slight orange blush. It has a conical shape with characteristic ribs and bumps at the crown. The flavour is excellent, sweet, sharp, juicy. Flesh colour is white. The flavour and size will be disappointing when grown in cool climates such as Northern Europe. In good storage conditions it can keep long. At room temperature its skin ripples and becomes very greasy after a few weeks. Rather prone to russet in wet climates.

The domain knowledge contains properties of the class under standard conditions (e.g. commercially grown, mature at harvest), but also some indications of changes in properties depending on the context. In addition to the domain knowledge concerning observable attributes and context, the class knowledge can have information about other properties such as availability as a function of the time of year, functional properties (a good cooking apple) or information about the parentage of the apple variety (e.g. Jonagold is a crossing of Jonathan and Golden Delicious).

Classes can be organized in a number of ways: hierarchical according to variety, hierarchical according to parentage or according to functional qualities (crap apple, dessert, culinary, universal).

4.2. APPLE APPLICATION TASK

Input to the problem solver that is required is a description of an individual apple in terms of its observable attributes and the context. A second input from the user is a query about the apple class. This query may concern the name of the class or classes that match the case, or a property of the class ("is **this** apple a good cooking apple?"; "what are the storage properties of this apple?").

No data are given before the reasoning process starts: the problem solver will incrementally acquire information from the user. The number of questions should be minimal. The user should be able to indicate that the answer to a question is "unknown". It is also assumed that observations are made by a perfect observer. This means that values assigned to attributes are made conforming to a gold standard and are not subject to doubt or subjective differences. This assumption is by no means a trivial one. Even more or less quantifiable properties such as *size*, *colour* and *stem length* require carefully defined calibration criteria to yield precise and objective observations. Other properties such as *greasiness* and *texture*, are already more difficult to judge. Properties concerning taste are even more difficult to standardize. In later parts of the paper we will return to these issues.

The knowledge of the domain is assumed to be correct and certain, but not necessarily complete.

4.3. STEP 1: COMPETENCE THEORY SPECIFICATION

Along the same line as was discussed in Section 3 we start with the definition of the problem space Ω:

$$\Omega = \langle P, S, DT, solution \rangle. \tag{13}$$

The next step in the construction of $\mathbf{T}_0$ is to define what the elements of the problem space Ω are, leading to the definition of the sub-theories that constitute the problem space. One of the steps to be made in the process of initial problem specification is to abstract from the application knowledge to a more abstract form which is suitable to be mapped onto a problem-oriented theory. This process involves decisions such as: which attributes will be distinguished, are values of attributes unique, do attributes have mutually excluding values.

Given the informal application knowledge presented in Table 6 and 7, we can start with the assumption that all information about the object that needs to be classified will be represented in a set of attribute-value pairs, where we distinguish between observable attributes and context attributes. The value "unknown" is allowed for any attribute.

TABLE 8
Problem definition for classification

Attribute set:	$\{ \cdots a_i \cdots \} = A$	
Value set:	$\forall a_i : \exists \{ \cdots v_{ij} \cdots \} = V_i$	
Feature set:	$\{ \cdots f_{ij} \cdots \} = F$, where	
	$f_{ij} = \langle a_i, v_{ij} \rangle \vee f_{ij} = \langle a_i, \mathit{unknown} \rangle$	
Observational attributes:	$A_0 \subset A$	
Context attributes:	$A_c \subset A$	
Derived attributes:	$A_d \subset A$	(14)
Feature set partitions:	$F_x = \{ \cdots f_{ij} \cdots \}$ where	
	$f_{ij} = \langle a_i, v_{ij} \rangle \wedge a_i \in A_x, x = o, c, d$	
Observation set:	$\{ \cdots o_i = \langle a_i, v \rangle \cdots \} = O$ where $a_i \in A_0 \cup A_c$	
Single-value observations:	$o_i \rightarrow \exists ! k$ such that $o_i = f_{ik}$	
Case reference set:	$\{ \cdots r_i \cdots \} = R$	
Problem set:	$\{ \cdots p_i = \langle r_i, a_d \rangle \cdots \} = P, r_i \in R, a_d \in A_d$	

Although some of the attributes in Table 6, such as taste, can be multivalued (e.g. taste = sweet and taste = juicy), we will formalize observations as single-valued features, as we did in Section 3. This assumption is valid if we assume that multivalued attributes can be transformed into a set of equivalent attributes that can only have a single value (e.g. juiciness = yes, sweetness = high). In addition to the attributes that can be asked from the user, we need knowledge about attributes for which the values are derived from the class definitions. These attributes can be specified in the query that the user poses.

The fact that observations are not assumed to be given before the reasoning process starts is somewhat difficult to capture is a static theory. One possible solution to this problem would be to model the dynamics of the information gathering process and define the problem as a dynamic object. A simpler, but less elegant solution is to define a problem as a reference to the object to be classified and parameterize observations with respect to this reference. A problem can then be represented as a tuple of the case reference and an attribute representing the query. Table 8 defines the elements of the problem space.

Table 9 defines the solution space. Classes are represented as simple propositions. A solution is a disjunction of classes that apply to the reference object, and a feature that assigns a value to the query attribute.

Class definitions in the apple domain are context dependent. Thus, we introduce the concept *context* to represent classes of contexts that affect the properties of the apple classes. Typical examples of contexts would be *cool_climate*, *fresh_commercial* or

TABLE 9
Solution and domain-theory definition for classification

Set of classes:	$\{c_i\} = C$	
Solution set:	$\{s_i = \langle CS, \langle a_i, v_{ij}\rangle\rangle\} = S,$	(15)
	$a_i \in A_d, v_{ij} \in V, CS \subseteq C$	

fresh_home_grown. Rules for determining contexts and the use of contexts in class definitions could be written in ifthen form along the following lines.

```
IF   orchard = commercial
  AND maturity at harvest = mature
  AND time since acquisition = 0
  THEN   Fresh-commercial

  IF   Golden Delicious
    AND Fresh-commercial
  THEN   background colour = yellow
    OR   yellow–green
```

Given this conceptualization of the domain knowledge, we can formalize the domain theory as in Table 10.

The next step is to formulate what actually may count as a solution to the posed problem. In our framework, this boils down to making an assertion about the relation between the *solution* predicate and the other elements in the problem space description Ω. Apart from Ω, an additional input to the solution specification step are "solution criteria": general requirements or constraints, often at first stated only informally, that are imposed on the solution. They may be of the type that a solution must be optimal or only satisfying. In the context of classification, examples of solution criteria may be whether or not a single solution is required, whether consistency between all observations and solutions is imposed, etc. These solution criteria more or less delineate which paradigms to choose within various possible approaches in the problem-solving process.

TABLE 10
Domain-theory definition for apple classification

Set of contexts:	$\{ \cdots e_i \cdots \} = E$	
Set of context definitions:	$\{ \cdots de_{ij} \cdots \} \in DT$ where	
	$de_{ij} = \bigwedge_{kl} f_{kl} \in F_c \rightarrow e_i$	(16)
Set of class definitions:	$\{ \cdots d_{ij} \cdots \} \in DT$ where	
	$d_{ij} = c_i \wedge e \rightarrow \bigvee_k f_{jk}, c_i \in C, e \in E, f_{jk} \in F_o$	

For a given problem space description Ω, inclusion of different solution criteria leads to different solution specifications.

In order to define the solution criteria for the example task, we need to introduce the set of observations that is gathered during the problem-solving process. We do this by introducing a new predicate *solve-query*. Given $p = \langle r_p, a_p \rangle \in p, s \in S$ is the solution to this problem p with O_p being a set of observations with respect to the reference object r_p:

$$solution(p, s) \leftrightarrow \exists O_p \; solve\text{-}query(p, O_p, s). \tag{17}$$

We first define a predicate *solves* that generates candidate classes. In defining the predicate *solve* we introduce the assumption that not all observations need to be explained by the solution, only a subset O_s. In addition, we require the full set of observations to be consistent with the domain theory DT. This solution criterion is weaker than the one specified in Section 3 and corresponds to the notion of *positive coverage* (Stefik, 1995). If only consistency would be required, the solution specification would be based on *conservative inclusion*.

$$\begin{aligned} solves(p, O, s) \leftrightarrow \; & s \cup DT \models O_s \subseteq O \\ & \wedge \; s \cup DT \cup O \not\models \bot \end{aligned} \tag{18}$$

No specification of the set O_s has been given here. There are several options. In diagnostic reasoning, O_s is often chosen to be the set of abnormal observations. In classification tasks, we could choose a fixed set of discriminating properties or we could dynamically determine O_s as the set of observations that is minimally needed to find a single solution. For the time being we leave the choice of O_s open in $\mathbf{T}_0$.

4.4. STEP 2: COMPETENCE THEORY REFINEMENT

Given the domain theory definition in $\mathbf{T}_0$, we can introduce a number of predicates that link the formal definitions to elements that will occur in the knowledge model of the operational problem solver (Table 11).

TABLE 11
Introduction of domain theory predicates in $\mathbf{T}_1$

$$class(c_i) \leftarrow \; c_i \in C \tag{19}$$

$$conditions(e_i, F) \leftarrow \; isdefined\left(\bigwedge F \rightarrow e_i \right) \wedge e_i \in E \tag{20}$$

$$\begin{aligned} has\_features(c_i, a_j, e, V_j) \leftarrow \; & class(c_i) \wedge \\ & V_j = \left\{ v_{jk} \,|\, isdefined\left(c_i \wedge e \rightarrow \bigvee_k f_{jk} \right) \right. \\ & \left. \wedge f_{jk} = \langle a_j, v_{jk} \rangle \right\} \end{aligned} \tag{21}$$

TABLE 12
Introduction of inference predicates in $\mathbf{T}_1$

$$
\begin{aligned}
\mathit{context}(O, e_i) \leftarrow\ & \mathit{conditions}(e_i, f) \wedge \\
& \forall f \in F f \in O && (22)\\
\mathit{explains}(O, c, \langle a, v \rangle) \leftarrow\ & \mathit{class}(c) \wedge \\
& \mathit{context}(O, e) \wedge \\
& \mathit{has\ features}(c, a, e, V) \wedge \\
& v \in V && (23)\\
\mathit{consistent}(O, c, \langle a, v \rangle) \leftarrow\ & \mathit{class}(s) \wedge \\
& \mathit{context}(O, e) \wedge \\
& v = \mathit{unknown} \vee (\mathit{has\ features}(c, a, e, V) \rightarrow v \in V) && (24)
\end{aligned}
$$

Statements like in Equation 20 and 21 can be viewed as *KB schemata*: definitions of structures in the domain knowledge. In formal terms, the above gives axiom schemes for the domain statements (right-hand side), a method-oriented meta-theory (left-hand side), connected through a naming relation (a mapping that may be implemented via rewrite rules) (van Harmelen & Balder, 1992; Akkermans *et al*., 1993*a*). The power of this type of meta-description is that it is coupled to domain knowledge, but in a generic and reusable fashion. This makes our approach useful in tackling the so-called indexing problem: how to annotate problem-solving methods such that they are usable and reusable in a library of generic KBS components (Klinker *et al*., 1991).

Knowledge-base schemata make explicit assumptions about the nature of the knowledge base e.g. a set of implication rules are interpreted as class definitions.

The next part of the refinement step is to explicate the decomposition of the solution relation, by introducing additional relations $\{q_i\}$, e.g. $\{q_i\}$ = {context, explains, consistent} (see Table 12).

We have now enough machinery to define the *solves* predicate in terms of the inference predicates. Table 13 is the refined version of Equations 17 and 18 in $\mathbf{T}_0$. The predicate *solve*-*query* uses *solves* to generate a set of candidates and *solve*-*query* computes the intersection of the values of the query attribute. The predicate succeeds if a singleton set remains.

Each of the statements on the right-hand side of Table 13 specifies a paradigmatic aspect of the problem-solving method: each statement represents an essential step in the reasoning process that the PSM must perform. Of course, one could make other paradigmatic choices, which would then lead to different specifications and thus to a different PSM. The essential point however is that conceptually refining a PSM competence theory is an activity that makes the underlying problem-solving paradigms as well as support knowledge assumptions explicit.

Clearly, theory $\mathbf{T}_1$ is much closer to an operational method than the top-level specification $\mathbf{T}_0$ of the previous subsection. On the other hand, although all conceptual ingredients are available now, the competence theory cannot yet be seen as fully

TABLE 13
Decomposition of the solves and solve-query predicates

$$solves(p, O, s) \leftarrow \exists O_s \; obs\_subset(O_s, O) \wedge \forall_o \in O_s \; explains(O, s, o) \wedge \forall'_o \in O \; consistent(O, s, o') \quad (25)$$

$$solve\text{-}query(\langle r_p, a_p \rangle, O_p, \langle CS, v \rangle) \leftarrow CS = \{s_i \mid solves(\langle r_p, a_p \rangle, O_p, s_i)\} \wedge context(O_p, e) \wedge \{v\} = \bigcap_{V_i} \forall c_i \in CS \; has\,features\,(c_i, a_p, V_i) \quad (26)$$

operational. The various quantifications in Equations 25 and 26 need to be translated to proper control structures in the operational PSM. Thus, we might say that the above competence theory represents a conceptual breakdown of what counts as a solution to our problem, but that we still are in need of an operational breakdown. The final step, leading to an operational inference and control structure for a PSM, is discussed in the next subsection.

4.5. STEP 3: OPERATIONALIZATION

So, we have to find an operationalization of the refined competence theory, in terms of a set of inferences, roles and a specification of the strategy (the control structure). Earlier, we saw three strategies for imposing control on the inference structure. Since the example task requires that a query can be answered on the basis of a (partial) classification, the strategy to prune the candidate set until a unique answer can be derived is most plausible. In the definition of the task, we assume that the context already has been determined and that the set of observations that must be explained is explicitly given in the knowledge base (obs-subset).

```
task apple-query-answering;
  input:
    obs-subset, context, query;
  output:
    query-value, candidate-set;
  specification:
    "Find the subset of apple classes that share a unique value for the query
    attribute";
end task apple-query-answering;
```

The task method uses an iteration **similar** to the one used in Section 3 with a different stop condition: the while loop finishes when the value for the query attribute is the same for all candidate classes (inferred by the compare inference). Another difference with the strategies in Section 3 is the differentiation between test-examples and test-consistent on the basis of the type of observation.

```
task-method pure-maximal-set;
  realizes: apple-query-answering;
  decomposition:
    inferences: generate, compare, select, test-explains, test-consistent,
                specify;
  control-structure:
    while more-solutions(generate) do
      generate(→ candidate);
      candidate-set := candidate union candidate-set;

  while compare(candidate-set + query → false) do
    select(observation-set → observation);
    foreach candidate in candidate-set;
      if observation in obs-subset
        then test-explains(candidate + context + observation
              → truth-value);
        else test-consistent(candidate + context + observation
              → truth-value);
        if truth-value = false
        then candidate-set := candidate-set/candidate;

  specify(candidate-set + query → query-value);
end task-method pure-maximal-set;
```

The corresponding inference structure is shown in Figure 5.

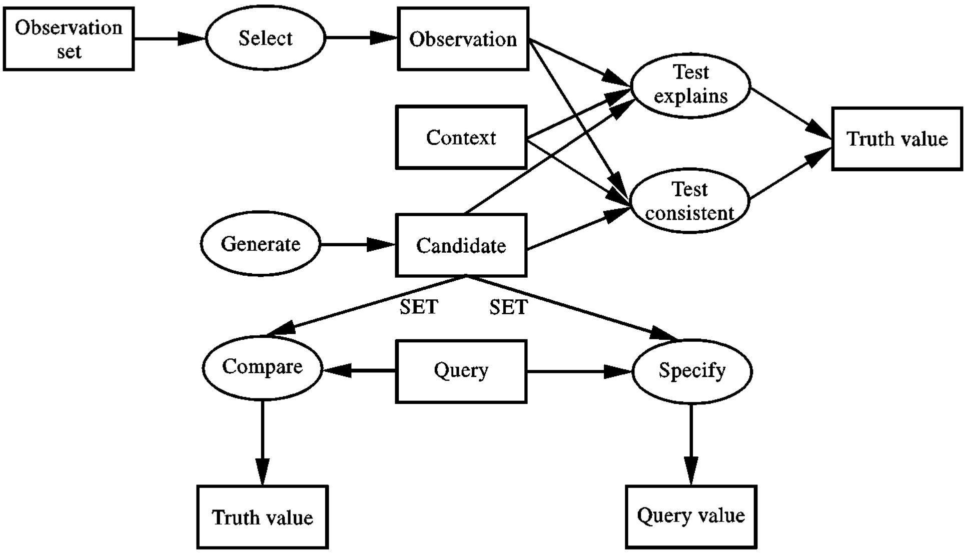

FIGURE 5. Inference structure for the chosen operationalization of the apple classification sub-task.

5. Conclusions

In this paper, we have described a method for the systematic construction of PSMs based on a successive refinement of a formal theory of the competence required from a PSM, resulting in an operational knowledge model of the PSM. We explicated the various conceptualization and operationalization steps that have to be taken in order to arrive from an informal problem statement to a full knowledge model of a PSM. The power of the approach sketched in this paper is that it explicates the ontological commitments and assumptions behind the problem-solving methods that are employed in systems that solve a particular class of tasks.

In earlier work, we have used the competence theory approach to analyse well known PSMs such as Cover-and Differentiate (Akkermans *et al.*, 1993*a*, 1994) and propose-and-revise (Wielinga *et al.*, 1995). The first exercise showed that various inferences and problem solving strategies in existing systems such as MOLE (Eshelman *et al.*, 1988) can be formally specified as refinements of a general competence theory of diagnosis. The latter work showed that the various solutions to the VT problem (Marcus *et al.*, 1988; Yost, 1992; Schreiber & Birmingham, 1996) can be viewed as different refinements and operationalizations of an initial competence theory for parametric design. In this paper, we have used various variants of a classification task to illustrate the competence theory refinement approach. Again, the approach highlights and explicates assumptions underlying the various PSMs for classification.

The explication of the commitments and assumptions made during the competence refinement process places strong constraints on the form and content of the knowledge represented in the domain theory. As such, the formalization process outlined in this paper also poses strong constraints on the knowledge acquisition process. For example, the assumptions explicited in this paper can be viewed as a formal basis for techniques that construct models of reasoning processes through a refinement process such as the GDM approach (van Heijst *et al.*, 1992).

The formalization process as used in this paper is not without problems. In earlier work, we have concluded that the non-monotonic nature of the *propose and revise* method is difficult to capture in intuitively understandable theories. In the work described in this paper, we have found that the mapping from the formal theories to the control specifications in the operational knowledge model, can involve quite complex transformations, which appear difficult to formalize. We are convinced however, that further work in formalization of PSMs will ultimately lead to a better understanding of what the competence of knowledge-based systems really is. Developing problem-solving methods and systems that empirically show their heuristic value is one thing, but ultimately the scientific goal of the field of knowledge engineering should be to understand what problems these systems can solve and what problems they cannot solve.

References

Aben, M. (1995). *Formal methods in knowledge engineering.* Ph.D. Thesis, University of Amsterdam, Faculty of Psychology. ISBN 90-5470-028-9.

Akkermans, J. M., van Harmelen, F., Schreiber, A. T. & Wielinga, B. J. (1993*a*). A formalisation of knowledge-level models for knowledge acquisition. *International Journal of Intelligent Systems*, **8,** 169–208. Reprinted in: K. M. Ford, & J. M. Bradshaw, Eds. (1993). *Knowledge Acquisition as Modelling.* New York: Wiley.

AKKERMANS, J. M., WIELINGA, B. J. & SCHREIBER, A. T. (1993*b*). Steps in constructing problem-solving methods. In N. AUSSENAC, G. BOY, B. GAINES, M. LINSTER, J.-G. GANASCIA & Y. KODRATOFF, Eds. *Knowledge Acquisition for Knowledge-Based Systems. Proceedings of the 7th European Workshop EKAW'93*. Toulouse and Caylus, France, in Lecture Notes in Computer Science, Vol. 723, pp. 45–65. Germany: Springer-Verlag.

AKKERMANS, J. M., WIELINGA, B. J. & SCHREIBER, A. T. (1994). Steps in constructing problem-solving methods. In B. R. GAINES & M. A. MUSEN, Eds. *Proceedings of the 8th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop. Vol. 2: Shareable and Reusable Problem-Solving Methods*, pp. 29.1–29.21. Alberta, Canada: University of Calgary: SRDG Publications.

BENJAMINS, V. R. (1993). *Problem solving methods for diagnosis*. Ph.D. Thesis, University of Amsterdam, Amsterdam, The Netherlands.

BREUKER, J. A. & VAN DE VELDE, W., Eds. (1994). *The CommonKADS Library for Expertise Modelling*. Amsterdam, The Netherlands: IOS Press.

CHANDRASEKARAN, B. (1988). Generic tasks as building blocks for knowledge-based systems: the diagnosis and routine design examples. *The Knowledge Engineering Review*, **3,** 183–210.

CHANDRASEKARAN, B., JOHNSON, T. R. & SMITH, J. W. (1992). Task-structure analysis for knowledge modeling. *Communications of the ACM*, **35,** 124–137.

CLANCEY, W. J. (1992). Model construction operators. *Artificial Intelligence*, **53,** 1–115.

CONSOLE, L. & TORASSO, P. (1990). Integrating models of the correct behaviour into abductive diagnosis. In L. C. AIELLO, Ed. *Proceedings ECAI-90*, pp. 160–166. London: ECCAI, Pitman.

ESHELMAN, L., EHRET, D., MCDERMOTT, J. & TAN, M. (1988). MOLE: a tenacious knowledge acquisition tool. In J. H. BOOSE & B. R. GAINES, Eds. *Knowledge Based Systems, Vol. 2*: *Knowledge Acquisition Tools for Expert Systems*, pp. 95–108. London: Academic Press.

FENSEL, D. (1995*a*). Assumptions and limitations of a problem-solving method: A case study. *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW'95)*. Banff, Canada.

FENSEL, D. (1995*b*). *The Knowledge Acquisition And Representation Language KARL*. Boston: Kluwer Academic Publisher.

FENSEL, D. & GROENBOOM, R. (1996). Mlpm: defing a semantics and axiomatization for specifying the reasoning process of knowledge-based systems. *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*. Budapest.

FENSEL, D., SCHOENEGGE, A., GROENBOOM, R. & WIELINGA, B. (1996). Specification and verification of knowledge-based systems. *Proceedings of the Workshop on Validation, Verification and Refinement of Knowledge-Based Systems, 12th European Conference on Artificial Intelligence (ECAI-96)*. Budapest.

KLINKER, G., BHOLA, C., DALLEMAGNE, G., MARQUES, D. & MCDERMOTT, J. (1991). Usable and reusable programming constructs. *Knowledge Acquisition*, **3,** 117–136.

MCDERMOTT, J. (1988). Preliminary steps towards a taxonomy of problem-solving methods. In S. MARCUS, Ed. *Automating Knowledge Acquisition for Expert Systems*, pp. 225–255. Boston: Kluwer.

MARCUS, S., STOUT, J. & MCDERMOTT, J. (1988). VT: an expert elevator designer that uses knowledge-based backtracking. *AI Magazine*, 95–111.

MUSEN, M. A. (1989). *Automated Generation of Model-Based Knowledge-Acquisition Tools*, Research Notes in Artificial Intelligence. London: Pitman.

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

POOLE, D. L. (1988). Representing knowledge for logic-based diagnosis. *Proceedings of the 5th International Conference on Generation Computing Systems*, pp. 1982–1290. Tokyo.

REITER, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, **32,** 57–96.

SCHREIBER, A. T. & BIRMINGHAM, W. P. (1996). The Sisyphus-VT initiative. *International Journal of Human–Computer Studies*, **43,** 275–280 (Editorial special issue).

SCHREIBER, A. T., WIELINGA, B. J., DE HOOG, R., AKKERMANS, J. M. & VAN DE VELDE, W. (1994). CommonKADS: a comprehensive methodology for KBS development. *IEEE Expert*, **9,** 28–37.

STEELS, L. (1990). Components of expertise. *AI Magazine*.

STEFIK, M. (1995). *Introduction to Knowledge Systems.* Los Altos, CA: Morgan Kaufmann.

TEN TEIJE, A. (1997). *Automated configuration of problem solving methods in diagnosis. Ph.D. Thesis*, SWI, University of Amsterdam.

TEN TEIJE, A., VAN HARMELEN, F., SCHREIBER, A. T. & WIELINGA, B. J. (1996). Construction of problem-solving methods as parametric design. In B. R. GAINES, & M. A. MUSEN Ed. *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop Alberta, Canada*, 9–14 November, Vol. 1, pp. 12.1–12.21. University of Calgary: SRDG Publications.

VAN DE VELDE, W. (1988). Inference structure as a basis for problem solving. In Y. KODRATOFF, Ed. *Proceedings of the 8th European Conference on Artificial Inteligence*, pp. 202–207. London: Pitman.

VAN HARMELEN, F. & BALDER, J. R. (1992). $(ML)^2$: a formal language for KADS models of expertise. *Knowledge Acquisition*, **4** (Special issue: 'The KADS approach to knowledge engineering', required in A. TH. SCHREIBER, *et al.*, Eds. (1993). KADS: *A Principled Approach to Knowledge-Based System Development.*

VAN HEIJST, G., TERPSTRA, P., WIELINGA, B. & SHADBOLT, N. (1992). Generalised directive models. *Proceedings of KAW-92.* Banff.

WIELINGA, B. J., AKKERMANS, J. M. & SCHREIBER, A. T. (1995). A formal analysis of parametric design problem solving. In B. R. GAINES, & M. A. MUSEN Ed. *Proceedings of the 8th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Vol. II, pp. 37.1–37.15. Alberta, Canada. University of Calgary: SRDG Publications.

WIELINGA, B. J., SCHREIBER, A. T. & BREUKER, J. A. (1992). KADS: a modelling approach to knowledge engineering. *Knowledge Acquisition*, **4,** 5–53 (Special issue 'The KADS approach to knowledge engineering'. Reprinted in: BUCHANAN, B. and WILKINS, D. Eds. (1992), *Readings in Knowledge Acquisition and Learning*, pp. 92–116. San Meteo, CA: Morgan Kaufmann).

YOST, G. (1992). *Configuring elevator systems.* Technical Report, Digital Equipment Corporation, 111 Locke Drive (LMO2/K11), Marlboro, MA 02172, USA.