Appears in ***International Journal of Human-Computer Interaction*, Vol. 49, No. 2, pp.121-157, August, 1998.**

Knowledge Modeling Directed by Situation-Specific Models

June 3, 1998

Michel Benaroch
School of Management
Syracuse University
Syracuse, NY 13244
mbenaroc@syr.edu
(315) 443-3492

Abstract

Clancey (1992) proposed the model-construction framework as a way to explain the reasoning of knowledge-based systems (KBSs), based on his realization that all KBSs construct implicit or explicit situation-specific *models* (SSMs). An SSM is a rational argument that explains the solution produced for a specific problem situation pertaining to a target application task (e.g., SSMs constructed for typical diagnosis tasks are causal arguments having the structure of a proof). From a knowledge engineering perspective it makes sense that the notion of an SSM should play a major role in the modeling of tasks. Motivated by this view, we present a structured knowledge modeling methodology called ***SSM-DKM*** – *SSM-Directed Knowledge Modeling*. In SSM-DKM, an SSM is a central structure that drives the entire modeling endeavor. In light of this fact, we explain how SSM-DKM supports three main stages in the knowledge engineering process – conceptualization, formalization and validation, and instantiation – and illustrate the application of SSM-DKM to a medical diagnosis task. The knowledge model that SSM-DKM produces for a target application task has two appealing traits. First, the model embodies explicit knowledge about the ontology of SSMs that the task entails creating as solutions, thus enabling the construction of a KBS that makes these SSMs explicit. Second, the model captures strategic (or problem-solving) knowledge in declarative terms pertaining to the ontology of SSMs created for the task. Both these beneficial traits have been illustrated in the context of ACE-SSM, a KBS architecture that constructs explicit SSMs (Benaroch, 1998).

1. Introduction

Clancey (1992) conceptually proposed the *model-construction framework* as a way to explain the reasoning of knowledge-based systems (KBSs). This framework considers problem solving to be a modeling activity, based on the realization that all KBSs construct implicit or explicit *situation-specific models* (SSMs). An SSM is a rational argument that explains the solution produced for a specific problem situation pertaining to some target application task (e.g., SSMs constructed for typical diagnosis tasks are causal arguments having the structure of a proof).

From a knowledge engineering perspective, it seems that the notion of an SSM should play a role in the modeling of application tasks. Indeed, leading knowledge modeling approaches, including Components of Expertise (Steels, 1993; 1990) and CommonKADS (Wielinga, Van de

Velde, Schreiber and Akkermans, 1993), already consider SSMs (or *case models* as they call them) during the modeling of an application task. For example, Components of Expertise uses SSMs to identify domain knowledge and operations needed to solve an application task. However, these approaches do not yet incorporate the notion of an SSM into the knowledge models that they produce.

A knowledge model reflecting the nature of SSMs created for a target application task is valuable mainly for one reason – it enables building a KBS that makes those SSMs explicit. An explicit SSM can serve as a working memory because it constitutes a complete trace of what a KBS did to produce final and intermediate inference results in the course of solving a specific problem situation. As such, an explicit SSM permits the KBS that creates it to explain the solutions it embeds, explain the problem-solving behavior that produced these solutions, and dynamically reflect on this behavior during problem solving (Benaroch, 1998; Clancey, 1992).

Van de Velde (1993) identified a related source of value from the ability to create explicit SSMs. He explains that KBS architectures that support the process of problem solving as modeling by constructing explicit SSMs could reflect more of the real nature of problem-solving from a knowledge-level perspective. One architecture for constructing explicit SSMs is ***ACE-SSM*** (Benaroch, 1998). ACE-SSM centers its reasoning around *goal knowledge* reflecting the ontology, or underlying structure, of SSMs that an application task aims to produce. Consequently, in ACE-SSM, strategic control decisions are a direct result of the current state of knowledge about the problem, reflected in the SSM being created. More specifically, where problem solving evolves an initial SSM (state) reflecting what the solver initially knows about the problem into a goal SSM (state) with some specific underlying structure described by goal knowledge, ACE-SSM grounds all strategic control decisions in the need to evolve an SSM into the goal state.

As we show later, the ability to model strategic (or problem-solving) knowledge in declarative terms pertaining to goal knowledge is a key benefit from grounding all strategic control decisions in the need to evolve an explicit SSM into a state specified by goal knowledge. Expressing strategic knowledge declaratively is known to be a difficult and practical knowledge engineering problem. Currently, none of the leading knowledge modeling approaches produces knowledge models that capture strategic knowledge declaratively, including CommonKADS (Wielinga, Schreiber and Breuker, 1992), Steels' Components of Expertise, the Task-Structure Framework (Chandrasekaran, Johnson and Smith, 1992), and Problem-Solving Methods (e.g., Marcus and McDermott, 1989; Eriksson, Shahar, Tu, Puerta and Musen, 1995; Fensel and Straatman, 1998). The reason is that these approaches focus their modeling effort mainly on *how* a task is solved, or on the solution strategy applied to the task. This contrasts with the focus that the model-construction framework puts on *what* a task entails producing (i.e., SSMs). Van de Velde (1994) and Breuker (1997) have suggested the need to link *what* a task entails with *how* the task is solved, in the context of creating a library of reusable problem-solving methods (or components) for CommonKADS. Recognizing that these methods implicitly create arguments of a specific structure, analogous to SSMs whose underlying structure is specified by what we call goal knowledge, they argue that linking *what* with *how* could be beneficial for the indexing and retrieval of reusable methods. Overall, while these authors recognize the need to link *what* with *how*, they do not motivate this need by a desire to model strategic knowledge in declarative terms or by a desire to explicitly integrate the notion of an SSM (or what they call *argument structure*) into the knowledge models that CommonKADS produces.

This paper presents a structured knowledge modeling methodology called *SSM-Directed Knowledge Modeling* – ***SSM-DKM*** – which is grounded in the model-construction framework. In SSM-DKM, an SSM is a central entity whose underlying structure drives the construction of a knowledge model in which goal knowledge is captured explicitly and strategic knowledge and

domain knowledge are modeled in declarative terms pertaining to goal knowledge. SSM-DKM specifically addresses the issues involved in three knowledge engineering stages:

(1) conceptualization: how to create a conceptual knowledge model that embodies goal knowledge reflecting the structure of SSMs sought, domain knowledge (or data) used to populate SSMs, and strategic knowledge for guiding the construction process of SSMs?
(2) formalization and validation: how to formalize a conceptual knowledge model (capturing goal, strategic, and domain knowledge), and how to check for its consistency, compactness, and completeness?
(3) instantiation: how to operationalize a formalized knowledge model by mapping it to representational constructs that can interact with an architecture like ACE-SSM?

The paper is organized as follows. Section 2 reviews the conceptual framework underlying SSM-DKM. It first explains the notion of an SSM, then frames the construction process of SSMs within Newell's (1990) problem-space paradigm, and finally outlines the modeling stages in SSM-DKM. The next three sections elaborate on these stages: Section 3 deals with conceptualization, Section 4 deals with formalization and validation, and Section 5 deals with instantiation. Section 6 positions SSM-DKM within the area of knowledge modeling, relative to the main knowledge modeling approaches. Section 7 offers concluding remarks and directions for future research.

2. Conceptual Framework

This section presents the conceptual underpinning of our knowledge modeling methodology, SSM-DKM. It first reviews the notion of SSMs, by looking at an example pertaining to a medical diagnosis task. It then shows how the reasoning process underlying the construction of SSMs can be explained using the problem-space paradigm (Newell, 1990). Finally, it provides an overview of the stages in SSM-DKM, from the creation of a conceptual knowledge model of a task to the instantiation of this model for use with the ACE-SSM architecture.

2.1. Situation-Specific Models (SSMs)

To better understand what SSMs are, it is useful to begin with an example. Figure 1 shows an SSM that was created based on the think-aloud protocol that Clancey (1988) provides for a sample problem situation pertaining to the medical diagnosis task NEOMYCIN solves. (Table 1 contains much of this protocol.) The diagnostic reasoning process underlying this SSM is as follows. It starts with the collection of initial patient data. The patient has a two-week history of headache, nausea, vomiting, and diplopia (double vision). These initial findings (symptoms) are next mapped to the more specific findings: headache-duration, nausea-duration, vomiting-duration, diplopia-duration. Since diplopia is an abnormal finding, it is the first to be pursued. It is generalized into the serious-CNS (central nervous system) finding, which is then found to be plausibly caused by increased-pressure-in-the-brain. In turn, it is found that increased-pressure-in-the-brain could be caused by some type of brain-mass-lesion: a brain-tumor, hematoma (collection of blood), or collection-of-pus. The serious-CNS-finding also provides evidence for meningitis being another plausible causing disease. But, given the duration of this finding, chronic-meningitis is next found to be a more specific plausible cause. At this point, there are two competing disease hypotheses: brain-mass-lesion and chronic-meningitis. More generally, the former is a focal-process, while the latter is an infectious-process. Since the picture so far is more suggestive of a focal lesion in the brain (at least in the mind of the expert whose think-aloud protocol is shown in Table 1), focal-process becomes the focus of the diagnostic process. This process continues to expand the SSM in the same fashion, until it finds a disease that can explain all patient symptoms.

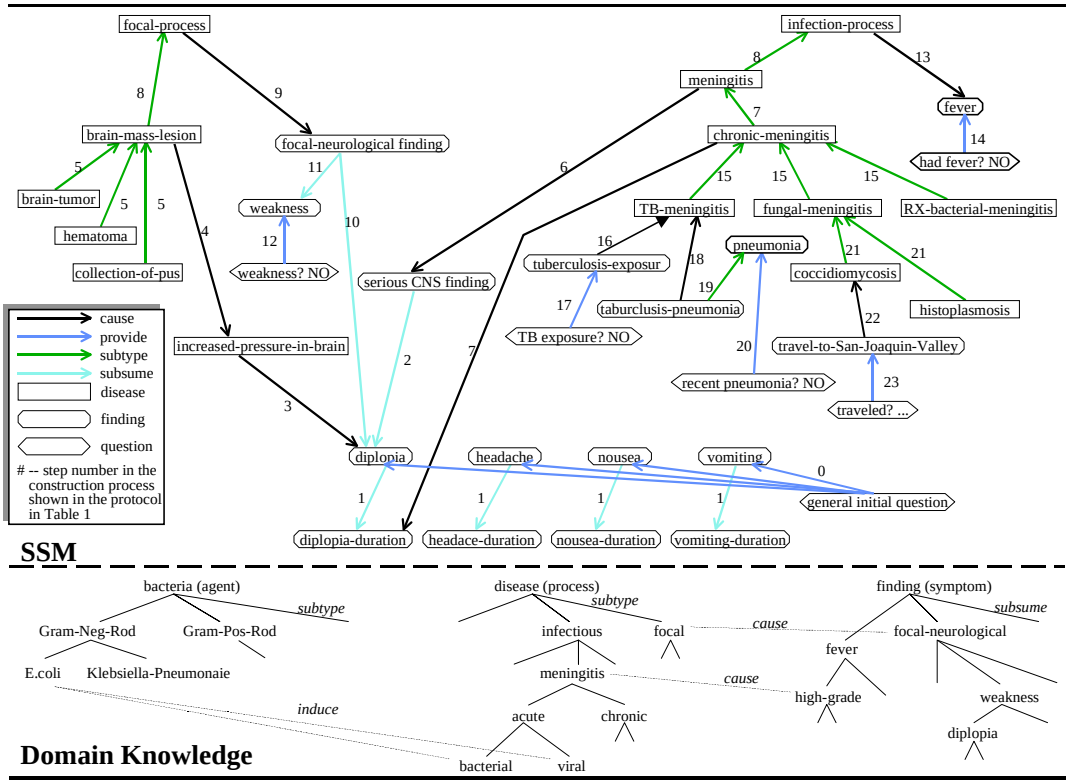


Figure 1: sample SSM and domain knowledge used to construct it

An important observation follows from the above sample SSM. Granted that the domain knowledge used to create this SSM comprises relational networks, we see that an SSM is essentially a directed graph that gradually grows links between nodes depicting domain objects (diseases, findings, etc.) and relations (cause, subtype, etc.) found to be relevant to the specific problem situation solved. In fact, an SSM models the domain phenomenon with which the specific problem situation solved is concerned. More specifically, in our sample medical diagnosis task, once the sample SSM in Figure 1 is completed, it would capture a causal argument showing what has happened to bring about the patient illness, on the premise that a disease is a bodily process that follows a specific causal script (Clancey, 1988). Section 3 elaborates on this notion of a disease process and how it is used for reasoning purposes.

The above observation raises two questions. First, do all SSMs created for a specific task have a coherent underlying structure? For sufficiently well understood tasks, the answer is yes. For example, in medical diagnosis, viewing a disease as a process that follows a specific causal script implies certain well-defined relations between domain objects that must hold true in every SSM created for this task. To illustrate, knowing that every abnormal finding must be explained by some causing disease(s), it follows that every SSM node depicting an abnormal finding must be linked to an SSM node depicting a disease. This illustrative relation is basically a "constraint" that every SSM created for the task must satisfy. We return to this question to address it in greater detail in Section 3.

Another question that follows from the above discussion is: what paradigm of intelligent problem-solving can explain the construction process of SSMs? This question is addressed next.

2.2. SSMs' Construction Process

The reasoning process underlying the construction of SSMs can be explained by Newell's problem-space paradigm of intelligence (Newell, 1990). As seen in Figure 2, this paradigm considers problem-solving to be a process that involves two searches – problem search and knowledge search.

- *Problem search* assumes the existence of a problem space for the task solved. This implies the availability of: (1) a goal state *g* that describes what needs to be known at the end of problem solving; (2) an initial state *i* that describes what is known at the start of problem solving; and (3) a set of operators *P* that can be used to generate new states in the problem space. As the outer loop in Figure 2 indicates, starting from state *i*, problem search applies one operator in *P* at a time to generate a new state in the problem space, with the hope that this state will match (or at least be closer to) the goal state *g*. Thus, problem search uses operators in *P* to generate states in the problem space that form a path from state *i* to state *g*.
- *Knowledge search* occurs in the inner loop of problem search, as seen in Figure 2. It guides problem search by using search control knowledge to select the operator in *P* that should be applied to generate the next state on the path from *i* to *g*. Knowledge search has usually associated with it a fix space whose structure (connectivity and how paths through it are described) pre-exists. That is, while problem search occurs in a space that is constructed dynamically, knowledge search typically occurs in a fix pre-existing structure.

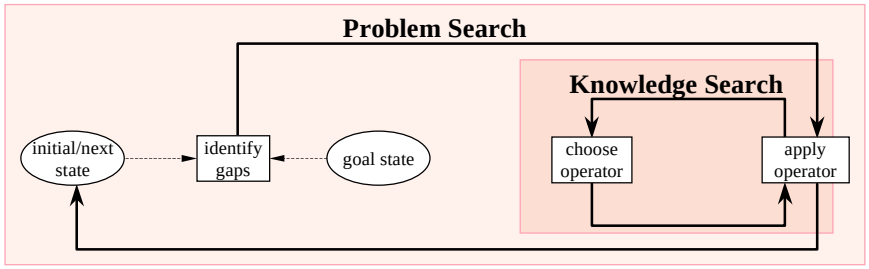


Figure 2: Newell's (1990) problem-space paradigm of intelligence

Framing the construction process of SSMs within the problem-space paradigm requires reliance on several postulates. First, where the goal of solving a task is to create an SSM of every problem situation the task involves, let ***G*** denote knowledge reflecting the structure that these SSMs must possess. This structure is described in terms of "constraints" that the SSMs must satisfy, or domain relations that must hold true in the SSMs. Second, relative to problem search, it can be said that ***G*** describes the goal state in the problem space, while the evolving SSM being created for a given situation describes the current state. Finally, search control knowledge is essentially a set of general-purpose and/or task-specific control principles, which produce the search strategy that determines the order by which states in the problem space are traversed (e.g., depth-first, breadth-first). For this reason, we will also refer to search control knowledge as strategic knowledge.

With these postulates in mind, we summarize below a conceptual process that constructs explicit SSMs by applying three iterative steps corresponding to the problem search and knowledge search depicted in Figure 2.

Step 1: In problem search (outer loop), inspect the evolving SSM (current state) to identify gaps relative to ***G*** (goal state), and post the gaps in the SSM as unsatisfied subgoals that are represented as node-chains. For example, consider an SSM with one abnormal finding node bound to diplopia, denoted (*F* diplopia). Here, one gap is the lack of a disease node in the SSM that explains this abnormal finding. Respectively, an unsatisfied subgoal is posted in the SSM

as the node-chain (*D* ?)→(*F* diplopia), where (*D* ?) is an unbound disease node.

Step 2: In knowledge search (inner loop), given the current SSM posting all identified gaps as unsatisfied subgoals, use strategic knowledge to choose which subgoal to satisfy next by producing a new state that closes one of the gaps in the SSM. For example, suppose that the choice has to be between two unsatisfied subgoals represented by the node-chains (*D* ?)→(*F* diplopia) and (*Fg* ?)→(*F* diplopia), where (*Fg* ?) is an unbound finding node denoting a generalization of diplopia. Here, a strategic (or search control) principle like "generalize findings before searching for their causes" would dictate pursuing first the (*Fg* ?)→(*F* diplopia) subgoal.

Step 3: Back in problem search (outer loop), apply a problem-space operator that is capable of instantiating those domain knowledge elements necessary to satisfy the subgoal chosen in the previous step. For example, for the subgoal (*Fg* ?)→(*F* diplopia), a proper operator would search a taxonomy of findings (that is part of the domain knowledge) to bind node (*Fg* ?) to a more general finding that subsumes diplopia (e.g., serious-CNS-finding). Upon updating the SSM with the result of the applied operator, the revised SSM defines the next state in the problem space.

After executing these steps, the updated SSM can still contain unsatisfied subgoals – old ones not yet pursued and new ones that the node(s) newly added to the SSM define relative to ***G***. Hence, the three steps above are repeated until the evolving SSM corresponds to a state in the problem space that satisfies all constraints in ***G***.

2.3. Overview of the SSM-DKM Methodology

If we are to adopt Newell's problem-space paradigm and use the above conceptual reasoning process to create explicit SSMs for a target task, we must first know how model the elements that this reasoning process would utilize for the target task. More specifically, we need to know:

(1) how to elicit and represent knowledge for describing the desired goal state in SSM terms?
(2) how to elicit and represent search operators and the domain knowledge that they use to expand an SSM as a way to generate new states in the problem space? and
(3) how to elicit and represent strategic knowledge for guiding the problem search?

SSM-DKM, the methodology presented in the rest of the paper, provides answers to these questions. This knowledge modeling methodology involves three stages: conceptualization, formalization and validation, and instantiation (see Figure 3). Based on Newell's (1982) knowledge-level concept, the first stage elicits a conceptual knowledge model of a target task. In SSM-DKM, a *conceptual knowledge model* can be informally said to comprise a goal model, ***G***, a domain model, ***K***, and a strategy model, ***S***, all expressed in a way that is free of implementation details. The next stage produces a *formalized knowledge model*, by expressing the conceptual model using first-order predicate calculus and validating it using logic-based techniques. The last stage converts the formalized model into an *operational knowledge model*, by instantiating its elements using three representational constructs. These constructs can be informally defined as follows:

- *G-operators* – goal (or state-difference) operators that inspect an evolving SSM (current state) to identify gaps relative to ***G***.
- *K-operators* – knowledge-producing operators that search the instantiated ***K*** (or populated KB) and activate those elements needed to satisfy subgoals posted in an SSM.
- *S-operators* – strategic (or search-control) operators that use ***S*** to choose the K-operator to apply next.

These constructs are supposed to interact with ACE-SSM – a KBS architecture that creates explicit SSMs using the above discussed reasoning process (Benaroch, 1998).

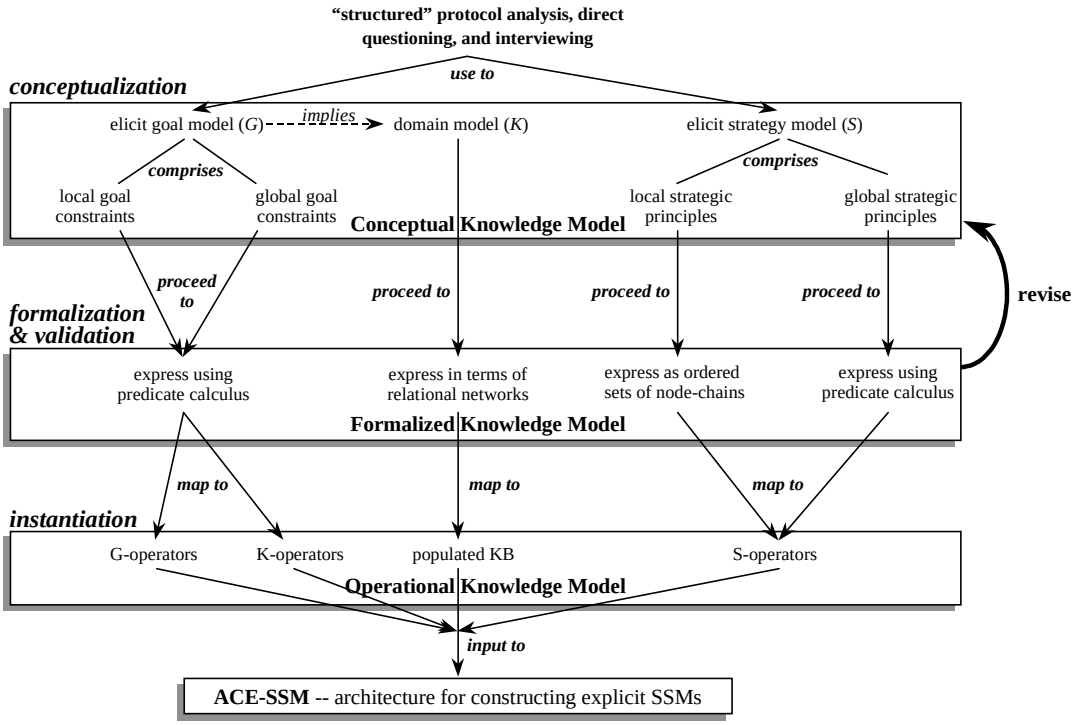


Figure 3: stages in SSM-DKM

Familiarity with ACE-SSM is important for two reasons. We need be able to separate the parts of ACE-SSM from the conceptual knowledge model that SSM-DKM produces in the conceptualization stage, to ensure that this model is implementation-independent. Moreover, to facilitate discussion of the instantiation stage, we need to understand how the representational constructs used to produce an operational model interact with ACE-SSM. We can see how these constructs work with ACE-SSM by reviewing clockwise the two inner circles in Figure 4. When triggered by ACE-SSM's inference procedure, G-operators analyze the SSM and return unsatisfied node-chains (gaps) to be posted in the SSM being created, S-operators analyze the SSM and return a pointer to the unsatisfied node-chain to be pursued next, and K-operators return the pursued node-chain satisfied. Thus, these representational constructs do not directly update the SSM created by ACE-SSM, although some directly analyze it. Only task-independent update operators, which are part of ACE-SSM's inference procedure, update the SSM using results produced by these constructs. Since under the relational networks representation all SSMs are directed graphs, ACE-SSM's update operators apply only general-purpose graph-oriented operations (add node, link nodes, append the roots of disconnected subgraphs, etc.). This means that they are task-independent in the sense that they do not need to know the semantics of SSM node-chains on which they operate, as long as the knowledge model of the task solved is committed to the relational network epistemology.

The next three sections elaborate on the three stages in SSM-DKM, following their order in Figure 3. These sections explain what each stage entails, upon defining more precisely terms we so far used informally.

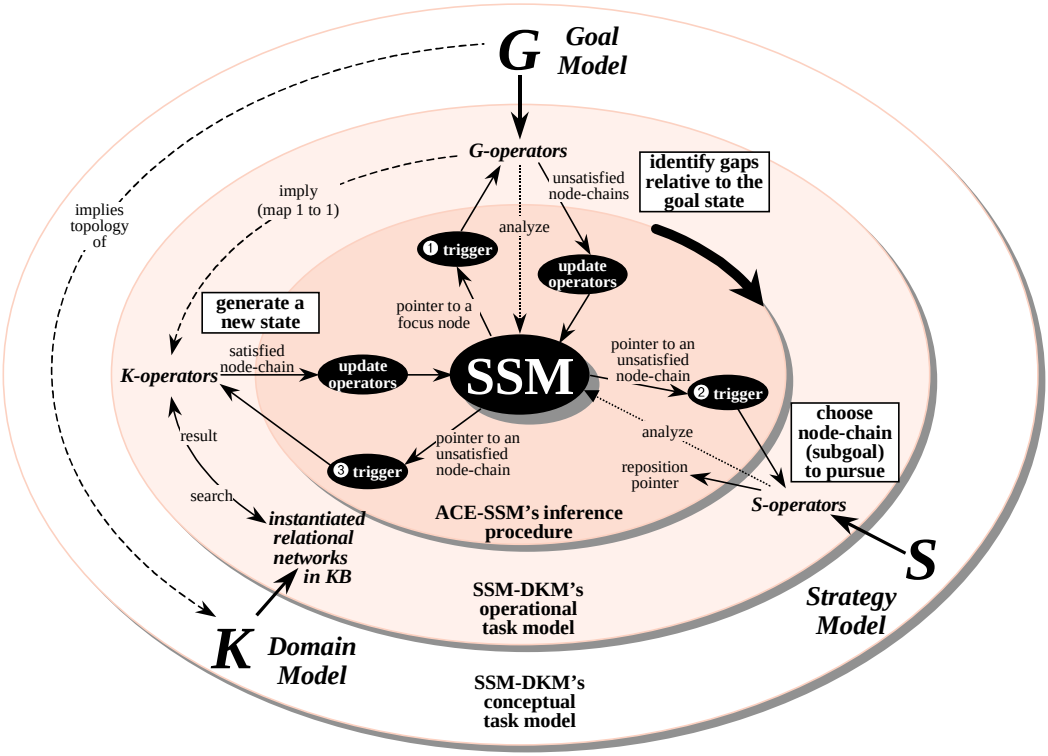


Figure 4: ACE-SSM's operation and interaction with SSM-DKM

3. Conceptualization

This section explains how to construct a conceptual knowledge model of a task. As indicated in Figure 3, such a model comprises three conceptual modeling constructs – *goal model (**G**)*, *domain model (**K**)*, and *strategy model (**S**)*. Our elicitation approach relies on several knowledge acquisition (KA) techniques, including "structured" protocol analysis and direct questioning. To illustrate how the activities involved in conceptualization are carried out using structured protocol analysis, Table 1 provides a sample think-aloud protocol and its gradual analysis during the construction of a conceptual knowledge model for our sample medical diagnosis task.

3.1. Eliciting a *Goal Model* and a *Domain Model*

A goal model, ***G***, describes the underlying structure of SSMs created for a target task. It is a set of goal constraints that must be satisfied by SSMs created for the task. As we show shortly, these goal constraints in essence define the semantics of domain phenomena modeled using SSMs constructed for the task. In eliciting a goal model, we distinguish between two types of goal constraints – *local* and *global* goal constraints.

Local goal constraints define direct relations between domain objects that must hold true in every SSM created for the task. For our sample medical diagnosis task, two such constraints are:

(g1) every SSM node depicting an observed abnormal finding must be causally linked to a disease node that explains it, and

(g2) every SSM node depicting an inferred (non-observed) abnormal finding linked to a disease node must be verified with the patient (asked about).

Local goal constraints can be identified through protocol analysis, interviewing, and direct questioning. These KA techniques help to create an *object-relation matrix* like the one in Table 2. Upon identifying object types (e.g., finding, disease) and their attributes (e.g. hard vs. soft finding), we form a matrix with one row and column for each object type, and then identify types of relations between the objects (e.g., *cause*, *subsume*) and place them in the matrix. In Table 1, the columns labeled pass-1 illustrate how these are identified using protocol analysis.

	Disease	***F***inding	***A***gent	***E***ntry-point	***O***rgan-system	...
Disease	*Cause, subtype*	*cause*				
Finding		*subsume (is-a, definition)*				
Agent	*Induce*		*subtype*	*enter*	*reside in*	
Entry-point						
Organ-system					*subsume*	
...						

Table 2: object-relation matrix

Three observations are in order relative to the object-relation matrix. First, in cases where certain similar relations impact equivalently the problem solving process, they can be replaced by one relation. For example, the *subsume* relation between findings is also used as a synonym of the *is-a* and *definition* relations between findings, because the distinction between these three relations plays no apparent role in problem-solving. Oftentimes such cases can be identified only once search control knowledge has been elicited. Second, developing a sufficiently fine distinction between certain object types could require using psychometric KA techniques like repertory-grid analysis (Boose, 1985). For example, in our sample task, such a need could arise due to the lack of a clear distinction between disease and finding objects. To illustrate, pressure-in-the-brain is a disease that causes diplopia, and at the same time a finding that is caused by brain-mass-lesion (see Figure 1). In this case, such KA techniques could help to reveal that some diseases can be treated as findings relative to deeper causing diseases. Such diseases are commonly referred to pathologic states (or pathologic bodily structures). For brevity, we avoid hereafter this distinction between diseases and pathologic states. Finally, an implicit assumption behind the object-relation matrix is the notion that all relations between domain objects can be represented as binary relations.

An object-relation matrix can be used to draw an Entity-Relationship (ER) diagram like the one in Figure 5. This diagram reveals that, for an application task that is understood sufficiently well, the goal model captures more than just object types and relations. It captures the semantics of domain phenomena with which the object reality of the task is concerned. For our sample diagnosis task, Figure 5 indicates that a disease is modeled as a bodily process that follows a specific causal script. Following the direction of arrows in Figure 5, this process says that: an etiologic agent (e.g., E.coli) enters the body (e.g., virally), moves to some organ system (e.g., meninges-of-the-brain) where it proliferates, inducing a disease (e.g., bacterial-meningitis) that causes specific symptoms (e.g., headache), directly or indirectly through other intermediary diseases (or actually pathologic states) (e.g., pressure-in-the-brain).

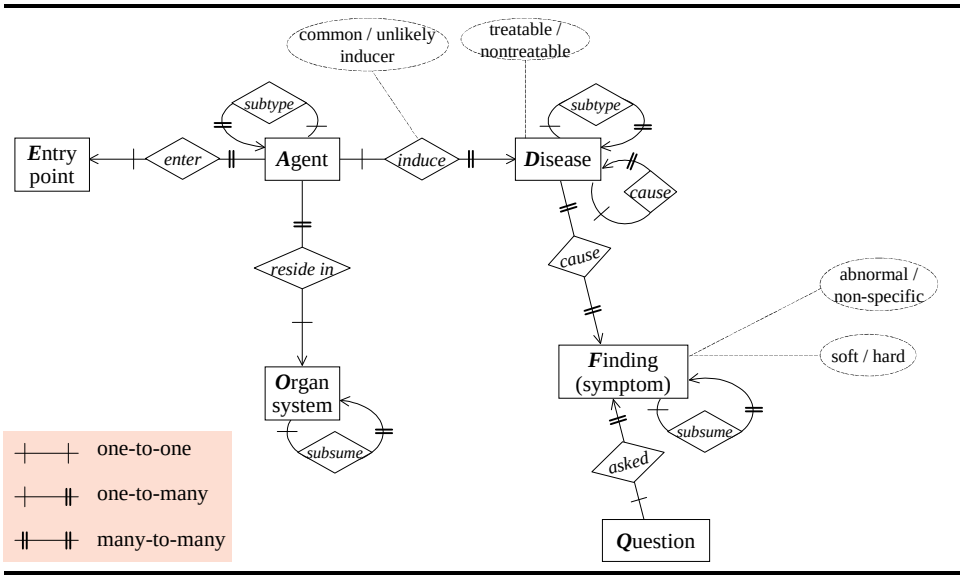


Figure 5: goal model depicted as an entity-relationship diagram

Global goal constraints are elicited relative to the semantics of the domain phenomena defined by local goal constraints. Global constraints define termination conditions that an SSM must satisfy to qualify as the solution sough. Identifying these constraints requires defining the meaning of SSM subgraphs and their roots. Being that an SSM is a directed graph that grows nodes and links during problem solving, it could embody several subgraphs. For example, at some point during its construction, the SSM in Figure 1 includes two subgraphs: one whose root is brain-mass-lesion, another whose root is chronic-meningitis. In our sample task, each SSM subgraph represents a competing (and possibly incomplete) candidate solution. Any such subgraph is the solution sough, if it represents an instance of a causal process that implicates the diagnosed disease depicted by its root. Relative to this notion, termination conditions captured by global goal constraints apply to the root of every candidate SSM subgraph, for example:

(g3) an SSM subgraph depicting the disease process sough must have as a root the most specific disease possible (to allow prescribing the most correct cure), and

(g4) the root of an SSM depicting the disease process sough must cover (explain) every abnormal finding known to be present in the patient.

The roots of SSM subgraphs depicting such competing candidate solutions form what is referred to in conventional diagnosis terms as the *diagnostic differential* – the set of most specific disease hypotheses.

As to the domain model, ***K***, Figure 3 indicate that the goal model, ***G***, implies this model. In the ER diagram notation of a goal model, every binary relation between object types can be directly mapped to a relational network in the domain model. For example, two relational networks identified based on Figure 5 are a specialization taxonomy of diseases and a causal network linking diseases and findings. Thus, a goal model depicted as an ER diagram can be equated with the domain model, because it reflects the topology of domain knowledge. This follows from the fact that a graphic notation of a goal model actually reflects the application ontology – the explicit list and organization of terms that constitute a representational scheme for the reality with which the target task is concerned (Gruber, 1993). For this reason, we shall hereafter refer to the identified object types and relationships as ontological objects and relations.

3.2. Eliciting a *Strategy Model*

Whereas a goal model captures the structure of SSMs in terms of conditions that their nodes (ontological objects) and links (relations) must satisfy, a *strategy model* (***S***) captures strategic principles that determines the order by which a particular SSM subgraph (or candidate solution) gradually grows specific nodes and links. Strategic principles determine which ontological relations to exploit and when, for the purpose of expanding (refining) a candidate solution, confirming a candidate solution, adding candidate solutions to the SSM, honing on a candidate solution, contrasting candidate solutions, etc.

The general control pattern that strategic principles yield is described by the abstract strategy model shown in Figure 6. This model distinguishes between local and global strategic principles, on the premise that an SSM is a directed graph embodying several subgraphs that are each comprised of node-chain instances (linked nodes depicting object instances). Global strategic principles focus the solver's attention to a specific SSM subgraph, or competing candidate solution, whereas local strategic principles determine at every moment the specific node-chain instance to grow next in the SSM subgraph chosen by global principles. Local strategic principles direct problem-solving most of the time, as the SSM subgraph in focus continues to grow one node-chain instance at a time. In contrast, global strategic principles take affect only once in a while, conditional on the occurrence of special changes to the SSM that require the solver to divert attention from one SSM subgraph to another. To define more precisely the strategic principles comprising a strategy model, we next look at local and global principles separately.

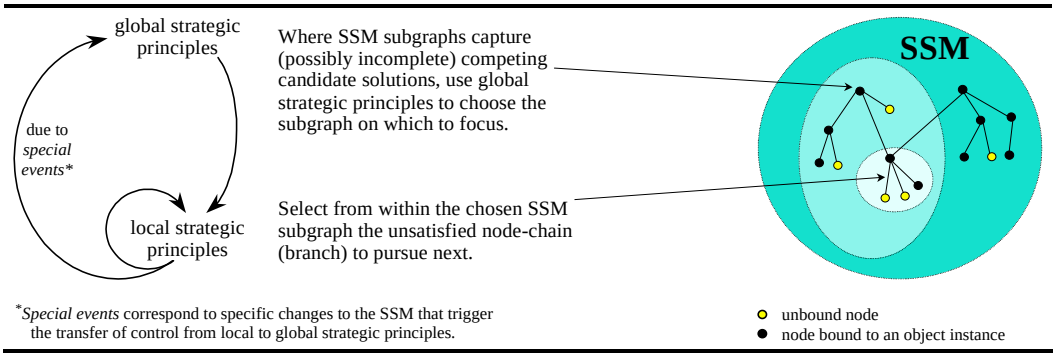


Figure 6: abstract strategy model showing the pattern of control yielded by strategic principles

Local strategic principles determine the order by which the SSM grows specific node-chain instances through "spreading activation" of various knowledge elements in the instantiated domain model (or populated KB). More specifically, given that at any moment during problem solving the solver focuses on one specific bound node in the SSM (e.g., diplopia), termed the *focus node*, local strategic principles determine the order by which to pursue the types of unsatisfied node-chains (subgoals) that can be associated with this node. For example, take the case of a focus node depicting an instance of a disease object. Based on Figure 5, the ER diagram notation of a goal model, Figure 7a shows that such a focus node can be associated with four types of unsatisfied node-chains that map it to:

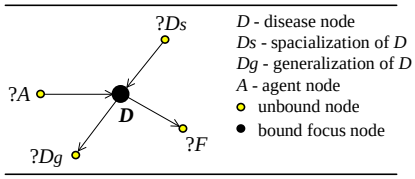
(1) a more general disease of which it is a subtype, denoted ?*Dg*→*D* (e.g., ?infection→meningitis);

(2) more specific diseases that are its subtypes, denoted *D*→?*Ds* (e.g., meningitis→?bacterial-meningitis);

(3) findings it causes, denoted *D*→?*F* (e.g., meningitis→?headache-duration); and

(4) its inducing agents, denoted ?*A*→*D* (e.g., ?E.coli→bacterial-meningitis).

Respectively, a local strategic principle like "test a disease hypothesis before refining it" would dictate pursuing the type of node-chain numbered (3) before the one numbered (2). Apparent from this discussion is the notion that different subsets of local strategic principles apply relative to the types of node-chains associated with each ontological object separately. Consequently, we also refer to these principles as *local object-centered principles*.



(a) different ontological node-chains associated with a focus node depicting a disease object

	?*Dg*→*D*	*D*→?*Ds*	*D*→?*F*	?*A*→*D*
?*Dg*→*D*		≻	≺	≻
D→?*Ds*			≺	≻
D→?*F*				≻
?*A*→*D*				

(b) node-chain matrix showing object-centered principles as precedence relations (≺/≻) on the order of pursuing the different ontological node-chains associated with a disease focus node

Figure 7: eliciting local object-centered strategic principles

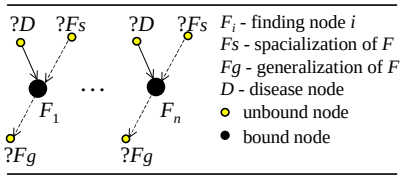
To elicit local object-centered strategic principles, we create for each ontological object a *node-chain matrix* like the one in Figure 7b. We first use the graphical ER model in Figure 5 to identify the types of node-chains (subgoals) associated with each ontological object. Then, we identify the order by which these types of node-chains are pursued during problem solving, using one of the following KA techniques.

- **Protocol analysis**: protocols can be recoded in terms of the order by which competing node-chains are pursued. We can illustrate how this is done using the columns labeled pass-2 in the sample protocol in Table 1. For each step in the reasoning process: (1) identify the focus node in the left column, (2) identify the types of unsatisfied node-chains associated with the focus node in the next column, (3) identify the type of node-chain that the solver pursued in the next column, and (4) identify the implied order of pursuing unsatisfied node-chains associated with the focus node in the right column. These four steps yield enough information to fill a node-chain matrix like the one in Figure 7b. For example, in line 2, the expert generalized a findings before mapping it to its causes, implying the order relation: pursue ?*Fg*→*F* before ?*D*→*F*, where *Fg* denotes a generalization of *F*.
- **Interviewing**: the expert might volunteer abstract strategic principles like "test a disease hypothesis before refining it", which implies the order relation: pursue *D*→?*F* before *D*→?*Ds*, where *Ds* is a specialization of *D* and ? denotes the unbound node in an unsatisfied node-chain.
- **Direct questioning**: the expert can be asked to answer direct questions like: given a disease hypothesis, do you first map it to findings it causes or to the agent(s) inducing it?

There is another type of local strategic principles called *local attribute-centered principles*. These are needed when the modality of certain types of node-chains (implied by relationship in the ER goal model) is not 1-to-1. For example, consider the relationship between diseases and findings. Figure 8a shows a case where, given a set of known findings $\{F_i\}$, and assuming that object-centered principles dictate pursuing next the type of node-chain denoted ?*D*→*F* (before *F*→?*Fs* and ?*Fg*→*F*), the question is which finding in $\{F_i\}$ to map first to a causing disease. Here, attributes of findings in $\{F_i\}$ help to discriminate between the candidate node-chains considered, $\{?D \rightarrow F_i\}$. In this case, the answer comes in the form of an attribute-centered principle like: "always pursue abnormal findings first." Line 2 in the sample protocol in Table 1 shows one case where this principle is used during problem solving.

To elicit attribute-centered principles, we form for each ontological object with attributes an *object-attribute matrix* like the one in Figure 8b. This matrix has one row and column for each attribute value. We fill it using the same KA techniques used for object-centered principles. To

illustrate, upon inspecting the columns labeled pass-2 in the protocol in Table 1, it appears that in line 2 the expert chose to first pursue diplopia before all other given findings because diplopia is an abnormal finding. This example implies the order relation: pursue *F*[abnormal] before *F*[other], where *F*[ ] denotes an attribute value that an instance *F* node can take on.



(a) similar set of ontological node-chains associated with a focus node depicting a finding object

	$F_{[soft]}$	$F_{[hard]}$	$F_{[abnormal]}$	$F_{[other]}$
$F_{[soft]}$		$\succ$	$\prec$	$\succ$
$F_{[hard]}$			$\prec$	$\succ$
$F_{[abnormal]}$				$\succ$
$F_{[other]}$				

(b) object-attribute matrix showing attribute-centered principles as precedence relations ($\prec$/$\succ$) on the order of pursuing similar ontological node-chains whose focus node is a finding object having one of the above attributes

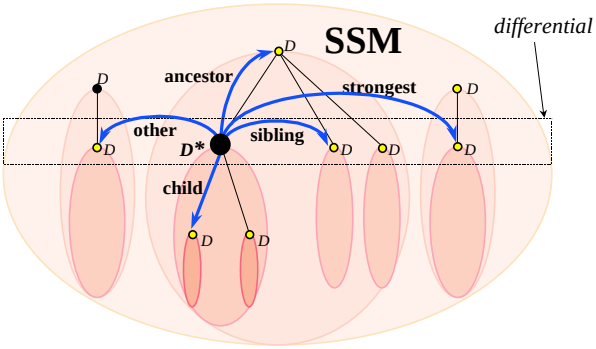
Figure 8: eliciting local attribute-centered strategic principles

Before we can explain how to elicit global strategic principles, we need to define them more precisely. As an SSM evolves, it could embody several subgraphs that depict different candidate solutions. For example, as seen in Figure 1 (and lines 2-6 in Table 1), the diplopia finding is generalized and then mapped to two plausible causing diseases – brain-mass-lesion and chronic-meningitis. These plausible causing diseases become the roots of different SSM subgraphs depicting two competing candidate solutions (or incomplete disease process descriptions). Assuming that during problem solving the solver focuses on one candidate solution at a time, *global strategic principles* tell the solver when to divert attention from the currently pursued candidate solution to another candidate. These principles are also called *subgraph-centered principles* because they determine which SSM subgraph the solver has to continue growing.

Eliciting global strategic principles first requires understanding the topology of SSMs relative to the application task. Understanding the topology of SSMs requires revealing (relative to the task): what an SSM subgraph stands for, how many such subgraphs could an SSM embody, what does the root of such a subgraph represents, how do the roots of SSM subgraphs relate to each other, etc. This information can be derived from the problem solving termination conditions captured by global goal constraints and from the domain model.

In our sample task, the topology of an SSM and the meaning of SSM subgraphs are determined by two things: the notion that a solution to any given problem situation is a disease process, and the relationships in ***K*** which hold among those ontological objects that are the roots of SSM subgraphs. More specifically, for our sample task, the roots of all SSM subgraphs (and their subgraphs) are disease nodes that are organized in the SSM following the topology of the portion of the disease taxonomy in the instantiated domain model that was thus far triggered by spreading activation. Of these, the most specific disease nodes form the diagnostic differential. Figure 9 shows the relationships that can exist between the roots of competing SSM subgraphs, relative to the topology of the disease taxonomy and to their membership in the differential.

The next step in eliciting global strategic principles is to identify "special events" that trigger the need to divert attention from the currently pursued SSM subgraph to another SSM subgraph. Recall that global strategic principles take affect conditional on specific changes to the SSM, whose occurrence is checked for after each time the SSM grows a new node-chain instance. These specific changes to the SSM could indicate, for example, that the strength of (or degree of belief in) the currently pursued SSM subgraph dropped below that of another subgraph, or that the the currently pursued subgraph no longer can be expanded.



Where *D** is the current solution focus, or root of the currently pursued SSM subgraph, global strategic principles can divert the solver's attention to another SSM subgraph whose root is the:

- "strongest" candidate solution in the differential
- child of *D**
- sibling of *D**
- some other candidate in the differential
- an ancestor of candidates in the differential

Figure 9: relations between candidate solutions as the basis for eliciting global subgraph-centered strategic principles

In our sample medical diagnosis task, all special events pertain to changes in the differential. More specifically, if the root of the currently pursued subgraph is termed the *solution focus*, denoted *D**, the following are special events that can result in changing this focus node:

(s1) *Change in strength*: where the strength of *D** is the degree of belief (e.g., certainty factor (CF)) in its subgraph being the solution sough, the strength of *D** dropped below that of other candidates in the differential, requiring to make the strongest candidate in the differential the solution focus. (Note that "strength" can be measured in other ways, for example, by the number of abnormal findings covered by the SSM subgraph *D** spans. See [Benjamins and Jansweijer, 1994] for details.)

(s2) *New candidate solutions*: a new candidate was added to the differential, either because its strength increased above some threshold (e.g., CF>0.2) or as a result of mapping some known finding to its causing diseases. In the latter case, the newly added disease node is not subsumed by any SSM subgraph, indicating that it is from a disease category not yet considered. In either case, it is necessary to check whether *D** is still the strongest candidate in the differential.

(s3) *Finer candidate solutions*: *D** was just specialized into more specific disease nodes, and this requires replacing it in the differential by its specializations and making one of the children the solution focus. This has the affect of focusing the solver on a more specific plausible disease within the last disease category considered (or narrowing down the current diagnostic context).

(s4) *Pursued solution focus*: an attempt to pursue (test and refine) *D** was completed, and so a sibling of *D** becomes the focus. This ensures that the solver remains focused on the last disease category (or diagnostic context) considered.

(s5) *Pursued siblings*: *D** and all its siblings have been pursued, thus requiring to switch focus to another candidate in the differential that has not yet been pursued.

(s6) *Pursued differential*: *D** and all candidates in the differential have been pursued, indicating the possible need to switch to categorical reasoning involving a broader disease category. Here, the solution focus could become, for example, an ancestor of candidates in the differential.

The above special events fall into three general groups. Event (s1) corresponds to changes in strength of candidates in the differential, events (s2)-(s3) correspond to changes in the composition of the differential, and events (s4)-(s6) correspond to changes in the status of candidates in the differential (from not yet pursued to pursued). These special events effect search in two general ways. Some narrow down the search within the currently pursued SSM subgraph, e.g., event (s3).

Others broaden the search to unexplored SSM subgraphs, e.g., event (s2). The effect of special events (s1) and (s3) is apparent in the protocol in Table 1, in lines 8 and 15-16, respectively.

As Clancey (1988) notes, strategic principles can be grounded in various rationale constraints. For example, depending on the width and depth of the disease taxonomy in the instantiated domain model, local strategic principles can be designed to induce a depth-first or a breadth-first search of this taxonomy, to make the problem solving process more efficient. This same idea applies to global strategic principles as well. To illustrate, due to a rationale constraint that entails minimizing the mental effort required in switching focus from one SSM subgraph to another, we may not want to stop pursuing the current solution focus every time it becomes "weaker" than other candidates in the differential, and instead choose to do so only when its strength drops below some threshold. This discussion indicates that some strategic principles do not necessarily have to be purely based on the reasoning process of an expert source, and instead be normative in some sense.

Once the strategic principles comprising a strategy model have been elicited, we can combine all the matrices used to assist in this endeavor with the identified special events. The result is an integrated matrix like the one in Table 3. This integrated matrix provides no additional information per-se. However, we choose to use it here to show that the structure of the elicited strategy model – or the elicited strategic principles and their organization – follows the structure of the abstract strategy model depicted in Figure 6.

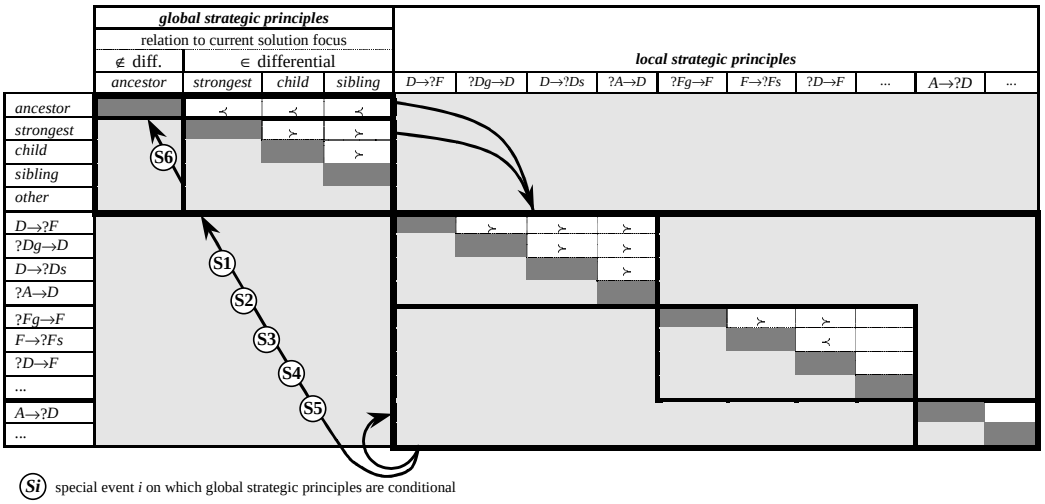


Table 3: integrated matrix showing that the elicited strategy model reflects the same abstract control pattern in Figure 6

4. Formalization and Validation

Having seen how a conceptual knowledge model is elicited, the next stage is to formalize and validate this model. This section explains the formalization approach SSM-DKM uses to produce a structure-preserving formalized model. A key benefit from a structure-preserving model is its ease of revision; recall from Figure 3 that the conceptual knowledge model may have to be revised upon its formalization and validation. Additionally, since much of the formalized knowledge model SSM-DKM produces is expressed in predicate calculus, this section also discusses the extent to which validation of this model can be taken using logic-based techniques.

4.1. Formalizing the Goal and Domain Models

The goal model, ***G***, is formalized by expressing its comprising goal constraints using first-order predicate calculus. Every goal constraint is written as a quantified logic sentence of the form (Thayse, 1988):

$$\textit{quantifier } O_1 \ldots O_n \; [\neg]\{\pi(O_j,O_k), \alpha(O_i)\} \; [[\wedge,\vee,[\neg]]\{\pi(\ldots), \alpha(\ldots)\}]$$
$$\Rightarrow \textit{quantifier } O_1 \ldots O_n \; [\neg]\{\pi(O_j,O_k), \alpha(O_i)\}[[\wedge,\vee,[\neg]]\{\pi(\ldots), \alpha(\ldots)\}].$$

In this sentence and hereafter, the following notation is used: O_i is an ontological object, π (italic) is a binary function constant, α (non-italic) is a unary function constant, and [ ] is an optional subsentence. Intuitively, these notation correspond to the ER diagram notation of a domain model: O_i is an entity set (e.g., disease, finding), π (italic) is a relationship between two entity sets (e.g., *cause*, *subsume*), and α (non-italic) is an attribute of an entity set (e.g., abnormal finding). For example, the goal constraints labeled (g1)-(g2) in Section 3.1 are expressed as:

(g1') $\forall F \;\; \text{abnormal}(F) \wedge \text{observed}(F) \Rightarrow \exists D \;\; \textit{cause}(D,F)$

(g2') $\forall F \;\; \text{abnormal}(F) \wedge \neg\text{observed}(F) \wedge (\exists D \; \textit{cause}(D,F)) \Rightarrow \exists Q \;\; \text{asked}(Q,F)$

where F is a finding object, D is a disease object, Q is a question asked about F, 'abnormal' and 'observed' are attributes of F, and *cause* and *asked* are binary relationships.

In principle, for every ontological object in the ER diagram notation of a goal model, there should be one local goal constraint for each relationship linking it with another object in this ER diagram. The logic behind this observation is simple: every local goal constraint inspects properties of a single ontological object that is the focus node in the SSM and returns exactly one unsatisfied node-chain (subgoal) associated with this object.

By contrast, when the conceptualization stage identifies several global goal constraints, these can be generally integrated into one constraint. All global goal constraints usually pertain to conditions that must be met by a key object of interest, e.g., the disease node that is the root of an SSM subgraph depicting the diagnosed disease process. While these constraints may have different antecedents, they all have an identical consequent that identifies the same object as the solution sough. Consequently, all global goal constraints identified during conceptualization could be integrated into one predicate calculus sentence whose antecedent is a conjunction of their antecedents:

$$\left.\begin{array}{l} \text{antcendent}_1 \Rightarrow \text{consequent}(D) \\ \quad\quad \ldots \\ \text{antcendent}_n \Rightarrow \text{consequent}(D) \end{array}\right\} \text{antcendent}_1 \wedge \cdots \wedge \text{antcendent}_n \Rightarrow \text{consequent}(D).$$

For example, the next formalized global goal constraint combines the conceptual global constraints labeled (g3) and (g4) in Section 3.1:

(g3') $\forall D \;\; (\neg\exists F \;\; \neg \textit{cause}(D,F) \wedge \text{abnormal}(F)) \wedge (\neg\exists D_1 \;\; \neg\exists F \;\; \textit{subtype}(D_1,D) \wedge \neg \textit{cause}(D_1, F)) \Rightarrow \text{solution}(D)$

where F is a finding object, D and D_1 are disease objects, 'abnormal' is an attribute of F, 'solution' is a relational constant identifying its argument as the disease sough, and *cause* and *subtype* are binary relationships.

There are two important observations to be made about formalized goal constraints. First, they all involve functions that access only the SSM, not the instantiated domain model (or populated KB). Second, they can be expressed purely in terms pertaining to the ontological identity of objects and relations corresponding to SSM node-chains, or also using graph-oriented terms pertaining to the topology of SSM. For example, a conceptual term like $\textit{subtype}(D_1,D)$ in the goal constraint labeled (g3') can be replaced by the graph-oriented term $\textit{child}(D_1,D)$.

As to the formalized domain model, ***K***, all its elements are implicit in the formalized goal model, ***G***. We saw earlier that the ER diagram notation of a goal model identifies specific

relations between ontological objects in the domain model, e.g., *cause*(*D*,*F*) and *subtype*(D_1,D_2). But, these need to be further formalized. When the instantiated domain model contains relational networks, each network would generally comprise sentences of the form $\pi(i_j,i_k)$, where π is a binary relation and *i* is an object instance, e.g., *cause*(meningitis, headache). However, we represent each such sentence as $\pi((O\ i\ [a]), (O\ i\ [a]))$, where the tuple (*O i* [*a*]) denotes a node in which *O* is an ontological object type, *i* is an instance of *O* (or a value to which *O* is bound), and [*a*] denotes one or more optional attribute-values of *i*, e.g., *cause*((*D* bacterial-meningitis),(*F* high-grade-fever abnormal)). Under our notation, we say that every sentence coincides with a specific *node-chain* instance, e.g., (*D* bacterial-meningitis)→(*F* high-grade-fever) for the last example. Respectively, each relational network in the formalized domain model has a signature of the general form:

$$\pi((O\ [a])\rightarrow(O\ [a])\ [p]),$$

where π is a relational constant, (*O* [*a*])→(*O* [*a*]) is an ontological node-chain in which [*a*] denotes one or more optional object attributes, and [*p*] is an optional a-priori probability (e.g., certainty factor) measuring the strength of the association each node-chain instance captures.

4.2. Formalizing the Strategy Model

Being that a strategy model, ***S***, comprises both local and global strategic principles (see Figure 7), we start with the formalization of local principles. We saw that unsatisfied subgoals are expressed as node-chains with one unbound node. Recalling that the bound node in the currently pursued node-chain (subgoal) is termed the *focus node*, we also saw that local object-centered strategic principles are basically declarative preferential constraints on the order of pursuing the types of node-chains associated with a focus node. These preferential constraints are expressed formally as ordered sets of node-chains. For example, upon rearranging rows and columns in the node-chain matrix in Figure 7b so that all cells above the main diagonal are filled with the precedence relation "≻", the principles that this matrix reflects relative to a disease focus node form a total order which is written formally as:

$$\{?\rightarrow D\rightarrow?:\ D\rightarrow?F\ \succ\ ?Dg\rightarrow D\ \succ\ D\rightarrow?Ds\ \succ\ ?A\rightarrow D\},$$

where ? denotes the unbound node in an unsatisfied node-chain. Local attribute-centered principles are expressed similarly. For example, upon rearranging the matrix in Figure 8b, the local attribute-centered principles it reflects relative to a finding focus node are written formally as:

$$\{F:\ F_{[\text{abnormal}]}\ \succ\ F_{[\text{soft}]}\ \succ\ F_{[\text{hard}]}\ \succ\ F_{[\text{other}]}\},$$

where (for convenience to the reader) $F_{[\ ]}$ denotes an attribute value that a finding object can take on.

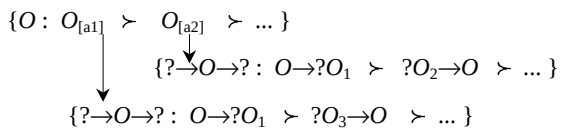
Our earlier discussion assumed that, given a focus node, object-centered principles determine which of the types of unsatisfied node-chains associated with this node to pursue next, and attribute-centered principles act as "tie breakers" when this node is associated with several node-chains of the same type. More generally, object-centered principles can be contingent on the attribute value of the focus node. For example, if the focus node is an observed (hard) finding, generalize it before mapping it to causing diseases; if the focus node is an abnormal finding, map it to causing diseases before generalizing it; etc. In the same spirit, attribute-centered principles can be contingent on a specific type of node-chain. Consequently, where *O* is a specific ontological object type and $O_{[ai]}$ is the *i*-th attribute value that *O* can take on, local strategic principles pertaining to this object can be combined into a nested list of ordered sets having one of the two general forms:

$$\{?\rightarrow O\rightarrow?:\ O\rightarrow?O_1\ \succ\ ?O_2\rightarrow O\ \succ\ \ldots\}$$

$$\{O:\ O_{[a2]}\ \succ\ O_{[a1]}\ \succ\ \ldots\}$$

$$\{O:\ O_{[a1]}\ \succ\ O_{[a3]}\ \succ\ \ldots\}$$

or



Unlike local strategic principles, global principles are expressed using predicate calculus because they are conditional on the occurrence of special events. As Figure 7 shows, relative to the current solution focus (or root of the currently pursued SSM subgraph), candidate solutions bear graph-oriented relations pertaining to the topology of SSMs (e.g., child, sibling, ancestor). For this reason, formalized global strategic principles are expressed using graph-oriented terms to the extent possible. For example, where is-focus(D) identifies its argument as the current solution focus, the global strategic principles labeled (s1), (s3) and (s4) in Section 3.2 are written as:

(s1') $\forall D$ is-focus(D) $\wedge$ ($\exists D_1$ in-differential(D_1) $\wedge$ strongest-belief(D_1) $\wedge$ ($D \neq D_1$)) $\Rightarrow$ is-focus(D_1),

(s3') $\forall D$ is-focus(D) $\wedge$ ($\exists D_1$ child(D_1,D) $\wedge$ $\neg$pursued(D_1)) $\Rightarrow$ is-focus(D_1),

(s4') $\forall D$ is-focus(D) $\wedge$ pursued(D) $\wedge$ ($\exists D_1$ sibling(D_1,D) $\wedge$ $\neg$pursued(D_1)) $\Rightarrow$ is-focus(D_1).

In the above formalized principles, several support functions are themselves written as sentences in predicate calculus. For example:

$\forall D$ belief(D)>0.2 $\wedge$ ($\neg\exists D_1$ child(D_1,D)) $\Rightarrow$ in-differential(D),

$\forall D$ $\neg\exists D_1$ ($D \neq D_1$) $\wedge$ in-differential(D) $\wedge$ in-differential(D_1) $\wedge$ belief(D)<belief(D_1) $\Rightarrow$ strongest-belief(D),

$\forall D \exists F$ *cause*(D,F) $\wedge$ is-bound(F) $\wedge$ ($\exists D_1$ *subtype*(D,D_1) $\wedge$ is-bound(D_1)) $\Rightarrow$ pursued(D).

The last example involves two functions worth explaining. The 'pursued' function checks if there was an attempt to both test a disease node (by mapping it to findings it causes) and specialize it (by mapping it to its subtypes). And, 'is-bound' is a function that checks whether its argument is bound in the SSM to an object instance, where bound means that the argument was visited and filled with a value from the instantiated domain model or with a user answer to a question. Like with goal constraints, all functions used to express formalized global strategic principles access only the SSM, not the instantiated domain model.

It is worth mentioning at this point an implementation-related issue. It is important to realize that, when the antecedents of global strategic principles are not mutually exclusive, the order by which these principles are applied during problem solving (or their physical order in the system) will necessarily compile control knowledge. For example, principle (s1') will always fire before principle (s4') above, unless the sibling of the current solution focus is also the strongest candidate in the differential. If we are to ensure that all search control knowledge is properly expressed in declarative form, this issue requires some attention probably during validation. An alternative that would solve this problem (but render the declarative principles longer) is to make the principles mutually exclusive by including in very one of them the negation of all other principles.

4.3. Validation

Validation[1] provides feedback regarding anomalies – inconsistencies, incompleteness, and redundancies – in the formalized knowledge model that can hinder the adequacy of the (implemented) operational knowledge model. Like recent work on validation (e.g., [Vermasen and Wergeland, 1994; Van Harmelen and Aben, 1996]), SSM-DKM focuses on validation in the context of abstract application-independent specification languages. Separating functional

[1]In our view, *validation* entails establishing the completeness, consistency and soundness of knowledge in a conceptual model, whereas *verification* entails checking that the formalized knowledge model is correctly mapped to an operational model (or system design).

implementation-independent properties of a knowledge model from implementation-specific ones facilitates using powerful validation techniques involving higher level mathematical abstractions. Additionally, it permits conducting validation early in the life-cycle to avoid costly implementation efforts.

In SSM-DKM, much of a formalized knowledge model is expressed using predicate calculus, providing for the use of powerful logic-based validation techniques discussed by Genesereth and Nilsson (1987) and by Thayse (1988). Since validation is one of several areas where elaborate discussion of our work can ensue, we only illustrate here the key benefits that a formalized knowledge model produced using SSM-DKM can offer in validation.

Because in SSM-DKM the goal model, ***G***, drives much of the knowledge modeling endeavor, we start with several useful observations regarding this model. A goal model captures information that a solver has about the domain phenomena with which the target task is concerned. This information, expressed as goal constraints, constitutes the solver's base set of beliefs (or truths) about the nature of these phenomena. In the formalized goal model, these beliefs are expressed as a finite set Γ_P of first-order predicate formulas. In effect, each formalized goal constraint (or formula) $\gamma \in \Gamma_P$ can be equated with an axiom that holds in the application domain, and the formalized goal constraints in Γ_P constitute the proper axioms (as opposed to logical axioms) of a theory of the task. Respectively, Γ_P is said to define an axiomatic system relative to this task, and the theory $\tau[\Gamma_P]$ is the closure of Γ_P under logical implication (Genesereth and Nilsson, 1987). This axiomatic system is obtained by adding the proper axioms of the theory to the sound and complete axiomatic system of predicate calculus. With this in mind, validation of the formalized goal model boils down to establishing the properties that the axiomatic system Γ_P must satisfy: consistency, compactness, soundness, and completeness. We illustrate next how SSM-DKM accordingly deals with the issues of consistency and compactness.

To check for the consistency of the goal model, the set of formalized goal constraints Γ_P is converted into clausal form, denoted Γ_C. For instance, the constraint ($\forall F$ abnormal(F) $\wedge$ observed(F) $\Rightarrow \exists D$ *cause*(D,F)) in predicate calculus form is converted into the set of clauses {$\neg$abnormal(F), $\neg$observed(F), *cause*($\phi(F)$, F)}, where $\phi()$ is a skolem function. Then, Γ_C can help to identify inconsistencies, by using the resolution algorithm to try to derive the empty clause {}. This is based on the *ground compactness theorem*: If a set Δ of ground clauses is unsatisfiable (inconsistent), then there is a resolution deduction of the empty clause from Δ (Genesereth and Nilsson, 1987).

To check for the compactness of Γ_C, the formalized goal model in clausal form, we use a different approach. For each clause $\gamma_i \in \Gamma_C$, remove γ_i from Γ_C and then try to derive γ_i from γ_1,..., γ_{i-1}, γ_{i+1},..., γ_n. That is, we try to find if γ_i is a sentence that can be proved from the set of sentences γ_1,..., γ_{i-1}, γ_{i+1},..., γ_n, i.e., $\Gamma_C - \{\gamma_i\} \vdash \gamma_i$. Respectively, for each clause $\gamma_i \in \Gamma_C$, let $\Gamma_C' = \Gamma_C - \{\gamma_i\}$, $\Gamma_C'' = \Gamma_C' \cup \{\neg\gamma_i\}$, and then try to derive the empty clause {} from Γ_C'' to prove that γ_i is not redundant. This approach relies on the *completeness theorem*: A theory $\tau[\Delta]$ – the set of all sentences available from a database of sentences Δ that is closed under logical implication – is complete if and only if, for every sentence ϕ, either $\phi \in \tau[\Delta]$ or $\neg\phi \in \tau[\Delta]$ (Genesereth and Nilsson, 1987).

As to validating the strategy model, ***S***, because this model captures strategic principles in a declarative form, SSM-DKM allows doing much more than knowledge modeling approaches that capture such knowledge procedurally. Relative to local strategic principles, for every node-chain matrix elicited for a single ontological object (e.g., Figure 7b), if it has all the node-chains associated with that object and it can be rearranged such that only cells above the main diagonal have the "$\succ$" precedence relation, the local principles that this matrix reflects are guaranteed to be complete, consistent, and compact. Both inconsistency and incompleteness of these local

principles result in the ability to derive only a partial order over unsatisfied node-chains, e.g.,

$$\{?\rightarrow D\rightarrow?: \ D\rightarrow?F \ \ \succ \ \{?Dg\rightarrow D \ \ D\rightarrow?Ds\} \ \succ \ ?A\rightarrow D\}.$$

This mean that the choice among partially ordered node-chains (e.g., pursuing $?Dg\rightarrow D$ before $D\rightarrow?Ds$ or vice versa) could be arbitrary. Generally, in such cases, inconsistency and incompleteness pose one problem. The less order local strategic principles impose on subgoals posted in the SSM, the more likely is the solver to end up doing an unguided search of the problem space.

As to global strategic principles, these can be validated just like the goal constraints comprising a goal model. Upon formalizing global strategic principles to produce the set of first-order formulas Σ_P and then transforming Σ_P into clausal form Σ_C, we can use Σ_C to check for their consistency. However, this does not eliminate the need to verify that the antecedents of global strategic principles are mutually exclusive. For this purpose, we can use conventional table-based approaches used to validate rule bases (Stachowitz, 1990; Cragun and Steudel, 1987).

There are certain validation issues that SSM-DKM cannot yet address. For example, establishing the completeness of a goal model and a strategy model is more difficult. Likewise, when an inconsistency of goal constraints or global strategic principles is identified, how to pinpoint sources of the inconsistency and suggest ways to resolve it. Nevertheless, while these and several other validation issues remain to be worked out, SSM-DKM opens the possibility of using a host of powerful logic-based validation techniques that are not accessible to most other knowledge modeling approaches (more about it in Section 6).

5. Instantiation

Once the formalized knowledge model is validated, the next stage is to instantiate it. In SSM-DKM, instantiation means populating the KB and mapping the formalized model to representational constructs that can interact with the ACE-SSM architecture. As shown in Figure 4, these constructs include G-operators, K-operators, and S-operators.

5.1. Instantiating the Goal Model

Local goal constraints in the formalized goal model are mapped to G-operators and K-operators. *G-operators*, also called *state-difference operators*, are rules that inspect the SSM and apply goal constraints. A G-operator returns the consequent of a formalized goal constraint as an unsatisfied node-chain, provided that the antecedent of the constraint is violated. For example, for the goal constraint ($\forall F$ abnormal(F) $\wedge$ observed(F) $\Rightarrow \exists D$ *cause*(D,F)), a suitable G-operator is written in pseudo-code as follows:

```
(G_OP::DISEASE_CAUSE_FINDING(F $var1)                    % argument: bound Finding focus node
    if (F $var1) is abnormal and                         % if constraint on F is violated
       (F $var1) is observed and
       (F $var1) is not linked to a disease node
    then return((D  ?$var2)→(F $var1))).                 % return an unsatisfied node-chain
```

The IF part of this G-operator takes as argument the SSM finding node bound to **$var1**, and checks whether it is an abnormal observed finding and whether there was no previous attempt to map it to its causing diseases. When the IF part is true, the THEN part returns the unsatisfied node-chain **(D ?$var2)→(F $var1)**. ACE-SSM's inference procedure uses this node-chain to add to the SSM a link between **(F $var1)** and the unbound node **(D ?$var2)**.

K-operators, also termed *knowledge-producing operators*, use the instantiated domain model (or KB) to derive a value to which to bind the unbound node in unsatisfied node-chains. Before we proceed, recall that the instantiated domain model must include the ontological identity of object instances it captures. For example, in a KB comprising explicit relational networks, a disease taxonomy would look as follows:

```
((subtype (D)→(D))
          ((D infectious-disease)→(D meningitis))
          ((D meningitis)→(D acute-meningitis))
          ((D meningitis)→(D chronic-meningitis))
          (...)),
```

and a network that distinguishes between agents that are common or unlikely inducers of diseases would look as:

```
((induce (A <common-inducer unlikely-inducer>)→(D))
         ((A enterobacteriaceae common-inducer)→(D pelvis-abscess))
         ((A gram-positive-rods unlikely-inducer)→(D pelvic-abscess))
         (...)).
```

There is a strict one-to-one mapping from G-operators to K-operators, as Figure 4 indicates. For example, a K-operator for handling the unsatisfied node-chain **(*D* ?$var2)→(*F* $var1)** produced by the above G-operator is:

```
(K_OP::LINK_F_TO_CAUSING_D((D ?$var2)→(F $var1))      % argument: node-chain with unbound node
                                                      % body: code that instantiates elements
             < ... body ... >                         %    in K to bind $var2 to the disease(s)
                                                      %    causing the finding bound to $var1
      return((D  $var2)→(F $var1))).                  % return a satisfied node-chain
```

Thus, a K-operator receives as argument a specific type of unsatisfied node-chain it can satisfy, it searches the instantiated domain model (or KB), and then returns the node-chain argument with the unbound node bound to proper instances. The body of a K-operator can take different forms, depending on how the instantiated domain model is physically coded. Where the instantiated domain model comprises explicit relational networks, the body could use graph-oriented operations to search these networks (e.g., get child, find ancestor). If instead the instantiated domain model comprised of a set of relational database tables, for example, where column names correspond to the ontological identity of objects, the body of a K-operator could activate an SQL query on these database tables.

5.2. Instantiating the Control Model

S-operators, also referred to as *strategic operators*, are the representational constructs used to capture strategic principles on the symbol-level. Preferential constraints capturing local principles as ordered sets of node-chains are applied by a simple general-purpose S-operator, which in pseudo-code takes the form:

```
(S_OP::choose_next_node_chain($p)                  % $p = pointer to current focus node in the SSM
    N = set of unsatisfied node-chains associated with $p and posted in the SSM
    apply object-centered principles to N to choose the type of node-chain to pursue next
    A = set of similar unsatisfied node-chain instances of the chosen type
    apply attribute-centered principles to A to choose the node-chain instance to pursue
    return($p = chosen unsatisfied node-chain))   % reposition pointer to chosen node-chain
```

As to formalized global strategic principles, these are instantiated using S-operators that are quite similar to G-operators, except that they reposition a pointer to a focus SSM subgraph instead of return an unsatisfied node-chain. For example, S-operators for principles (s1'), (s3') and (s4') in Section 4.2 are written in pseudo-code as:

```
(S_OP::switch_to_strongest($p)                     % $p = pointer to the current solution focus
    Diff = differential()                          % recompute the differential
    $p = max(belief(Diff)))                        % reposition pointer to the strongest candidate
    return($p))                                    %    in the differential


(S_OP::switch_to_child($p)                         % $p = pointer to the current solution focus
    if   children($p,"D") <> NIL                   % if current focus has D node children
    then $p = first(children($p,"D"))              % reposition pointer to the first child
         return($p))
    else
         return("fail")                            % return: "inapplicable S-operator"
    end-if
```

```
(S_OP::switch_to_sibling($p)            % $p = pointer to the current solution focus
     if pursued($p)                     % if the current solution focus was pursued
         S = siblings($p)               % S = siblings of the current solution focus
         do until S = NIL
              $p1 = pop(S)              % reposition pointer to the first sibling not
              if not pursued($p1)       %    yet pursued
              then return($p1)
         end-do
     end-if
     return("fail"))                    % return: "inapplicable S-operator"
```

In the above examples, terms like **`siblings($p)`** and **`children($p)`** are general-purpose functions that are an integral part of ACE-SSM's inference procedure. These functions know nothing about the semantics of SSMs relative to the application task modeled. Instead, they treat SSMs purely as directed graphs, and hence apply only graph-oriented operations to generate their results.

5.3. Instantiating the Domain Model

Instantiating the domain model, ***K***, means creating and populating a KB, based on the KB structure reflected in the formalized version of ***K***. Respectively, the *application ontology* can be equated with the ontology of the formalized ***K***, where the populated KB is an instantiation of this ontology. Because the domain model follows directly from the goal model, we also refer to the application ontology as the *goal ontology* of the task.

Like in other knowledge modeling approaches, a well-defined domain model can be used to direct the KB population endeavor. The constituents of this model can serve as high-level templates that put constraints on the types of domain knowledge that must be acquired to populate the KB. This idea is discussed extensively, for example, in the context of automated KA tools (Musen, 1989). To illustrate, an automated KA tool called SALT knows the roles that elements of ***K*** fill during problem-solving in its underlying propose-and-revise method, and it can thus present a modeler with fill-in templates for the different parts of the KB being populated (Marcus and McDermott, 1989).

Sometimes, when there already exists an instantiated KB that can be reused across several applications in the same domain, it might be possible to avoid creating a dedicated KB from scratch. Here, the formalized domain model plays a different role. Where $\boldsymbol{K}_D$ describes the structure of the reusable KB in terms of the *domain ontology*, so-called *mapping relations* have to be used to align the domain ontology and the application ontology. Mapping relations are used to close semantic and syntactic gaps between the domain ontology that $\boldsymbol{K}_D$ reflects and the application ontology that ***K*** reflects. When only syntactic gaps exist between ***K*** and $\boldsymbol{K}_D$, the domain ontology and the application ontology can be equated, and it is necessary to define only mapping relations that rename terms in ***K*** to fit $\boldsymbol{K}_D$. This kind of mapping relations can be applied as a pre-processing step to G-operators, K-operators, and S-operators. When there also exist semantic gaps between ***K*** and $\boldsymbol{K}_D$, two other types of mapping relations might be needed. One type is used to augment $\boldsymbol{K}_D$ with objects and relations in ***K*** that are missing in $\boldsymbol{K}_D$. These relations can also be applied as a pre-processing step. Another type of relations is used to transform parts of $\boldsymbol{K}_D$ that are arranged differently in ***K***. For example, such a mapping relation can group attributes of several object types in $\boldsymbol{K}_D$ into one object type in ***K***, to hide a distinction between the object types that the target task does not use in problem-solving. These relations must be applied at run-time.

Although the use of mapping relations is not yet supported by SSM-DKM, it is relatively easy to expand SSM-DKM to handle mapping relations similar to the ones supported by existing knowledge modeling approaches. For example, within the PROTEGE-II workbench, a tool called MARBLE allows to adapt a given application ontology to a given domain ontology by instantiating fill-in templates of mapping relations called *renaming relations*, *filtering relations*, and *class*

mappings (Gennari, Tu, Rothenfluh, and Musen, 1994). Another example is CommonKADS, whose associated formalization languages use mapping relations called *lifting rules* (Fensel and Van Harmelen, 1994). The point is that all approaches that support the use of mapping relations do so on the premise that there is available a domain model that captures an explicit application ontology. We saw above that this requirement is met by the formalized domain model produced by SSM-DKM.

6. Relation to Other Work

At the most general level, a way to position SSM-DKM relative to other knowledge modeling approaches is to look at the facets of a task that such approaches aim to model explicitly. We distinguish between three main facets: ***what*** the task entails producing as solutions (e.g., SSMs), ***how*** the task is solved, and ***using-what*** domain knowledge solutions are produced. In this sense, SSM-DKM is the result of another step in the progression of work focusing on how to model and represent various task facets explicitly. Existing approaches focus on making explicit mainly the *how* and *using-what* facets. SSM-DKM focuses on making explicit the *what* facet as well. More specifically, in SSM-DKM, because the notion of an SSM defines the *what* facet, an SSM is a central structure that drives the modeling of other facets. As we showed here and elsewhere (Benaroch, 1998), modeling the *what* facet explicitly facilitates the construction of explicit SSMs as well as permits the modeling of the *how* facet in declarative terms.

Based on (Benaroch, 1997), the rest of this section compares SSM-DKM to the leading knowledge modeling approaches that aim at producing explicit knowledge models, including: CommonKADS (Wielinga et al., 1992), Chandrasekaran and Johnson's (1993) task-structures framework (TSF), Steels' (1990; 1993) Components of Expertise (COE), and reusable problem-solving methods (PSMs) (e.g., Marcus and McDermott, 1989; Eriksson et al., 1995; Fensel and Straatman, 1998). We first compare the modeling constructs comprising approach-specific conceptual knowledge models, then look at the model construction process that these approaches support, and finally discuss how these approaches deal with formalization and validation. We do not extend the comparison to the instantiation stage for a simple reason. While all approaches provide logical arguments in favor of various operational benefits (e.g., explanation, maintainability) that follow from their use, there is a growing sense that these claimed benefits remain to be evaluated empirically (e.g., see [Menzies, 1998] for a review).

6.1. Conceptual Knowledge Modeling Constructs

Because the compared approaches differ on their modeling constructs, the conceptual knowledge models that they produce do not capture equally well all facets (*what*, *how*, and *using-what* facets) of the tasks they aim to model.

Starting with the *what* facet of a task, a concrete way to model this facet is to describe the SSMs that the task entails producing as solutions. Only SSM-DKM models this facet explicitly, using the goal model construct. The goal model captures the semantics of domain phenomena modeled using SSMs, thus reflecting the application (goal) ontology. A key derived benefit is the ability of the operationalized knowledge model (or resulting KBS) to construct explicit SSMs. KBSs that do not have explicit knowledge about the *what* facet create only implicit SSMs that are inaccessible for inspection and interpretation. For example, the working memory of a production system contains the elements of an SSM without showing their dynamic and temporal relationships.

The other approaches implicitly model parts of the *what* facet, through modeling of the *how* facet using different constructs. TSF's, COE's and PSMs' task-structure as well as CommonKADS' task model capture the solution strategy applied to the task through the notion of

procedural subtasks. In CommonKADS, the task model is created based on the inference-structure construct. The inference-structure construct shows the elementary steps (inferences) that a solver takes during problem solving and how they relate to each other via input-output roles that object types fill in them, whereas the task model hierarchically groups inferences in the inference-structure based on a bottom-up abstraction of their related objectives. By contrast, in TSF, COE and PSMs, a task-structure shows a top-down mapping of the target task and its subtasks to methods, where the methods pre-define the choice of subtasks. Despite this difference, both the task model and the task-structure constructs show a hierarchical decomposition of the task all the way to primitive subtasks (inferences), where higher level subtasks procedurally "hardwire" the strategic knowledge (or principles) underlying the sequencing of lower level subtasks. SSM-DKM's strategy model captures these same strategic principles in purely declarative terms pertaining to the application (goal) ontology.[2] SSM-DKM does not use the notion of method (and inference-structure). This should be no surprise – Newell (1990) acknowledged that the notion of method is implicit in the problem-space paradigm. A possible weakness of SSM-DKM is its lack of support of task decomposition, or hierarchical organization of strategic knowledge. For complex application tasks, the benefits from supporting decomposition are obvious.

Going back to the *what* facet, parts of this facet are implicit in the inference-structure and method constructs. In the inference-structure construct, an inference linking specific object types through input-output relations could be considered to implicitly place a specific type of link between nodes in SSMs. The inference-structure could be hence said to implicitly reflect local goal constraints. Respectively, it seems that CommonKADS could explicitly model part of the *what* facet by linking its inference-structure construct to SSM-DKM's goal model construct. As to the method construct, recent work on reusable methods incorporates more of the *what* facet into the notion of method ontology. One of the items included in the ontology of a method is the method's *competence*, which reflects the function (goal) of the method (e.g., Fensel and Straatman, 1998). For example, the competence of the propose-and-revise method is defined in terms of assumptions like: (1) the goal is to design a "system" defined in terms of components with parameters and values of these parameters, and (2) the solution is an assignment of values for parameters that satisfy internal constraints (Coelho and Lapalme, 1996). These assumptions can be seen as the method's pre-conditions and post-conditions, respectively. In essence, such assumptions pertain to the *what* facet of the tasks that a method targets, and they correspond to what SSM-DKM calls global goal constraints. However, except for SSM-DKM, none of the other approaches that do capture parts of the *what* facet do so in support of producing explicit SSMs. One of the results is their inability to model the *how* facet in declarative terms pertaining to the application (goal) ontology.

Finally, in capturing the *using-what* facet, the compared approaches seek to model the application ontology in terms of the static object types and relations that fill input-output data roles during problem solving. Here, too, approaches differ on the way they model this facet. In SSM-DKM, the application ontology directly follows from the goal model construct, which identifies explicit input roles that static object types and relations fill in the SSMs constructed for a task. This is why we said in Section 5.3 that SSM-DKM equates the goal ontology with the application ontology. In TSF, COE and PSMs, the application ontology is implied by input-output roles that object types fill in methods, where only COE makes these roles explicit using its model-

[2] This point can be illustrated using a simple example. Consider a task-structure with one top-level task called Pursue-Hypothesis, which has only two elementary subtasks (inferences): Test-Hypothesis looks for findings (symptoms) to support a given disease (malfunction), and Refine-Hypothesis specializes a given disease. In SSM-DKM, Test-Hypothesis and Refine-Hypothesis would be two K-operators that take as arguments the node-chains $D \rightarrow ?F$ and $D \rightarrow ?Ds$, respectively. Now, suppose that Pursue-Hypothesis always invokes Test-Hypothesis before Refine-Hypothesis, to induce a breadth-first search of some disease taxonomy. In SSM-DKM, the strategic principle that Pursue-Hypothesis embodies would be represented as the local strategic principle $\{D \rightarrow ?: D \rightarrow ?F \succ D \rightarrow ?Ds\}$. This declarative principle would induce the same search pattern.

dependency diagram construct. In CommonKADS, the application ontology also follows from the input-output roles that object types fill in the inference-structure. But, an important difference is that an inference-structure shows both static and dynamic object types (e.g., disease hypotheses). Unlike CommonKADS, the other approaches do not explicitly model dynamic object types. In SSM-DKM, dynamic object types are SSM constituents, e.g., disease hypotheses (or candidate solutions) correspond to SSM subgraphs. In this sense, it might be useful for SSM-DKM to explicitly model dynamic object types, like in CommonKADS, to simplify the specification of global goal constraints in terms of dynamic object types.

So far, the most noticeable difference between the compared approaches is along the task facet driving the conceptual modeling effort. Except for SSM-DKM, all other approaches focus on the *how* facet, or on the solution strategy applied to the task. Although Newell (1982) warned that this could mislead one to consider implementation details too early, these approaches provide modeling constructs that help to avoid this danger. SSM-DKM takes a more drastic step in this direction by focusing on the *what* facet. When problem solving is formulated as search, focusing on the *what* facet also allows SSM-DKM to ground *how*-related control decisions in the need to evolve SSMs into a state specified by the goal model, thus permitting to model strategic knowledge in declarative terms pertaining to the application (goal) ontology. Some recent work in the context of PSMs also adopts the view of problem solving as search, for the purpose of better describing the ontology of reusable methods. For example, Motta and Zdrahal (1998) proposed adding two optional elements to the description of method ontologies: "focus" and "focus selection knowledge". These two terms correspond, in part, to the role of strategic principles in SSM-DKM (see Section 3.2). However, unlike SSM-DKM, Motta and Zdrahal's proposal does necessarily facilitate the modeling of strategic knowledge in declarative terms, because their view of problem solving as search is not anchored in the need to evolve an explicit SSM into some desired state.

6.2. Construction Process of Conceptual Knowledge Models

As to the process of constructing a conceptual knowledge model, we can distinguish between construction from scratch and construction through reuse. Both SSM-DKM and CommonKADS support construction from scratch. Naturally, due to the difference in their orientation (discussed in Section 6.1), they follow a different process on which we will not elaborate here. The more significant difference is on how the compared approaches support model construction through reuse.

TSF and COE offer libraries of reusable building blocks – generic method/task mappings – using which conceptual models can be constructed top-down. Both advocate configuring a task-structure by gradually mapping (sub)tasks to specific methods, which formulate expectations about problem-solving behaviors by identifying their implied subtasks and data requirements. However, this configuration process leaves some critical decisions up to the modeler (e.g., when to expand the task-structure depth-first or breadth-first, depending on the degree of coupling of subtasks). The consequences are illustrated in an empirical study involving three knowledge engineers who used TSF for the same task (Allemang and Rothenfluh, 1992): (1) one of the subjects was unable to answer pointed questions about how he constructed the task-structure; (2) the task-structure produced by one subject was significantly different of those of the other two subjects; and (3) even the task-structures of these other two subjects had several considerable differences, despite some high-level similarity.

CommonKADS similarly supports reuse by offering a library of generic conceptual models. Earlier versions of CommonKADS (Wielinga et al., 1992) required selecting a generic model from the library by searching the library top-down based on the task's input-output goal (design, diagnosis, etc.), and then using the model as a skeleton using which think-aloud protocols could be abstracted. However, experience revealed that a selected model that partially fits the task can be misleading, as Rademakers and Vanwelkenhuysen (1993, p. 368) explain: "A danger which we actually experienced

is that the model largely influences what we perceive. It makes us look for certain data, neglecting other issues, which finally turn out to be very relevant." A key source of this problem is that CommonKADS' generic models reflect a *how*-oriented ontology. These models' focus on the *how* facet blurs details specific to the *what* facet, often leading a modeler to adapt, knowingly or unknowingly, parts of the task to their underlying solution strategy.

Some work on libraries of reusable PSMs exhibits problems similar in nature. For example, Benjamins (1994) built a library of methods, which a tool called TINA uses to compose task-structures by applying rewrite rules that select methods whose data requirements are met by available domain knowledge. One experience with this library shows that: "the most suitable model generated by the library had to be modified in several ways ... This caused significant difficulties, both in identifying the modification requirements, and in creating the necessary adaptation." (Orsvarn, 1996, p. 1).

Other work on libraries of reusable methods (e.g., Motta and Zdrahal, 1996; Fensel and Straatman, 1998), some of which is done in the context of CommonKADS (Akkermans, Wielinga and Schreiber, 1993), recognizes that reusable methods must make explicit features and assumptions of their target application tasks. Relatedly, KMOST (Benaroch and Tanniru, 1996) is an approach which proposes that mapping a task to a suitable method requires matching different features and assumptions of the task against those of the kind of tasks that different generic methods can solve. KMOST structures the task analysis preceding method selection by providing guidelines that help to identify for each specific application (sub)task a different minimal set of key task features that have to be characterized to aid in method selection for that task. In line with KMOST, recent work on reusable methods expands the ontologies of methods to include features and assumptions of their target tasks, e.g., through the notion of method competence. The bottom line is that work on reusable methods is increasingly capturing more information about the *what* facet of tasks, however, this information is incorporated into the description of methods whose focus is on the *how* facet.

SSM-DKM facilitates a reuse approach that focuses on the *what* facet, on the premise that conceptual knowledge models of tasks that construct SSMs similar in topology involve a similar goal (application) ontology. SSM-DKM proposes a library of conceptual models that is organized as an object-oriented typology of goal (application) ontologies pertaining to types of processes, structures, functions, behaviors etc. modeled using SSMs (Benaroch, 1998). Respectively, SSM-DKM also offers structured adaptation guidelines. These first identify goal constraints that have to be modified in the generic goal model being adapted, to reflect *what* features specific to the target task (e.g., structural differences across the SSMs that the generic model supports and the SSMs that the target task creates). Then, because strategic knowledge is expressed in terms of the goal (application) ontology, these guidelines use the adapted goal model to point out those parts of the corresponding strategy model that must be adapted. Hence, in SSM-DKM, strategic knowledge is also reusable. Benaroch (1996) illustrates these adaptation guidelines by showing how the conceptual model for the medical diagnosis task used in this paper is adapted to the mechanical diagnosis task solved by a KBS called CHECK. To summarize, SSM-DKM gives attention to the *how* facet only after the *what* facet has been adapted, because *how* knowledge is anchored in a *what*-oriented ontology.

6.3. Formalization and Validation of Conceptual Knowledge Models

Conceptual modeling aside, the compared approaches vary on their support of formalization and validation. COE supports no formalization or validation. Its COMMET workbench (Steels, 1993) allows mapping a conceptual model directly to executable computational mechanisms, and its KREST tool (Geldof and Soldzian, 1994) can detect only macro errors in the conceptual model (e.g., check that a leaf in the task-structure is not mapped to a method, and that all subtasks are gathered in a single task tree). In TSF, a task-structure is formalized as a SOAR problem-space computational model, but it is unclear how this model is then validated (Chandrasekaran and Johnson, 1993). The

remaining approaches do support formalization and validation, but to a varying degree.

CommonKADS stimulated the development of over 10 dedicated formalization languages (Fensel and Van Harmelen, 1994). Most of these produce a structure-preserving formalized model because their formalization constructs correspond one-to-one to the constructs in conceptual models. A structure-preserving formalized model and its conceptual model source can be viewed as one entity combining both semi-formal and formal descriptions. Two benefits of this trait are: the simplification of revision issues once a modeler is given focused validation feedback, and the ability to produce a structure-preserving design that simply adds localized implementation details to the formalized model. We saw that SSM-DKM's formalization approach also produces structure-preserving models, however, without relying on a dedicated formalization language.

A key problem with CommonKADS' dedicated formalization languages goes back to the conceptual models that they target. They involve more than one representational scheme because CommonKADS models strategic knowledge procedurally. This characteristic limits their ability to support validation. For example, consider ML(2), a formalization language that focuses on the logical semantics of conceptual models (Van Harmelen and Aben, 1996). In ML(2), the declarative parts of a conceptual model are expressed using predicate calculus, facilitating a static validation of those parts. But, the developers of ML(2) admit that offering similar support for the task model is not yet possible because this model is expressed procedurally (Van Harmelen, Aben, Ruiz, and Van de Plassche, 1996). Another example is MODEL-K (Voss and Karbach, 1993), an imperative language that provides a formalized model with the operational semantics needed to produce a full working system. MODEL-K allows plugging into the body part of inferences various computational mechanisms implemented using Prolog, Lisp, message passing rules, etc. Yet, MODEL-K's procedural orientation provides only for some dynamic validation, whereby separate parts of the formalized model can be executed to check for their operational adequacy and efficiency. KARL is one language for formalizing the procedural parts of a conceptual model as dynamic logic programs (involving loops, conditional statements, etc.), which a system called KIV can use to check that those procedural parts achieve their required function (Fensel and Schoenegge, 1997).

By contrast, since SSM-DKM models strategic knowledge declaratively, it can formalize all parts of its conceptual models using the same representational scheme – as conditional reflective predicate-logic sentences that interact with SSMs. As we saw in Section 4.3, this permits SSM-DKM to extend the use of conventional logic-based techniques to the static validation of strategic knowledge. Additionally, the logical semantics of strategic knowledge provides for its operational semantics as well. It should be easy to compare different problem-solving strategies in terms of their tendency to focus search effectively, avoid dead-ends, improve computational efficiency, etc. A modeler has to just change declarative strategic principles and rerun the model, instead of recode them procedurally like in MODEL-K and ML(2).

7. Conclusion and Future Research

This paper presented a structured knowledge modeling methodology called SSM-DKM and its conceptual foundations. SSM-DKM is intended to produce knowledge models that support the construction of explicit SSMs, which could endow a KBS with considerable operational benefits. For this purpose, SSM-DKM focuses on modeling the *what* facet of a task, in terms of the structure of SSMs that must be created as solutions to problem situations associated with the task. The previous section compared SSM-DKM to the dominant knowledge modeling approaches, clearly showing the modeling benefits derived from focusing on the *what* facet.

Perhaps the most important modeling benefit is the ability to express strategic knowledge (or problem-solving knowledge) declaratively, instead of hardwiring this knowledge into production rules or procedural subtasks. In essence, this allows viewing strategic knowledge as if it forms a KB

separate from the object-level KB containing domain knowledge. This view opens the possibility of using a host of available KBS techniques to formalize strategic knowledge, validate it, apply it, reason about it, etc. In particular, this view allows framing intelligent reasoning uniformly in terms of the two problem-spaces associated with problem search and knowledge search, as Newell advocated in the context of SOAR (Newell, 1990). The ability to do so effects mainly two issues. Binding of control occurs at run-time, unlike in standard sequential and procedural modeling languages where the binding occurs at the time the system is created. Additionally, it allows modeling strategic knowledge in a way that provides a KBS with the ability to dynamically reflect on its own problem-solving behavior.

Like with any new knowledge modeling methodology, much additional research is needed mainly in four areas. One area pertains to the application of SSM-DKM. So far SSM-DKM has been applied to medical and mechanical diagnosis tasks, and it is currently being applied to design tasks in the domains of financial risk management and elevator configuration. It is necessary to apply SSM-DKM in other domains, to gain more experience that would help to appreciate its associated benefits and drawbacks.

A second area for future research deals with automated tool support. It is necessary to investigate how to design automated tools for supporting the different stages in SSM-DKM. However, before such support tools can be developed, empirical work is needed to gain insights into what knowledge engineers find to be practical strengths and weaknesses of SSM-DKM. Such studies could help to answer several useful question, for example: (1) are the notations that SSM-DKM uses (e.g., ER diagram, object-relation matrix) suitable for what they intend to elicit? and, (2) at what modeling stage (conceptualization, formalization, instantiation) should one start expressing parts of a knowledge model in graph-oriented terms underlying the structure of SSMs, instead of in purely domain-specific terms.

The third area has to do with supporting reuse. It is necessary to expand SSM-DKM to support the construction of knowledge models through reuse and adaptation of generic ones. For this purpose, it is necessary to develop a taxonomy of goal models based on the similarity of what SSMs for tasks of the same type represent – processes, functions, structures, behaviors, etc. In other words, there is a need to develop a typology of goal ontologies that will allow reusing *what* and *how* knowledge following the scheme outlined in section 6.2 and more extensively discussed by Benaroch (1996, 1998). A good starting point might be the extensive literature on reusable problem-solving methods. Since all such methods construct (implicit) SSMs through interactions of their implied primitive subtasks (or inferences), it would be fruitful to analyze the inputs and outputs of these subtasks. Doing so for a certain method would not only reveal much of the goal ontology underlying the class of tasks that the method targets, but also serve as a first step toward using the declarative strategic principles formalism of SSM-DKM to express the *how* knowledge that the method embodies. A related issue is the need to see how SSM-DKM might go about supporting a hierarchical organization of declarative strategic principles, parallel to the way methods support task decomposition.

The last area for future research is in the intersection of SSM-DKM and ACE-SSM. The operational model that SSM-DKM produces is intended to run under the ACE-SSM architecture. Being that ACE-SSM is a straightforward implementation of the conceptual reasoning process we described in Section 2.2, it involves some overhead cost for applying goal constraints captured using G-operators and strategic knowledge captured using S-operators. One possibility is to develop more efficient algorithms for ACE-SSM's inference procedure to lower this overhead cost. Another possibility is to develop a more generic architecture. For example, we can think of a production system that is capable of reasoning with three different KBs: an object-level KB capturing domain knowledge, a meta-level KB capturing goal knowledge, and a strategy KB

capturing strategic knowledge. Of course, such an architecture would require revising parts of the instantiation stage in SSM-DKM.

References

[1] AKKERMANS, J.M., WIELINGA, B. & SCHREIBER, T.H. (1993). Steps in constructing problem-solving methods. In A. AUSSENAC et al., Eds. *Knowledge-Acquisition for Knowledge-Based Systems*, Lecture Notes in AI, no. 723. London: Springer-Verlag.

[2] ALLEMANG, D. & ROTHENFLUH, T.E. (1992). Acquiring knowledge of knowledge acquisition: A self study of generic tasks. In T. WETTER, A.D. ALTHOFF, J. BOOSE, B.R. GAINS, M. LINSTER & F. SCHMALHOFER, Eds. *Current Developments in Knowledge Acquisition*, pp. 353-372. London: Springer-Verlag.

[3] BENAROCH, M. (1998). Goal-directed reasoning with ACE-SSM. *IEEE Transactions on Knowledge and Data Engineering*, forthcoming. http://sominfo.syr.edu/facstaff/mbenaroc/resume/publications.htm.

[4] BENAROCH, M. (1997). Knowledge modeling research: a critical analysis. Working Paper, School of Management, Syracuse University.

[5] BENAROCH, M. (1996). Roles of design knowledge in knowledge-based systems. *International Journal of Human-Computer Studies*, **44**(5), pp. 689-721.

[6] BENAROCH, M. & TANNIRU, M. (1996). Conceptualizing structurable tasks in the development of knowledge-based systems. *Decision Sciences*, **27**(3), pp. 415-449.

[7] BENJAMINS, V.R. (1994). On the role of problem solving methods in knowledge acquisition – experiments with diagnostic strategies. In L. STEELS, G. SCHREIBER & W. VAN DE VELDE, Eds. *A Future for Knowledge Acquisition*, pp. 137-157. London: Springer-Verlag.

[8] BENJAMINS, V.R. & JANSWEIJER, W. (1994). Towards a competence theory of diagnosis. *IEEE Expert*, **9**(5), pp. 43-52.

[9] BOOSE, J.H. (1985). A knowledge acquisition program for expert systems based on personal construct psychology. *International Journal of Man-Machine Studies*, **23**, pp. 495-525.

[10] BREUKER, J. (1997). Problems in indexing problem solving methods. In D. FENSEL, Ed. *Problem Solving Methods for Knowledge Based Systems*. pp. 19-36. Nagayo, Japan: IJCAI Press.

[11] CHANDRASEKARAN, B. & JOHNSON, T.R. (1993). Generic tasks and task structures: history, critique and new directions. In J.M. DAVID, J.P. KRIVINE & R. SIMMONS, Eds. *Second Generation Expert Systems*, pp. 233-272. London: Springer-Verlag.

[12] CHANDRASEKARAN, B., JOHNSON, T.R. & SMITH, J.W. (1992). Task-structure analysis for knowledge modeling. *Communications of the ACM*, **35**(9), pp. 124-137.

[13] CLANCEY, W.J. (1992). Model construction operators. *Artificial Intelligence*, **53**, pp. 1-115.

[14] CLANCEY, W.J. (1988). Acquiring, representing and evaluating a competence model of diagnostic strategy. In M. CHI, R. GLASER & M.J. FARR, Eds. *The Nature of Expertise*, pp. 343-418. Hillsdale, NJ: Lawrence Erlbaum Associated.

[15] COELHO, E. & LAPALME, G. (1996). Describing reusable problem-solving methods with an ontology of roles. *Tenth KA for KBSs Workshop, KAW'96*. ftp://ksi.cpsc.ucalgary.ca/KAW/KAW96/34coelho.ps.Z.

[16] CRAGUN, B. & STEUDEL, H. (1987). A decision-table-based processor for checking completeness and consistency in rule-based expert systems. *International Journal of Man-Machine Studies*, **26**(5), pp. 633-648.

[17] ERIKSSON, H., SHAHAR, Y., TU, S.W., PUERTA, A.R. & MUSEN, A. (1995). Task modeling with reusable problem-solving methods. *Artificial Intelligence*, 79(2), pp. 293-326.

[18] FENSEL, D. & SCHOENEGGE, A. (1997). Using KIV to specify and verify architectures of knowledge-based systems. *Proceedings of the 12th IEEE International Conference on Automated Software Engineering, ASEC-97*, Incline Village, Nevada.

[19] FENSEL, D. & STRAATMAN, R. (1998). The essence of problem-dolving methods: making assumptions to gain efficiency. *International Journal of Human-Computer Studies*, **48**(2), pp. 181-215.

[20] FENSEL, D. & VAN HARMELEN, F. (1994). A comparison of languages which operationalise and formalise KADS models of expertise. *The Knowledge Engineering Review*, **9**(2), pp. 105-146.

[21] GELDOF, S. & SLODZIAN, A. (1994). From verification to modeling guidelines. In L. STEELS, G. SCHREIBER & W. VAN DE VELDE, Eds. *A Future for Knowledge Acquisition*, pp. 226-243. London: Springer-Verlag.

[22] GENESERETH, M.R. & NILSSON, N.J. (1987). *Logical Foundations of Artificial Intelligence*. California:Morgan Kauffmann Publishers.

[23] GENNARI, J.H., TU, S.W., ROTHENFLUH, T.E. & MUSEN, M.A. (1994). Mapping domains to methods in support of reuse. *International Journal of Human-Computer Studies*, **41**(3), pp. 399-424.

[24] GRUBER, T.R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, **5**, pp. 199-220.

[25] MARCUS, S. & MCDERMOTT, J. (1989). SALT: a knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence*, **39**(1), pp. 1-38.

[26] MENZIES, T. (1998). Evaluation issues for problem solving methods. *Eleventh KA for KBSs Workshop*, KAW'98. http://ksi.ucalgary.ca/KAW/KAW98/menzies2/.

[27] MOTTA, E. & ZDRAHAL, Z. (1998). A principled approach to the construction of a task-specific library of problem solving components. *Eleventh KA for KBSs Workshop*, KAW'98. http://ksi.ucalgary.ca:80/KAW98S/motta/.

[28] MUSEN, M.A. (1989). *Automated Generation of Model-Based Knowledge-Acquisition Tools*. London: Pitman.

[29] NEWELL, A. (1990). *Unified Theories of Cognition*. Boston, MA: Harvard Press.

[30] NEWELL, A. (1982). The knowledge level. *Artificial Intelligence*, **18**(1), pp. 82-127.

[31] ORSVARN, K. (1996). Principles for libraries of task decomposition methods – conclusions from a case-study. *Tenth KA for KBSs Workshop, KAW'96*. ftp://ksi.cpsc.ucalgary.ca/KAW/KAW96/55orsvarn.ps.Z.

[32] RADEMAKERS, P. & VANWELKENHUYSEN, J. (1993). Generic models and their support in modeling problem solving behavior. In J. DAVID, J.P. KRIVINE & R. SIMMONS, Eds. *Second Generation Expert Systems*. London: Springer-Verlag.

[33] STACHOWITZ, R.A. (1990). Verification and validation of knowledge-based systems. Tutorial Notes. *Hawaii International Conference on Systems Science*, Kailua-Kona, Hawaii.

[34] STEELS, L. (1993). The componential framework and its role in reusability. In J.M. DAVID, J.P. KRIVINE & R. SIMMONS, Eds. *Second Generation Expert Systems*, pp. 273-298. London: Springer-Verlag.

[35] STEELS, L. (1990). Components of expertise. *AI Magazine*, **11**(2), pp. 30-49.

[36] THAYSE, A. (1988). *From Standard Logic to Logic Programming*. England: John Wiley & Sons.

[37] VAN DE VELDE, W. (1994). An overview of CommonKADS. In J. BRUEKER & W. VAN DE VELDE, Eds. *CommonKADS Library for Expertise Modeling*. pp. 9-29. Amsterdam, Netherlands: IOS Press.

[38] VAN DE VELDE, W. (1993). Issues in knowledge level modeling. IN J.M. DAVID, J.P. KRIVINE & R. SIMMONS, Eds. *Second Generation Expert Systems*, pp. 211-231. London: Springer-Verlag.

[39] VAN HARMELEN, F. & ABEN, M. (1996). Structure-preserving specification languages for knowledge-based systems. *International Journal of Human-Computer Studies*, **44**(2), pp. 187-212.

[40] VAN HARMELEN, F., ABEN, M., RUIZ, F. & VAN DE PLASSCHE, J. (1996). Evaluating a formal KBS specification language. *IEEE Expert*, **11**(1), pp. 56- 62.

[41] VERMASEN, A. & WERGELAND, T. (1994). A formally-based methodology for deriving verifiable expert systems from specifications. *Proceedings of the AAAI'94 Workshop on Validation and Verification of Knowledge-Based Systems*, Seattle, WA.

[42] VOSS, A. & KARBACH, W. (1993). Implementing KADS expertise models with Model-K. *IEEE Expert*, **8**, pp. 74-81, August.

[43] WIELINGA, B., VAN DE VELDE W., SCHREIBER G. & AKKERMANS H. (1993). Towards a unification of knowledge modeling approaches. In J.M. DAVID, J.P. KRIVINE & R. SIMMONS, Eds. *Second Generation Expert Systems*, pp. 299-331. London: Springer-Verlag

[44] WIELINGA, B.J., SCHREIBER, A.T. & BREUKER, J.A. (1992). KADS: a modeling approach to knowledge engineering. *Knowledge Acquisition*, **4**(1), pp. 5-53.