The Ontology of Tasks and Methods

B. Chandrasekaran and J. R. Josephson

Laboratory for AI Research
The Ohio State University
Columbus, OH 43210
Email: {chandra, jj}@cis.ohio-state.edu

Abstract

Much of the work on ontologies in AI has focused on describing some aspect of reality: objects, relations, states of affairs, events, and processes in the world. A goal is to make knowledge sharable, by encoding domain knowledge using a standard vocabulary based on the ontology. A parallel attempt at identifying the ontology of problem-solving knowledge would make it possible to share problem-solving methods. For example, when one is dealing with a type of problem known as *abductive inference*, the following are some of the terms that recur in the representation of problem-solving methods: *hypotheses, explanatory coverage, evidence, degree of confidence, plausibility, composite hypothesis*, etc. Method ontology, in good part, is task- and method-specific. "Generic Tasks," "Heuristic Classification," "Task-specific Architectures," and "Task Structures" are representative bodies of work in the knowledge-systems area that have focused on problem-solving methods. However, connections have not been made to work that is explicitly concerned with ontologies. Making such connections is the goal of this paper.

Ontologies as Content Theories

In philosophy, ontology is the study of the kinds of things that exist. In AI, the term has largely come to mean one of two things:

1. a representation vocabulary, typically specialized to some domain or subject matter. Usually, a language based on Predicate Calculus and employing this vocabulary is used to encode knowledge that can be shared.
2. occasionally, a body of knowledge describing some domain, typically a commonsense knowledge domain. For example, CYC (Lenat and Guha 1988) often refers to its knowledge representation of some area of knowledge its ontology.

In this paper, we use the term ontology in the first sense, except that we broaden the notion of knowledge to include knowledge about problem solving.

The current interest in ontologies is really the latest version of our field's alternation of focus between content theories and mechanism theories. Sometimes everyone gets excited by some mechanisms, be they rule systems, frame languages, connectionist systems, fuzzy logic, etc. The mechanisms are proposed as the secret of making intelligent machines. At other times, there is a realization that, however great the mechanism, it cannot do much without a good content theory of the domain on which to set the mechanism to work. Moreover, it is often realized that once a good content theory is available, many different mechanisms might be used to implement effective systems, all using essentially the same content (Chandrasekaran 1994).

In AI, there have been several attempts to characterize the essence of what it means to have a content theory. McCarthy and Hayes' (McCarthy and Hayes 1969) Epistemic versus Heuristic distinction, Marr's three levels (Marr 1982) – Information Processing Strategy level, algorithms and data structures level, and physical mechanisms level – and Newell's Knowledge Level versus Symbol Levels (Newell 1981) all grapple in their own ways with characterizing content. Ontologies are quintessentially content theories. However, it is not entirely clear what they are a content theory of. We identify two dimensions on which such content theories might lie.

Dimensions for Ontology Specification in Knowledge Systems

Knowledge systems need to have two kinds of knowledge:

1. Knowledge about the objective realities in the domain of interest (Objects, relations, events, states, etc. that obtain in some domain)
2. Knowledge about problem-solving

Even though the term "domain knowledge" is normally used to include knowledge about problem-solving in some domain, in this paper, we will use it as a short hand for "domain factual knowledge," i.e., to describe the first type of knowledge. The domain knowledge dimension has by far been the focus so far of most of the AI investigations on ontologies. The dimension of representing problem-solving knowledge will be the focus of this paper. Identification of ontologies for the problem-solving dimension is equally important. It can contribute to sharing and reuse of problem-solving knowledge just as much as ontologies for domain knowledge can contribute to sharing and reuse of domain knowledge.

We start by reviewing the conceptual basis of the work on ontologies for domain knowledge. Then, we introduce and discuss issues in ontology construction for the problem-solving knowledge dimension.

Domain Knowledge

We characterized domain knowledge as factual knowledge about some domain. In the context of building problem-solving systems, it is useful to make an additional distinction between knowledge that is true in the domain in general and knowledge that pertains to specific problem instances. This is a somewhat artificial distinction when it comes to human knowledge. However, this distinction is useful because most problem-solving systems are organized in such a way that information about the specific problem instance that is being solved is kept separate from the knowledge base. As new problems are presented, the problem-instance database is refreshed, while the same knowledge base is used again and again. For example, in the case of medical knowledge-based systems, this knowledge base would have knowledge about the relations between diseases and symptoms. The patient database would initially contain the patient's initial manifestations, and as the problem-solving proceeds, the database would contain additional inferred knowledge about the patient, until the hypothesis about what is wrong with the patient is recorded in the database. From a representation point of view, information in both the bases is knowledge. Issues in representing both sets of facts, what is true of the specific problem instance and what is true in the domain in general, are quite similar.

The most common representation language in AI, Predicate Calculus, presupposes the following theory of existence: that there are things – objects -- in the world (the world need not necessarily be physical), there are properties which the objects may possess, and there may exist various relations between the objects. (Predicates in Predicate Calculus are used to represent the relations as well as the properties, the latter being represented as monadic predicates.) The language has additional symbols, the quantifiers and connectives. We do not discuss them here.

Additional formalisms involving time are needed to bring in states, events and processes. With the representational repertoire of objects, relations, states, events and processes, and ubiquitous general relations such as those involving part-whole, class-instance and cause-effect a good deal of domain knowledge can be modeled.

The representational repertoire of objects, relations, states, events and processes does not say anything about what classes of objects, relations, states, events, and processes exist. They are left as commitments to be made by the person modeling the domain of interest. For example, some one, faced with expressing his knowledge of a certain part of the world, might assert that there are certain categories of things called animals, minerals and plants,; that *Has-Life(x),* and *Contains-carbon(x)* are relevant properties for the objects; that *Larger-than(x,y), Can-eat(x,y)* are two of the relations that may be true or false between any two objects. These commitments are not arbitrary -- any old declaration of classes and relations won't do. For them to be useful, such commitments should reflect some underlying reality, i.e., should reflect real existence, hence the term "ontology" for such commitments.

We won't dwell in this paper on the well-known ways in which the development of domain-specific ontologies leads to the development of special-purpose representation languages for modeling domains for which the terms are relevant.

There is no sharp division between domain-independent and domain-specific ontologies in representing knowledge. For example, the terms *object, physical object, device, engine, and diesel engine*, all describe objects, but in an order of increasing domain-specificity. Similarly terms for relations between objects can span a range as well: e.g., *connected (component1, component2)* relation can be specialized as *electrically-connected, physically-attached, magnetically-connected* and so on.

There have been several recent attempts at creating engineering frameworks in which to construct ontologies. Neches, et al (Neches, Fikes et al. 1991) describe an enabling technology called KIF intended to facilitate expression of domain factual knowledge using a Predicate Calculus-like formalism. A language called Ontolingua (Gruber 1992; Gruber 1993) has been proposed to aid in the construction of portable ontologies. In Europe, the KADS project has take a similar approach to modeling domain knowledge (Wielinga, Schreiber et al. 1992).

Before we leave this section, we want to make two points about Predicate Calculus. The first point is about the sufficiency of Predicate Calculus notions as the basis

for ontologies. When the idea of knowledge is extended to include images or other sense modalities, it is possible that we will need other basic formalisms. However, for now, AI is using Predicate Calculus in the same spirit in which Newell proposed it as an appropriate, but not necessarily privileged, language in which to encode the Knowledge Level description of some system. We do not wish to engage in a discussion in this paper about the limitations of this view, except to note that, in the current state of knowledge systems technology -- whether one uses frames, semantic nets, or rule languages -- one way or another, the underlying semantics of knowledge representation seems to draw on the ontological categories of predicates and relations.

The second point concerns the implications for implementation in adopting Predicate Calculus as the formalism for describing knowledge ontologies. The use of the language of objects and predicates for discussing ontologies does not imply any commitment to implementing the knowledge system in any variant of Predicate Calculus. One is simply taking a Knowledge Level stance in describing the knowledge system, whatever the means of implementation. In this view, we can ask of any intelligent system, even one implemented in, say, neural networks, "What does the system know?" As we shall see from the next section, we can also ask about it, "What is its method of problem-solving?"

Knowledge of Problem-solving Methods

Terms which are exclusively internal constructs for the problem solver will be part of the vocabulary for the problem-solving knowledge dimension, while an agent's knowledge (really beliefs) about the objective facts in the domain will be part of the domain knowledge dimension.

How fundamental is the distinction between objective knowledge about the domain and problem-solving knowledge? Actually, from one point of view it is not fundamental at all. Once we have terms in which to describe processes in the word, we have the basics for describing the behaviors of objects, be they cognitive behaviors or behaviors of other sorts, like that of humans constructing houses or electronic buzzers making sound in response to button-pushing. Behavior is a kind of process, and can be described as a sequence of state changes in some appropriately defined object. On the other hand it is useful to make the distinction within a cognitive agent between representations which model the agent's beliefs about objective reality and constructs intended to use these beliefs in the service of problem-solving goals.

Let us consider a knowledge system intended to perform diagnosis in some domain. Terms such as *device, event, component,* and *component connection* are clearly part of the ontology in the knowledge dimension. Additionally, the notion of diagnosis as a task does not arise unless we have the notion of expecting certain functions of the device. Thus, the ontology for domain knowledge would also include terms such as: *observable event, function of component/device, malfunction, normal/ abnormal behavior*, as well as relational knowledge such as *Can-cause(malfunction, {observation | malfunction})*. The domain knowledge might also have process knowledge in the form of causal processes that realize the function in the device.

Problem-solving knowledge specifies how to use factual knowledge about the domain to solve the problem. In the following we will take a Knowledge Level stance towards describing this knowledge. That is, we will describe the abstract structure of this knowledge, what types of entities it consists of, and what relations hold between these entities. In particular, the structure will show the relation to the elements that comprise domain knowledge, also described at the Knowledge Level. We will describe the problem-solving knowledge as reasoning rules, but that is just for descriptive convenience. This knowledge may be encoded in different implementations in different ways.

The notion of an *inference structure* as developed in the KADS methodology and summarized in (Aben 1995) begins to capture the notion of problem solving ontology. To take an example from (Coelho and Lapalme 1996), *Select_parameter* is an inference type that takes as input two types of knowledge: information regarding *Input_parameters* and *Parameter_dependency_relations* and produces an output of the type *Selected_parameter.* In the KADS literature, the information types are called *knowledge roles*, to capture the idea that they can be filled by domain knowledge. Earlier, in our discussion of domain knowledge we made a distinction between general domain knowledge and problem instance-specific knowledge. General domain knowledge will include knowledge about *Parameter_dependency_relations.* The problem instance database will contain information about the values of *Input_parameters* and *Selected_parameter* for the problem instance. Developing a problem solving ontology is thus identifying types of inferences and types of entities and relations that can fill the structure of the inference.

What is in problem-solving knowledge?

The problem-solver creates and changes a number of internal objects during the process of problem-solving. The problem state vector is the set of state variables representing these internal objects. In the case of diagnosis, examples of problem state variables are: *diagnostic hypotheses, observations explained by hypothesis H, the best hypotheses so far.* In the case of

design, examples of problem state variables are: *partial design, design candidate, specifications satisfied by the design candidate, best candidate so far.*

Active problem-solving goals and subgoals are part of the problem state description. All problems start with the specification of a problem-solving goal. The goal is usually stated as a variable of interest whose value is unknown. When the problem is solved, this variable is assigned a value. As problem-solving proceeds intermediate goals might be created and assigned values.

Following usage in the field, we will use the term *task* to describe a generic family of goals, such as diagnosis or design. For example, the diagnostic task is identifying a relation of the following type: <malfunction M> caused <abnormal observations {...}>. The prediction task is to construct a relation of the following type: <event | action A> will cause <state S> at instant t. The design task is to construct an object of the type <Connection-relation C {object 1, ...On}> such that the constructed object has certain required properties. In all these cases, the problem-solver's goal is to construct an internal object with certain properties and with components which relate to the domain knowledge elements in certain ways. In the case of diagnosis, the goal object is called the *diagnostic conclusion.* In the case of design, the goal object is called the *design solution.*

The basic unit of problem-solving knowledge may be considered to be composed of reasoning steps (or rules) of the following form:

<conditions on problem state>,
<conditions on domain knowledge>,
<conditions on data describing the problem instance>
→
changes to < problem state >

For example, a piece of diagnostic problem-solving knowledge might be[1]:

> "If the problem state includes the goal *Evaluate hypothesis H,* and if domain knowledge indicates that *H* can be evaluated as *confirmed* if the observations O_1, ..., O_n have the values v_1, ..., v_n respectively, and if O_1, ..., O_n do have values v_1, ..., v_n in the data describing the problem instance, then evaluate *H* as *confirmed* ."

A primitive (sub)goal is one that can be evaluated immediately by a reasoning rule using the domain knowledge in the knowledge base. Generally this is possible only if the problem state description enables the system to evaluate immediately the <conditions on the problem-solving state> part of the reasoning rule. Subtasks which are not primitive would need to be solved by the application of several reasoning steps.

The problem solving steps or inferences can be formally described at the Knowledge Level by indicating what the step does. Coelho and Lapalme propose the following notation:

```
(define-inference <inference-name>
  :cond <KIF-sentence>
  :body <inference-function>
  :result <KIF-sentence>)
```

For example, they describe the *Select_parameter* inference (which is part of a problem-solving strategy called "Propose and Revise"):

```
(define-inference Select_parameter
  :cond (parameter ?p1 exists and
  has no value, and ?p1 depends on
  another parameter ?p2 which
  exists and has a value)
  :body (Select ?p1)
  :result (Selected_parameter ?p1))
```

In our notation, this will be described as the reasoning step:

> If problem state includes a parameter ?p1 which has not been assigned a value and a parameter ?p2 which has been assigned a value, and if there is a problem solving subgoal to assign a value to it, and if the domain knowledge has a constraint relating the values of ?p1 and ?p2, then change the problem state such that ?p1 has the appropriate assigned value and the problem solving subgoal is removed.

The difference in our formulation is that we separate the domain factual knowledge from other aspects of the reasoning step which depend on the internal state of the problem solver.

Problem-Solving Method. There is another ontology element that is important in describing problem-solving knowledge. A *problem-solving method* is associated with a task. A method is a collection of subtasks plus appropriate control information about invoking the subtasks (Chandrasekaran 1990) plus specification of any passing of information between the subtasks. If the method's subtasks are achieved, the task with which the method is associated is achieved. For example, a common method for diagnosis is to decompose the diagnostic task into the following subtasks: *generate plausible diagnostic hypotheses*, and *compose hypotheses so as to give the best explanation.* A number of different control strategies for moving between generation and composition are possible (Josephson and Josephson 1994). A method is thus a

[1] Our examples of diagnostic problem-solving knowledge and ontologies are chosen mainly for clarity in making the conceptual points rather than for accuracy or completeness in describing diagnostic knowledge. In particular, the rule for establishing hypotheses is usually substantially more complex than the example.

bundling of subtasks, some of which may be primitive — that is, achieved immediately by a reasoning step —, while others may have further reasoning steps or methods associated with them.

Let us consider a particular subtask of diagnosis, e.g., *diagnostic hypothesis generation*, to illustrate what is meant by the control knowledge associated with a method. A common method for this task is *hierarchical classification* using a malfunction hierarchy. This method requires that the malfunction hierarchy be navigated top down, refining a malfunction hypothesis if it is judged to be likely present, and rejecting the malfunction and all its refinements if it is judged to be likely absent. When it is judged to be likely present, additional determinations of its degree of plausibility and the observations it explains are also to be made. This method organizes the invocation of the following subtasks (i) identify the malfunction hierarchy, (ii) *for a given malfunction in the hierarchy, determine if that malfunction is present*, if it is likely present, determining (iii) *determine the plausibility of a malfunction hypothesis and the abnormal observations it explains.*

Changing the example from diagnosis to design, a common method for generating candidate designs is the use of so-called *design plans*. Plans are, again, an organized collection of design subtasks. Plans additionally have control knowledge involving the order in which to invoke the design subtasks, and they also describe how the results of the subtasks affect the problem specification for other subtasks. Design plans are thus problem-solving methods that are common in the design task.

Control Vocabulary. Control determines the order of invocation of subtasks in a method. What is the vocabulary for control? In particular, how task-specific is the control vocabulary? Control may be explicitly specified using the standard vocabulary of control: sequential control, conditional branching, iteration, recursion, etc., whatever is appropriate for the underlying computational architecture. There appears to be no task-specific vocabulary for control. Control may also emerge from an interaction of domain knowledge and the problem state. For example, in CSRL (Bylander and Mittal 1986), hierarchical navigation over a hierarchy is explicitly programmed, i.e., the entire hierarchy is explicitly declared and the order of navigation is explicitly specified. In the Soar implementation of classification by Johnson (Johnson 1991), on the other hand, the control is specified as simply, "consider the successor of concept just established." Information about successors of any concept can be added modularly. At run time, the Soar-based classifier will string together the appropriate sequence of concepts to consider. Complex control behaviors may emerge as a result of the interaction between the architecture and the contents of the knowledge base.

Task satisfaction conditions. Associated with every task is knowledge that describes the satisfaction conditions for the task. Such a piece of knowledge for diagnosis, i.e., for accepting a hypothesis as the diagnostic conclusion, might be:

> "If a diagnostic hypothesis H explains the abnormal observations better than other diagnostic hypotheses, then H is to be accepted as the diagnostic conclusion."

Generally, the initial problem state is unlikely to contain all the diagnostic hypotheses and information about what each of them explains. Thus, for most diagnostic problem instances, the components of the reasoning rule describing the satisfaction conditions typically set up reasoning subtasks. Because of this diagnosis typically a non-primitive task. The subtasks set up by the above reasoning rule are: *determine the diagnostic hypotheses*, determine *the abnormal observations each hypothesis explains*, and *determine the measure of goodness of the explanation provided by each hypothesis.* For each subtask, there is a similar need to provide the satisfaction conditions.

The notion of problem solving methods as separate items of knowledge that can be instantiated with domain knowledge and reused originated with (Chandrasekaran 1985; Clancey 1985) and later extended by (Musen 1988; Steels 1990), and, in the KADS project, by (Benjamins 1993). The literature is quite extensive by now on problem solving methods and we will not attempt to be comprehensive in discussing it.

Ontology for Problem-solving Knowledge

While, as we said, the idea of reusable problem solving methods is by now quite common in the literature on knowledge-based systems, ontologies for problem solving knowledge have not gotten much attention. Musen and co-authors [Musen, et al, 1995] explicitly mention method ontologies as terms that describe the knowledge roles used by a PSM. Coelho and Lapalme (Coelho and Lapalme 1996) have recently attempted to formalize inference ontologies. Benjamins and associates (see for example, (Fensel and Benjamins 1996)) have also been looking into the structure of method representation. They use a representational elements called "assumptions" to link the parts of a method to the kinds of knowledge that they expect. A method ontology should make clear the ontological commitments of a method.

Let us review what we have done so far. We have identified the following kinds of entities that are required for representing problem solving knowledge: goals (and tasks), other components of the problem solving state, reasoning rules, and methods. Just as ontologies in the

domain objective knowledge dimension were types of predicates, relations, states, events and processes, the ontologies for the problem solving dimension will be types of goals, types of other problem solving state components, reasoning rules and methods.

To repeat from the examples that we considered during the development of the ideas, we can see some example ontologies for various tasks.

Task: *Diagnosis*

Components of the problem solving states: *diagnostic hypotheses, compound hypothesis, observations explained by a hypothesis, plausibility of a hypothesis, ...*

Methods: *Hierarchical classification, abductive assembly, ...*

Subtasks: *Establish hypothesis, refine hypothesis, .*

Task: *Design*

Components of the problem solving states: *design candidate, component design, expected behavior of design candidate, specification met by design candidate, ..*

Methods: *Design plan, plan refinement, qualitative simulation of design candidate, ..*

Subtasks: *Evaluate design, modify design, ..*

Neither the tasks, nor the ontologies listed above for the tasks, are complete. The main point here is to communicate the idea of problem solving ontologies as constructions parallel to domain factual knowledge ontologies.

Note that the required ontology is closely connected to the task. The problem-solving knowledge for the diagnostic task typically has a somewhat different ontology than the one for design. We say somewhat, since, both tasks can share some subtasks. For example, both can use simulation, diagnosis to decide on what observations may result from a malfunction, and design to verify if the design specifications are satisfied.

The dependence of a method on a task comes in two ways. First of all, clearly, associated with a method is an indication of what the function of the method is. Without it, we would not know what the method does, or what it is good for. Many method descriptions involve a method in another way as well, in the way its knowledge roles are specific to some other task. Let us consider an example. Classification is a method which is often used for the subtask of generating diagnostic hypotheses in the diagnostic task. One way to represent this task is to specify that the knowledge type for the classification hierarchy is the hierarchy of malfunctions, and that the knowledge type for the observations is the behavioral symptom of the object under diagnosis. However, this makes classification only useful as a subtask in diagnosis. As we know, however, it can be useful as a method for any problem involving a selection from a set of hierarchically organized selection choices. In order to make the classification method more generally useful, in the method ontology for classification, we may indicate that the knowledge type for the classification hierarchy is the set of hierarchically organized selection choices, and the knowledge type for observation is any observable feature of the problem. This increases the reusability of the method, but at the cost of some additional binding at run time: for the diagnostic problem, the set of hierarchically organized selection choices needs to be identified with the malfunction hierarchy and the observations with the behavioral features of the object under diagnosis. Beys, et al, (Beys, Benjamins et al. 1996) have investigated the representation of what they call *task-neutral methods.*

Task Structures Provide Problem-solving Knowledge Ontology

For the last decade or so, an area of research within knowledge-based systems has focused on the representation of problem-solving knowledge. In this stream of research, the major insight has been the focus on identifying generic tasks and studying the methods that are especially appropriate for them. Thus Clancey's Heuristic Classification (Clancey 1985) and Chandrasekaran's Generic Tasks (Chandrasekaran 1986) identified a number of such generically useful problem-solving tasks and particularly appropriate problem-solving methods for them. Heuristic Classification was a method with the subtasks of *data abstraction, heuristic match,* and *class refinement.* The Generic Task paradigm identified *hierarchical classification, abductive assembly, hypothesis assessment, design-plan selection and refinement,* and *data abstraction* as some of the most ubiquitous tasks in knowledge systems. This framework also proposed how complex problems might be solved by the composition of several different generic tasks. For example, a diagnostic system might be built out of the methods for abductive assembly, classification, hypothesis assessment and data abstraction. This architecture is really for the generic problem of best-explanation finding, a task discussed in detail in (Josephson and Josephson 1994) – this task is very important, since perception, natural language understanding, diagnostic problem-solving and scientific discovery can all be viewed as instances of best-explanation finding.

In later work, instead of identifying a unique preferred method with a task, Chandrasekaran developed the notion of a task structure (Chandrasekaran 1990). The task structure identifies a number of alternative methods for a task. Each of the methods sets up subtasks in its turn. This kind of task-method-subtask analysis can be carried on to a level of detail until the tasks are primitive tasks with

respect to the knowledge in the knowledge base.

This is not the place to give the details of the task analysis. The main points to be made here are the following. As a result of the GT and Task Structure work, we now have a good repertoire of tasks and methods. The descriptions of the tasks and methods is a rich source of ontologies for problem solving. The examples we gave in the earlier section for diagnosis and design are but a small subset of the ontologies that can be constructed for problem solving knowledge from the work on GTs and Task Structures. The fact that the GT and Task Structure work focuses on tasks of certain generality makes the ontologies that arise from them of potential general interest as well.

The earlier generation of Generic Task languages can be viewed in the light of knowledge reuse. To take a simple example, a Generic Task language called CSRL (Bylander and Mittal 1986) was widely used to build classification problem-solving systems. CSRL can be viewed as giving the user an ability to:

1. synthesize a classification method using a method-specific ontology consisting of terms such as "establish concept" and "refine concept," within a control vocabulary that allowed variations on top-down navigation of the classification hierarchy, and
2. represent domain factual knowledge for classification in the chosen domain.

Thus, the method ontology for classification directly resulted in a number of system builders reusing the problem-solving knowledge for classification embedded in CSRL. The Protégé family of planning tolls of Musen and his associates has a similar connection to the method ontology idea we have been discussing in this paper.

Concluding Remarks

We have argued for ontology engineering efforts to move in two parallel tracks, one with the current focus on representing domain knowledge, and the other a new focus on representing problem-solving methods. One of the major benefits of identifying and standardizing ontologies is the potential for knowledge sharing. We should be able to share knowledge of problem solving just as easily as factual knowledge about specific domains.

We have provided in this paper what we believe is a clearer analysis of the components of the problem solving knowledge and related it to domain factual knowledge ontology. We thus build on previous work in this area, some our own.

An important consideration in reusability of ontologies, whether of domain factual knowledge or of problem solving knowledge, is the challenge to reusability made by the Situated Cognition (SC) movement (Winograd and Flores 1987; Menzies 1996). The relevance of SC for us is the idea that the way in which knowledge is needed is highly dependent on the situation. For one situation, "A causes B" might be the relevant knowledge, while for another, "A causes B after a delay of 1 sec" is the relevant knowledge, while for yet a third situation, "A causes B as long as D and E are not true" becomes the appropriate form in which the knowledge is needed. Note that we are talking about the same knowledge, but represented in different ways for different situations. SC proponents hold that these kinds of additional conditions may be added indefinitely, thus making the prospect of representing the essence of that item of knowledge, once and for all at the Knowledge Level, rather slim. Similar comments may be made about the representation of problem solving methods as well.

In our view, this issue needs to be approached empirically. For one thing, in the above examples, what changes radically from situation to situation is not the ontology itself, but assertions made using the ontology. Thus, it is very likely that in general the rate of changes needed in ontologies from situation to situation is likely to be mush less than the changes needed to a specific knowledge base. We feel that reuse issues should be investigated by varying the situations and task specifications around a starting situation in such a way that we can track the needed changes to ontology and the knowledge base. This also implies that, in order truly to investigate reuse, we need systems that support the changes needed for knowledge and method ontologies as situations are changed. This appears to be an important new direction of research.

Acknowledgment

This material is based upon work supported by The Office of Naval Research, grant number N00014-96-1-0701, DARPA order number D594.

References

Aben, M. (1995). Formal Methods in Knowledge Engineering. *SWI*, University of Amsterdam.

Benjamins, R. (1993). Problem-Solving Methods for Diagnosis. *SWI*, University of Amsterdam.

Beys, P., R. Benjamins, et al. (1996). *Remedying the Reusability - Usability Tradeoff for Problem-Solving Methods*. 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta, Canada, Department of Computer Science, University of Calgary.

Bylander, T. and S. Mittal (1986). CSRL: A language for classificatory problem solving and uncertainty

handling. *AI Magazine*. **7:** 66-67.

Chandrasekaran, B. (1985). Generic Tasks in Expert System Design and Their Role In Explanation of Problem solving. *Proceedings of the National Academy of Science/Office of Naval Research Workshop on AI and Distributed Problem Solving, National Academy of Sciences*. Washington, DC.

Chandrasekaran, B. (1986). Generic Tasks in Knowledge-Based Reasoning: High-level building blocks for expert system design. *IEEE Expert*. **1:** 23-30.

Chandrasekaran, B. (1990). Design Problem Solving: A task analysis. *AI Magazine*. **11:** 59-71.

Chandrasekaran, B. (1994). "AI, knowledge and the quest for smart systems." *IEEE Expert* **9**(6): 2-6.

Clancey, W. J. (1985). Heuristic Classification. *Artificial Intelligence*. **27:** 289-350.

Coelho, E. and G. Lapalme (1996). *Describing reusable problem-solving methods with a method ontology*. 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta, Canada, Department of Computer Science, University of Calgary.

Fensel, D. and R. Benjamins (1996). *Assumptions in model-based diagnosis*. 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta, Canada, Department of Computer Science, University of Calgary.

Gruber, T. (1992). Ontolingua: A mechanism to support portable ontologies. Stanford, CA, Stanford University Knowledge Systems Laboratory.

Gruber, T. (1993). "A translation approach to portable ontology specifications." *Knowledge Acquisition* **2**(5): 199:220.

Johnson, T. R. (1991). Generic Tasks in the Problem-Space Paradigm: Building flexible knowledge systems while using task-level constraints, The Ohio State University. Ph. D. thesis, Department of Computer and Information Science.

Josephson, J. R. and S. G. Josephson, Eds. (1994). *Abductive Inference: Computation, Philosophy, Technology*, Cambridge University Press.

Lenat, D. B. and R. Guha (1988). The World According to Cyc. Austin, TX, MCC.

Marr, D. (1982). *Vision*. San Francisco, CA, W. H. Freeman.

McCarthy, J. and P. Hayes, J. (1969). "Some philosophical problems from the standpoint of artificial intelligence." *Machine Intelligence* **6**: 133-153.

Menzies, T. (1996). *Assessing responses to Situated Cognition*. 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta, Canada, Department of Computer Science, University of Calgary.

Musen, M. A. (1988). Generation of Model-Based Knowledge-Acquisition Tools for Clinical-Trial Advice Systems, Stanford University.

Musen, M. A., Gennari, J. H., Eriksson, H., Tu, S. W., Puerta, A. R. (1995). "PROTEGE-II: Computer support for development of intelligent systems from libraries of components," Proceedings of MEDINFO '95, Eighth World Congress on Medical Informatics, 766:770.

Neches, R., R. Fikes, et al. (1991). "Enabling technology for knowledge-sharing." *AI Magazine* **12**(3): 16:36.

Newell, A. (1981). The Knowledge Level. *AI Magazine*. Summer: 1-19.

Steels, L. (1990). Components of Expertise. *AI Magazine*. **11:** 28-49.

Wielinga, B. J., A. T. Schreiber, et al. (1992). "KADS: A modeling approach to knowledge engineering." *Knowledge Acquisition* **4**: 5-53.

Winograd, T. and F. Flores (1987). "On understanding computers and cognition: A new foundation for design: A response to the Reviews." *Artificial Intelligence* **31**: 250-261.