Ontology of Tasks and Methods*

B. Chandrasekaran[1], J. R. Josephson[1] and V. Richard Benjamins[2]

[1]Laboratory for AI Research, The Ohio State University, Columbus, OH 43210,
chandra,jj@cis.ohio-state.edu, http://www.cis.ohio-state.edu/lair/

[2]Dept. of Social Science Informatics (SWI), University of Amsterdam, Roetersstraat 15,
1018 WB Amsterdam, The Netherlands, richard@swi.psy.uva.nl,
http://www.swi.psy.uva.nl/usr/richard/home.html

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 has a goal of sharable problem-solving methods. For example, when one is dealing with *abductive inference* problems, the following are some of the terms that occur in the representation of problem-solving methods: *hypotheses, explanatory coverage, evidence, likelihood, plausibility, composite hypothesis*, etc. Method ontology is, in good part, goal- and method-specific. ``Generic Tasks," ``Heuristic Classification," ``Task-specific Architectures," ``Task-method Structures," ``Inference Structures" and ``Task Structures" are representative bodies of work in the knowledge-systems area that have focused on domain-independent problem-solving methods. However, connections have not been made to work that is explicitly concerned with domain ontologies. Making such connections is the goal of this paper. This paper is part review and part synthesis.

1 Ontologies as Content Theories

In philosophy, ontology is the study of the kinds of things that exist. Ontologies are often said, colorfully, to "carve the world at its joints." In AI, the term has largely come to mean one of two related things.

- A representation vocabulary, typically specialized to some domain or subject matter. More precisely, it is not the vocabulary as such that qualifies as an ontology, but the

* Earlier versions of this paper have been presented at the 1997 AAAI Spring Symposium and the1998 Banff Knowledge Acquisition Workshop. The current version has been substantially expanded.

conceptualizations that the terms in the vocabulary are intended to capture. For example, the ontology doesn't change by translating the terms from pseudo-English to pseudo-French. In engineering design, one might talk about the ontology of the domain of electronic devices. Such an ontology might have conceptual elements such as ``transistors," ``operational amplifiers," ``voltages," and so on, and relations between these elements, such as one class of devices is a subtype or a part of another, or that certain terms are properties of certain devices. Identifying such terms --and the underlying conceptualizations-- generally requires careful analysis of the kinds of objects and relations that can exist in the domain. In what has come to be called ``Upper Ontologies," i.e., ontologies that describe generic knowledge that holds across many fields, the analysis required to establish the ontologies is a major research challenge.

- Occasionally, a body of knowledge describing some domain, typically a common sense knowledge domain, using such a representation vocabulary. For example, CYC (*Lenat & Guha, 1990*) often refers to its knowledge representation of some area of knowledge as 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. Our goal is both to review the work on ontology of problem solving knowledge and to propose a framework in which new work may be carried on more productively.

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 mechanism, such as rule systems, frame languages, neural nets, fuzzy logic, constraint propagation, unification, etc. The mechanisms are proposed as the secret of making intelligent machines. At other times, there is a realization that, however wonderful 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 equally well 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 & 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, 1982*) all grapple in their own ways with characterizing content. Ontologies are quintessentially content theories.

1.1 Why Are Ontologies Important?

Ontological analysis clarifies the structure of knowledge. The first reason is that they form the heart of any system of knowledge representation. If we do not have the conceptualizations that underlie knowledge, then we do not have a vocabulary for representing knowledge. Thus the first step in knowledge representation is performing an effective ontological analysis of some field of knowledge. Weak analyses lead to incoherent knowledge bases. A good example of the need for

good analysis comes from the field of databases (*Wieringa & de Jonge , 1995*). Consider a domain in which there are people, some of whom are students, some professors, some are other type of employees, some are females and some males. For quite some time, a simple ontology was used in which the classes of students, employees, professors, males and females were represented as ``types of'' humans. Soon this caused problems because it was noted that students could also be employees at times and can also stop being students. Further ontological analysis showed that ``students,'' ``employees,'' etc. are not ``types-of'' humans, but rather they are ``roles'' that humans can play, unlike categories such as ``females,'' which are in fact a ``type-of'' humans. Clarifying the ontology of this data domain made it possible to avoid various difficulties in reasoning about the data. Analysis of this sort that reveals the subtle connections between terms can often be quite challenging to perform.

Ontologies enable knowledge sharing. The second reason why ontologies are important is that they provide a means for sharing knowledge. We just described how demanding it can be to come up with the appropriate conceptualizations for representing some area of knowledge. Suppose we do such an analysis and arrive at a satisfactory set of conceptualizations, and terms standing for them, for some area of knowledge, say, the domain of ``electronic devices.'' The resulting ontology would likely include terms such as ``transistors,'' and ``diodes,'' and more general terms such as ``functions,'' ``causal processes,'' ``modes'', and also terms in the electrical domain, such as "voltage," that would be necessary to represent the behavior of these devices. It is important to note that the ontology – defined by the basic concepts involved and their relations – is intrinsic to the domain, apart from a choice of vocabulary to represent it. If we can come up with a set of terms to stand for the conceptualizations, and a syntax for encoding the conceptualizations and the relations among them, then the efforts that went into analysis can be encoded into an ontology. This ontology can be shared with others who have similar needs for knowledge representation in that domain, avoiding the need for replicating the knowledge analysis. These ontologies can form the basis for domain-specific knowledge representation languages. In contrast to the previous generation of knowledge-representation languages, such as KL-One, these domain-specific languages may be termed ``content-rich.'' That is, they have a large number of terms that embody a complex content theory of the domain.

Given such an ontology, specific knowledge bases describing specific situations can be built. For example, manufacturers of electronic devices could build catalogs that describe their products. With the shared vocabulary and syntax, such catalogs could be shared easily, and used in automated design systems. This kind of sharing vastly increases the potential for reuse of knowledge.

We will now briefly review the basics of work on ontology to set the stage for discussing a specific type of ontology, that of problem solving knowledge.

1.2 Terms for Describing the World

An ontology helps us to describe facts, beliefs, hypotheses, and predictions about the world in general, or in a limited domain, if that is what is needed. Constructing ontologies for representing factual knowledge is still an on-going research enterprise. Ontologies range in abstraction from very general terms that lie at the heart of our understanding and descriptive capabilities in all

domains, to terms that are restricted to specific domains of knowledge. Basic phenomena of *space, time, parts* and *subparts* apply to all domains, while the concept of *malfunction* applies to engineered or biological domains, and even more specifically, the concept of *hepatitis* applies to medicine. These examples also suggest that there is no sharp line of abstraction separating the general from the domain- specific. Domains come in differing degrees of generality. Ontologies required to describe knowledge of some domain may require, in addition to domain-specific terms, terms from higher levels of generality. Terms at very general levels of description are often said to be part of the so-called ``upper ontology." There are many open research issues about the correct ways to analyze knowledge at this level, and disagreements and open problems abound. To give some idea of the issues involved, here is a quote from a recent call for papers: (1)

> ``*On the one hand there are entities, such as processes and events, which have temporal parts... On the other hand there are entities, such as material objects, which are always present in their entirety at any time at which they exist at all. The categorical distinction between entities which do, and entities which do not have temporal parts is grounded in common sense. Yet various philosophers have been inclined to oppose it. Some ... have defended an ontology consisting exclusively of things with no temporal parts. Whiteheadians have favored ontologies including only temporally extended processes. Quine has endorsed a four-dimensional ontology in which the distinction between objects and processes vanishes and every entity comprises simply the content of some arbitrarily demarcated portion of space-time. One further option, embraced by philosophers such as David Lewis, accepts the opposition between objects and processes, while still finding a way to allow that all entities have both spatial and temporal parts.*''

Sowa (*Sowa, 1997*), CYC (*Lenat & Guha, 1990*) and Guarino (*Guarino, 1995*) are some researchers in AI who have proposed alternative upper ontologies. As a practical matter, there is agreement that there are *objects* in the world; these objects have *properties* that can take *values;* the objects may exist in various *relations* with each other; the properties and relations may change over *time;* there are *events* that occur at different *time instants;* there are *processes* in which objects participate and that occur over time; the world and its objects can be in different *states;* events may *cause* other events as *effects;* and objects may have *parts*. Further, perhaps not as basic facts of the world but as ways of organizing them, is the notion of *classes, instances,* and *subclasses*, where ``classhood" is associated with shared properties. Thus, *Is-A* relations indicating subclass relations are fundamental for ontology representations.

The representational repertoire of *objects, relations, states, events* and *processes* does not say anything about what classes of these entities exist. They are left as commitments to be made by the person modeling the domain of interest. Even at very general levels, such commitments already appear. Many ontologies agree to have as root the class ``thing" or ``entity", but already at the next more specific level, they start to diverge, a fact which is nicely illustrated by the slightly different taxonomies of the top levels in existing ontology projects such as CYC, Wordnet, Generalized Upper Model, Gensim, etc. (see *Fridman-Noy & Hafner, 1997* for an overview). The more specific the domain to be modeled, the more the ontological commitments. For example, someone, 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 relations that may hold between any two objects. These commitments are not arbitrary --any old declaration of classes and relations will not do. For them to be useful, such commitments should reflect some underlying reality, should reflect real existence. Thus the philosophical term ``ontology'' is especially appropriate for such commitments.

As mentioned, there is no sharp division between domain-independent and domain-specific ontologies for 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. Ontologies are terms that are needed to describe the world, but an ontology for representing domain facts can of course be used to represent non-facts, hopes, expectations, etc.

Two Levels of Ontology. Research on ontologies generally proceeds by asking the question, ``What is the ontology of P?'' where P is some type of entity, process or phenomenon. P may be something very general and ubiquitous such as ``causal processes,'' or ``liquids.'' Or P can be of narrower scope such as ``devices,'' or ``diseases of biological organisms.'' It is usual to identify two levels at which such an analysis is often conducted, and correspondingly, two levels of ontology for P can be distinguished (this distinction is reminiscent of the distinction between core and peripheral ontologies in (*van Heijst et al., 1997*)).

At the first level, one identifies the basic conceptualizations needed to talk about all instances of P. For example, the first level ontology of ``causal process'' would include such concepts as ``time instants,'' ``systems,'' ``system properties,'' ``system states,'' ``causes of change states,'' and ``effects (also states).'' We need this vocabulary to talk about causal. At the second level, one would identify and name different types of P, and relate the typology to additional constraints on, or types of, the concepts in the first-level ontology. For the causal process example, we might identify two types of causal processes, ``discrete causal processes,'' and ``continuous causal processes.'' These terms, and the corresponding conceptualizations, are also parts of the ontology of the phenomenon being analyzed. Second-level ontology is essentially open-ended: new types may be identified at any time.

How task-dependent are ontologies? What kinds of things actually exist does not depend on what our goals are. In that sense, ontologies are not task-dependent. On the other hand, exactly what aspects of reality in some domain get identified and written down in a particular ontology depends on what tasks the ontology is being built for. An ontology of the domain of *fruits* would focus on some aspects of reality if it is being written for selecting pesticides, and on different aspects if it is being written to help chefs select fruits for cooking. As we will see, assumptions or requirements of problem-solving methods can capture explicitly the way in which ontologies are task-dependent (*Fensel & Benjamins, 1996*). Assumptions are therefore a key-factor in practical sharing of ontologies.

Technology for Ontology Sharing. There have been several recent attempts at creating engineering frameworks in which to construct ontologies. Neches, et al. (*Neches et al., 1991)*

describe an enabling technology called KIF intended to facilitate expression of domain factual knowledge using a formalism based on augmented Predicate Calculus. A language called Ontolingua (*Gruber, 1993)* has been proposed to aid in the construction of portable ontologies. In Europe, the CommonKADS project has taken a similar approach to modeling domain knowledge (*Schreiber et al., 1994)*. These languages use various versions of Predicate Calculus as the basic formalism. Predicate Calculus facilitates the representation of objects, properties, relations, and classes. Variations such as Situational Calculus can be used to introduce time so that states, events and processes can be represented. If the idea of knowledge is extended to include images and other sense modalities, radically different kinds of representation may be needed. For now, PC provides a good starting point for ontology-sharing technologies.

Using a logical notation for writing and sharing ontologies does not imply any commitment to implementing the knowledge system in that or a related logic. 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?"

We are now ready to discuss the ontology of a specific phenomenon, that of problem-solving. We think that almost all the work on ontologies, until recently (*Fensel et al., 1997*, Mizoguchi *et al.*, 1995), has been focused on one dimension of knowledge content. In order to explain this claim, we will need to identify different dimensions to the study of ontologies. We turn next to this task.

1.3 Dimensions for Ontology Specification in Knowledge Systems

In building a problem-solver, we need two types of knowledge:

1. Domain factual knowledge: Knowledge about the objective realities in the domain of interest (Objects, relations, events, states, causal relations, etc.)

2. Problem-solving knowledge: Knowledge about how to achieve various goals. A piece of this knowledge might be in the form of a problem-solving method (PSM) specifying, in a domain-independent manner, how a class of goals can be accomplished.

Early research in KBS mixed together both factual and problem-solving knowledge into highly domain-specific rules and called all of it ``domain knowledge." As research progressed, however, it became clear that there were systematic regularities in how domain knowledge (we'll use ``domain knowledge" as a short hand for ``domain factual knowledge") was used for different goals. It also became clear that this knowledge could be abstracted and reused.

The domain knowledge dimension has been the overwhelming focus of most of the AI investigations on ontologies. In our view, the historical reasons for this are the following. In AI, there have been two distinct sets of practical applications of knowledge representation. One of them has been the area that started out as Expert Systems, but has evolved into a somewhat broader area called Knowledge-Based Problem-Solving (KBPS). Another area within AI that has also needed knowledge representation is natural language understanding (NLU). Ontological analysis is typically required to identify appropriate semantic structures for an understanding

program to map utterances to. Knowledge also plays a crucial role in disambiguation. Database and information systems communities have also recently begun to take interest in ontology issues. In practice, much of the work on ontologies, and in knowledge representation in general, has been driven more by concern with the needs of these communities, and these needs have mostly to do with the structure of what we have called factual knowledge. NLU and database systems typically do not do much problem solving of the sort done by KBPS systems.

KBPS systems on the other hand clearly need to be concerned with uses of the knowledge for complex chains of reasoning. Thus the field of KBPS came to a realization in quick order that in addition to factual knowledge there is also knowledge about how to use the knowledge to achieve problem-solving goals. In fact, the so-called "second generation" research in knowledge systems was fueled by this emphasis on methods appropriate for different types of problems. Most of the work on knowledge representation that has gone on within the KBPS community is not even known to the general knowledge-representation community.

The dimension of representing problem-solving knowledge will be the focus of this paper. We will begin by analyzing the ontology of a problem-solver, and note the role of problem-solving knowledge in it. We will ask: What is problem-solving knowledge made of? What are methods? What specific methods are there for what kinds of problems? What is the relationship between method ontologies and factual-knowledge ontologies? We will also discuss sharing and reusing method knowledge, since these are the major motivations of this line of research.

2 Ontology of a Problem Solver

The major elements of a first-level ontology of a generic instance of problem solver are the following:

1. A problem-solving goal
2. Domain data describing the problem-instance
3. Problem-solving state
4. Problem-solving knowledge (PSK)
5. Domain factual knowledge (DFK)

The underlying picture is that the problem-solver's state changes as a result of applying problem solving knowledge and domain knowledge to problem data. Eventually, a part of its state description might contain a solution to the goal, at which point the problem solving process stops. It might also stop when, given its knowledge and data, no progress toward the goal is possible. This description most likely does not suffice for all forms of what one might consider problem-solving. For example, distributed problem solving, where a number of independent agents collectively solve problems might require extensions to our description. However, for the points we wish to make, this characterization is a good starting place.

2.1 Problem-Solving Goals

A simple example of a problem-solving goal is a diagnostic goal, "*Explain abnormal observations of a system."* This goal description is composed of two parts, an "attitude" term (*Explain*) that takes as argument an external world description (*abnormal observations*). This applies to goal descriptions in general. They are made up of attitude terms about world descriptions. Thus, the first-level ontology for goals is simply "attitudes on world descriptions." Of course, fully a describing a goal instance would require using attitude terms as well as the terms needed for their arguments, i.e., the DFK ontology that enables one to describe world states, events, object configurations, and process descriptions.

We have mentioned that second level ontology terms arise by specializing first level ontology terms and identifying appropriate constraints between them. "Diagnostic goal" is a second-level term in the goal ontology; it is obtained by specializing the attitude term to "Explain" and constraining its argument to be world descriptions of the form "abnormal observations." Other attitude types are: ``make true that," ``prevent," "determine whether," "identify," and ``assign likelihood to."

Planning systems have goals to generate actions needed to achieve certain desired world states or avoid undesired ones. Design systems have goals of synthesizing object configurations that will behave in desired ways. Prediction or simulation systems have the goal of predicting future world states, and so on. Thus *diagnose, design, plan, predict*, etc., are some of the common terms in the second-level ontology for problem-solving goals. *Classification* goals are very common in knowledge systems as well. Many of these second-level elements of the goal ontology were identified during the research on task-oriented approaches. In fact, the term ``task" was used to refer to goal types of certain generality (2). Tasks come in a range of generality: ``diagnose medical problems," is more specific than ``diagnose systems," but is more general than ``diagnose liver illnesses." ``Explain observations," is more general than ``diagnose." ``Explain observations" can cover many other task types besides diagnosis.

2.2 Data Describing the Problem Instance

A problem instance is also described in terms of domain factual ontology. In diagnosis, it is a set of observations. In prediction, it is description of actions and conditions that obtain in some world. In design, it is a set of constraints and specifications. A problem instance for a logistic planner might be a set of specific supply items to be delivered to specific locations and under specific weather, transportation and storage availability conditions.

The ontology of problem instance data is the same as that of domain factual knowledge. The second-level ontology for problem instance data parallels that of the goal. Data for diagnostic goals are variously called *normal* and *abnormal observations*, and *symptoms*; for design goals, they are *specifications, constraints*, and *functions*; for prediction goals, they are *initial conditions*, and *actions*, and so on.

2.3 Problem States

The problem-solver creates and changes a number of internal objects during the process of problem solving. A problem state is a set of values of state variables representing these internal objects. Problem state includes information about current goals and subgoals. It would also include all knowledge inferred during problem solving: e.g., elements of candidate solutions, their plausibility values, rejected solutions and reasons for them. In the case of diagnosis, problem state would contain information such as: *current diagnostic hypotheses, observations explained by hypothesis H, the best hypotheses so far*. In the case of design, problem state would contain information such as: *partial design, design candidate, specifications satisfied by the design candidate, best candidate so far*. Thus task types determine types of problem state variables. Active problem-solving goals and subgoals constitute a distinguished part of the problem state description. As problem solving proceeds, some of the subgoals are achieved, new subgoals are created, and some goals are abandoned.

2.4 Problem-Solving Knowledge

The basic unit of problem solving knowledge (PSK) is a mapping of the following form:

<conditions on the problem state (including goals)>
<conditions on domain knowledge>
<conditions on data describing the problem instance>
–>
changes to <problem state (including goal components)>

The above is not intended to be seen as a rule (which is an implementation formalism in KBS), but as a Knowledge Level description of a basic unit of problem- solving knowledge. It describes what the first-level ontology of a reasoning step is. It says that problem-solving behavior is responsive to the current state of problem solving (including goals), and uses domain knowledge and problem data to make changes to problem state, achieve goals and to set up subgoals, and to obtain additional data. For example, a piece of diagnostic problem-solving knowledge might be (3):

> ``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*."

Problem-solving knowledge may be indexed by the goal it serves (in the example, *Evaluate hypothesis*). This facilitates sharing of this type of knowledge. It can be applied to any domain in which we need to assess hypotheses as long as we have the domain knowledge corresponding to the ``observations –> hypothesis" part in the reasoning step. This is a key part of our analysis: the above is not a rule such as a rule of Mycin, in which domain factual knowledge and problem-solving knowledge were combined. It is an abstract description of a piece of problem-solving knowledge.

Problem Solver Element	First Level Ontology	Second Level Ontology
Problem Solving Goal	Attitude terms whose arguments are world descriptions (see Domain Factual Knowledge (DFK) below)	*Explain, determine whether, identify, classify, construct* Diagnosis Example*:* *Explain abnormal observations in terms of malfunction causes*
Problem instance data	DFK terms (see below)	Diagnosis Example*:* *Sensor values, abnormal values,*.:
Problem state components	Solution candidates, parts of solutions, goal, subgoals, solution valuation, satisfied and unsatisfied problem requirements,..	Diagnosis Example*:* *Malfunction hypotheses:* solution candidates, parts of solution *Confirmed |rejected hypotheses, plausibility of hypotheses*: solution valuation *Explained and unexplained observations*: satisfied and unsatisfied problem requirements *Evaluate malfunction hypothesis*: subgoal
Problem Solving Knowledge	Conditions on (sub)goal, state, problem instance data, DFK, changes to state	Ontology of goal, state, and data given above. Change of state in the form: *change hypothesis evaluations, set up explanatory, data seeking, and parsimony* subgoals
Domain Factual Knowledge	Objects, properties, states, processes, classes,..	*Devices, functions, malfunctions, modes, causes and effects, observations, behavior, components, …*

Table 1. Ontology of a problem solver and its elements. The example of diagnosis is used to illustrate second-level ontology.

PSK units may come at different degrees of abstraction, based on how abstract the indexing goals are. The goal may range in abstractions: for example, from ``Establish hypothesis," through ``Establish diagnostic hypothesis," to ``Establish device malfunction," to ``Establish diode failure." Actual implementations in specific domains may combine problem-solving knowledge and domain knowledge, the way early rule-based systems did. For example, a system may have a rule of the form:

``If the goal is to establish diode malfunction, then if voltage $v_i = 0$, then confirm diode malfunction."

This unit of problem-solving knowledge is harder to share across domains, even though for this specific domain it may have exactly the same result as the application of the more abstract PSK unit.

2.5 Domain Knowledge

We have already discussed the issues in describing factual knowledge in the world: Objects, properties, relations, classes and subclasses, states, processes, events, and parts are some of the elements in that ontology. We have also indicated that the ontology for domain knowledge is determined by the needs of the goal and the problem-solving knowledge. A second level ontology for diagnostic systems would contain terms such as *device, event, component*, and *component connection, function, malfunction, symptom*, and *normal/abnormal behavior*, and relational knowledge of the sort *Can-cause(malfunction, {observation | malfunction})*. The domain knowledge might also have knowledge of the causal processes that realize the function of a device. Similar analyses can be made for other tasks. The second-level ontology for domain knowledge for design and diagnosis has much in common: the concepts of devices, components and causal relations.

Table 1 is a summary description of the elements of a problem solver and their ontology.

2.6 Relations between Different Elements of the Problem-Solving Ontology

Elements of a problem-solver use domain knowledge of specific types, and place mutual ontological constraints. McDermott coined the term *knowledge roles* to refer to how problem-solving knowledge required domain knowledge of certain types. CommonKADS work --and subsequent work built on it (*Schreiber et al., 1994), (Coelho & Lapalme, 1996*)-- concerning *inference structures* tries to make explicit the relation between different elements of a problem solving knowledge.

Let us take an example used by Coelho and Lapalme : *Select_Parameter* task (which, to look ahead somewhat, is a subtask in the *Propose-and-Revise* method). *Select* is an attitude term. Parameters are properties of some objects in the domain. Using our notation, this unit of problem-solving knowledge may be described as:

> **(Sub)Goal**: *Select value for parameter ?p1*
> **Condition on Problem Instance Data**: *value of parameter ?p2*
> **Condition on Problem State**: problem state includes *a parameter ?p1* which has not been assigned a value
> **Condition on Domain Knowledge**: domain knowledge has a *constraint relating the values of ?p1 and ?p2*
> **Changes to Problem State**: change the problem state such that ?p1 has the value allowed by the constraint, and remove subgoal

It is this close relationship between goal types (tasks), problem-solving knowledge, and domain knowledge that makes possible the sharing of problem-solving knowledge. Problem-solving knowledge can be reused in a different domain and task by applying it to knowledge stated in the appropriate domain ontology. We'll discuss this in some detail later in the context of methods.

3 Problem-Solving Methods

A problem solving method is an organized package of PSK units, indexed by the problem-solving goal to which it is applicable. Why would one need to organize reasoning steps in the form of methods? Recall that a PSK unit may set up a subgoal, as a means to achieve the goal for which it is invoked. Suppose there is a PSK indexed with goal G, and this PSK sets up a subgoal, G1. Suppose there is another PSK for the goal type G1. One may then use the combination of the two PSK's as a packaged unit, index it by G and invoke it whenever G is encountered. A method may consist of just one PSK unit. In general, however, methods derive their value by their larger granularity, since a complex reasoning strategy may thus be reused.

In the example we just gave, the situation was quite simple: the PSK unit for G set up G1 and the two PSK units then helped achieve G. However, more complex situations will commonly arise. G may set up more than one subgoal, and a subgoal G1 may have more than one PSK unit available to solve it, depending on conditions on data and DFK. Thus a method may include alternate ways of accomplishing some of the subgoals. It also needs to include control knowledge to organize invocation of subgoals, and also knowledge about exchanging and integrating information between subgoals. Note that, just as PSK units, a method will either achieve the main goal, or set up one or more subgoals that will need to be solved by additional methods.

How general is the characterization of methods as compositions of problem-solving knowledge units, related by subgoal relations? Again, similar to our point about our model of the problem solver itself, the definition does not cover everything that would intuitively appear to be a method. Our definition of a method is intended to capture the range of methods that a universal subgoaling system such as SOAR (*Laird et al., 1987)* can accomplish. Our goal here is not to account for all methods, but to indicate what method ontology looks like.

First-Level Ontology for Methods. The first-level ontology of a PSM, then, is simply the ontology of PSK units plus control knowledge. The method ontology thus includes the goal (also referred to as competence), the goal-subgoal tree it induces, the forms in which it requires data and domain knowledge (i.e. assumptions and requirements of the method), and control knowledge for organizing the invocation of the subgoals (see also (*Fensel et al., 1997)*). We have discussed all the above elements earlier, except for control knowledge.

Control Ontology. Control may be explicitly specified using the standard vocabulary of control: sequential control, conditional branching, iteration, recursion, etc. There appears to be no task-specific vocabulary for control. Control flow of specific sorts may also emerge from an interaction of domain knowledge and the problem state. For example, in hierarchical classification, navigation over a hierarchy may be explicitly programmed. Or, one may simply have the knowledge, ``consider the refinement of current concept," as in (*Johnson, 1991*). If the domain knowledge has information about refinements for the concepts, then the resulting behavior will be hierarchical navigation, without this strategy being explicitly stated. Complex control behaviors may emerge as a result of the interaction between the architecture and the contents of the knowledge base.

Indexing of Methods. The method is indexed by its goal. Just as in the case of an individual PSK unit, the goals and subgoals of the methods can occur at different degrees of generality. If the goals are very abstract and general – say, ``Explain"– then the user of the method has the task of mapping it to his needs. If the goal is very specific – say, ``diagnose TC tuner circuit"– then the method's reusability is limited. This has come to be called the *usability-reusability trade-off* (*Klinker et al., 1991)*. There have been proposals in the literature to distinguish a class of methods called *task-neutral methods* (*Beys et al., 1996)* i.e., methods that are not intended for a specific goal type. The intent is to capture the idea that some methods have extremely general applicability, while others seem to be more narrowly tailored to tasks of great specificity. We think that there is no binary separation between task-specific and task-neutral methods. Instead, there is a spectrum within which we can talk about more general and more specific (*Fensel, 1997*) The point remains that the more general the task index of a method, then the more work that will be needed to recognize how to apply it to a specific task. We will revisit this issue later when we discuss the issues surrounding sharing and use of methods.

Operationalization of Methods. A method of the sort we have been discussing is *operationalized.* That is, the description of a method can be used directly to implement a problem solver. As long as domain knowledge and problem data of the types required by the method definition are available, the method should ``run."

However, the term ``method" has often been used in the literature to refer, not to operationalized methods, but to general approaches and strategies. For example, one often sees references to the ``divide and conquer" method; this method is so general, and potentially so ubiquitously applicable, that it is impossible a *priori* to operationalize it for all cases to which it is applicable. On the other hand, that method has indeed been successfully operationalized for specific domains, such as for composition of a certain class of algorithms (*Smith, 1991*).

Then there are methods, such as ``Propose-and-Revise," and ``Heuristic Classification," that are also quite general and may be applied to almost any problem in principle. These methods have been operationalized: if we can find domain knowledge to suit their requirements, they can be applied. Almost every problem in the world can in theory be solved by proposing an initial solution of the right sort and then critiquing and revising it. The difficulty comes in identifying the right sort of initial solution, and right sorts of criticisms and modifications for arbitrary problems. Similarly, any problem can in theory be solved by categorizing its solution space into classes and mapping from the problem statement into solution categories. If this were easy to do in practice, then we do not need any other methods at all. The rub comes when one tries to identify, for an arbitrary problem, knowledge corresponding to good initial solution, criticism knowledge, etc. (for the Propose-and-Revise method), or a categorization of the solution space and mapping from the problem statement to the solution category space (for the Classification method). Thus, in practice, these methods, though operationalizable in general, can only be usefully applied to a narrow range of problems. For example, Propose-and-Revise has been used to solve parametric design problems in domains where there is often clear and straightforward knowledge available about initial candidates and about how to assess and modify the parameters (*Schreiber & Birmingham, 1996*). Similarly, classification methods can be operationalized and applied to problems where knowledge about solution hierarchies and mappings from problem

data to solution hierarchy elements is not hard to get. Simple types of selection problems, and diagnostic problems where malfunction hierarchies are available, are examples of problems for which the Classification method has been readily applicable.

3.1 Second-Level Method Ontology

A second level ontology for methods would identify, characterize and name methods based on *how* they achieve their goals. How a method works is the goal-subgoal structure induced by all the PSK units in the method. This goal-subgoal structure can be modified in many ways: by replacing one of the PSK units with a new one for the same subgoal, or adding a new PSK unit for the subgoal, we get a different overall goal-subgoal structure. Each variation counts as a new method. Because of this, even for simple goal types, the number of distinct methods may be too numerous to list and name.

In spite of the proliferation problem, one might still identify especially useful combinations of goal-subgoal structures for various goal types, and make these methods available for sharing. This was the approach adopted in the much of the work on task-oriented approaches. Thus work on Heuristic Classification (*Clancey, 1985*), Generic Tasks (*Chandrasekaran, 1986*) and KADS' Interpretation Models (*Wielinga & Breuker, 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 & Josephson, 1994*). This task is very important, since perception, natural language understanding, diagnostic problem-solving and scientific discovery can all be viewed as best-explanation finding problems.

In related later work, instead of identifying a unique preferred method with a task, the notion of a task structure was developed (*Chandrasekaran, 1990*). The task structure identifies a number of alternative methods for a task. Each of the methods sets up subtasks in its turn. The methods are typically shallow in the nesting of subgoals, increasing the chances that a user would use them as a unit without much modification. This kind of task-method-subtask analysis can be carried on to a level of detail where the tasks are primitive tasks with respect to the knowledge in the knowledge base. The advantage of developing task structures for a task, as opposed to a specific method, is that there is greater flexibility in putting together a method that meets with the needs of the specific situation. Methods with shallow goal-subgoal trees have relatively small number of PSK units. Of course, these methods would leave a number of subgoals unsolved. The user is free to seek other methods for these subgoals, thus providing flexibility. For example, the method of logarithms for multiplying two numbers might simply say, ``Find logarithms of the two numbers, add them, and then find the antilogarithm of the sum," without specifying how to achieve the subgoals ``Find logarithm of number," and ``Find antilogarithm of number." The task

structure would identify alternate methods for each of these subtasks. The logarithm method is highly reusable, while giving the user freedom to use appropriate methods for the subgoals. Figure 1 illustrates task structure.

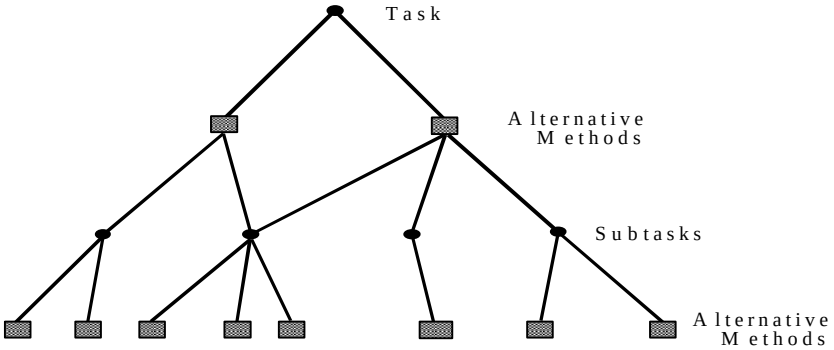

Figure 1. Task structure. A task may be solved by a number of alternative methods. Each method may set up subtasks.

Task structures have been developed for the task of design (*Chandrasekaran, 1990*), diagnosis (*Benjamins, 1993*), planning (*Barros, L. Nunes de et al., 1996*) and abductive inference (*Josephson & Josephson, 1994*). This is not the place to give the details of these task analyses. The main points to be made here are the following. As a result of the Generic Task and Task Structure work – and in general, of work on task analysis – we now have a good repertoire of tasks and methods. The descriptions of the tasks and methods are 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 task-oriented approaches. The fact that this 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 & Mittal, 1990*) 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.

Although the notion of problem-solving ontology as introduced in this paper is relatively new compared to the work on domain ontologies, some work has been performed in the field of knowledge engineering that provided important input. Already in the beginning of the nineties, in Japan Mizoguchi's group started to talk about task-ontologies as valuable instruments to link the vocabulary and view of a user to that of a problem solver (Mizoguchi *et al.)*. These task ontologies comprise the vocabulary and the reasoning steps of specific tasks and, in that sense, they relate to our notions of problem-solving goal and problem-solving knowledge

Work carried out at Stanford on Protégé also is relevant for problem-solving ontologies. A method ontology, in their view, defines the concepts and relationships that are used by the method to achieve its goal (*Gennari et al., 1994)*. Thus, "method ontology" in this research refers to domain ontology from a method point of view. In this sense, Protégé-II is closely related to what are called ontological assumptions or requirements of problem-solving methods (*Benjamins et al., 1996)*. Such assumptions may define ontological commitments of PSMs in a domain-independent way. However, assumptions not only refer to domain knowledge needed by the PSM, but also to the task a method is supposed to realize, in which case they are called teleological assumptions. Such assumptions define a weakening of the goal to be achieved by introducing assumptions about the precise problem type. For example, the use of the Classification method for diagnosis typically involves a weakening of the goal. Normally, we want an answer for diagnosis that doesn't have more malfunction hypotheses than necessary. The Classification method would typically produce a number of plausible hypothesis classes, the method cannot check for redundancy. Thus in fact the PSM achieves a weaker goal.

Several different research communities are nowadays working on ontologies. The SPAR initiative (Shared Planning and Activity Representation (*Tate, 1998*)) aims at developing a standard language for describing plans and domain knowledge used in planning. SPAR then corresponds to a second-level ontology for planning, but it does not, however, include an ontology of planning problem-solving steps.

4 Sharing of Method Knowledge

We argued that problem-solving method knowledge is a somewhat neglected dimension of ontology studies, and that a motivation for studying its ontology is knowledge sharing. Consider a knowledge-system builder who wishes to build a medical decision-making system. He wants a system that will accept a description of a patient's complaints and produce an answer describing what is wrong with the patient. We have deliberately not used the term diagnosis at this point. In order to build the system, he is told that he will have to find knowledge libraries containing knowledge about his medical domain and also about problem-solving methods. He will first have to search the method libraries for a method that relates to the goal type for his problem-solver. If successful in finding an appropriate method, he will have to use its operationalized specifications to implement it. Then he will have to search knowledge libraries for medical knowledge of the type the method needs. In theory he should be able to put the knowledge system together.

Suppose in a library of problem-solving methods there is a method for the goal type of

classification. We know that this method is a good candidate – simple diagnostic systems have been built by viewing diagnosis as classification of given symptoms into a disease/malfunction hierarchy. Let us assume that his diagnostic problem is actually simple enough for this approach to work. Our system builder sends out the goal, ``discover what is wrong with a patient", hoping that this would match a method in the library. The classification method in the library is indexed with the goal: ``classify a situation into a classification hierarchy". There is clearly a gap here. Before the system builder can realize that this method is applicable, he will have to figure out the relation between the level of abstraction at which his goal is stated and that of the method index. Note that we are not simply talking about the vocabulary differences --we are assuming that this problem is handled by the standardization of terms. We are talking about the deeper problem of realizing that ``finding what is wrong with a patient" is an instance of ``diagnosis," which under some conditions is solvable by ``classification."

Assuming that our system builder is able to recognize that his problem can be solved by a classification method, suppose he retrieves that method and implements it. Now he is going to need domain knowledge. The classification method in the library would describe the needed knowledge in terms of classification hierarchy, observations, and causal associations between observations and the classes. In order to map this ontology to something closer to his domain, he will need to map ``classification hierarchy" to ``disease hierarchy," and ``observations" to ``symptoms," ``complaints," etc.

We have thus seen three possible gaps that have to be bridged before we can perform effective method sharing.

1. The task and the method may use different underlying ontologies, and thus different terminology.

2. The task associated with the the method may not be strong enough for achieving the task at hand.

3. The method requirements are stated in different terminology than the domain knowledge that the method has to reason about.

Fensel (*Fensel, 1997*) proposes *adapters* as an instrument to bridge these gaps. An adapter can establish a mapping between the task and method ontology, can introduce teleological assumptions to weaken to goal to be achieved, and can establish a mapping between method and the domain ontology. In addition, adapters can introduce ontological assumptions on domain knowledge in order to satisfy the requirements of the method. A significant part of the work of adapters thus amounts to relating different ontologies to each other. How to do this is still an ongoing issue. Earlier we spoke about task-neutral methods. Clearly, the more task-neutral a method is, the more work needs to be done before it can be applied to a task and a domain, or, in other words, the more adapters need to be used. Adapters thus have the capability to turn task-neutral PSMs into task- dependent ones. Figure 2 illustrates how a system builder with a task interacts with a method library and shows the role of the adapters.

The simple example of our system builder who wishes to build a system that would decide what is wrong with a patient illustrates the problems in sharing and using method knowledge in a

seamless way. Nevertheless, method-sharing has been practical to some extent for more than a decade. The notion of problem-solving methods as separate items of knowledge that can be hooked up with domain knowledge and reused originated with (*Chandrasekaran, 1986*, *Clancey, 1985*), and extended by (*Musen, 1989*, *Steels, 1990*), and, in the CommonKADS project, by (*Benjamins, 1993*.) Typically, in this line of research it was required that the user of a method and the method indexing share the same terms in the description of the goal. For example, once one realizes that what he has is a classification problem, the Generic Task framework provided the CSRL language to help build a classification problem-solver. The language guided the system-builder in inputting domain knowledge that was required by the method. Similarly, DSPL helped users build routine design systems. However, neither the knowledge requirements nor the methods were stated in a formal way, making it difficult to search for knowledge or import methods from a different source.

5 Concluding Remarks

In this paper, we made a distinction, within a problem-solving agent, between knowledge of the world about which the agent is doing problem solving, and knowledge about how to solve problems. Of course, when an agent A is reasoning about another agent B, the domain knowledge component of agent A will include knowledge about problem solving in general, since A may need to predict, interpret, or debug the behavior of B. In this case, of course, A will also have problem-solving knowledge of its own. For this reason, it is useful to have a dual view of problem solving knowledge. In one view, problem-solving knowledge is a separate component of a problem solver. In another view, it is simply a kind of knowledge, part of the ontology of the phenomenon called problem solving that intensional agents engage in, whether they be artificial or natural.

Within the perspective of designing problem solving systems, we have argued for ontological engineering efforts to move in two parallel tracks, one with the current focus on representing domain knowledge, and the other with a new focus on representing problem-solving knowledge. 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 problem solving knowledge and related it to domain factual knowledge. It turns out that much research on task analysis forms a good basis to develop such problem-solving ontologies. We thus build on previous work in this area, some our own.

The topic that we have treated in this paper is important because a knowledge system consists of both domain knowledge and problem-solving knowledge. Therefore, it is necessary to study domain and problem-solving ontologies together and pay attention to their integration. Moreover, for practical knowledge sharing and reuse identifying only the ontologies for domain and problem-solving knowledge is not enough. We also need to establish mechanisms for relating them to each other, such as mapping terms or making assumptions. And it is even harder, but necessary, to automate these mechanisms as much as possible. Several new projects include these goals in their objectives such as DARPA's HPKB (4) project and Esprit's IBROW3 (5) project, but there is still a long way to go.

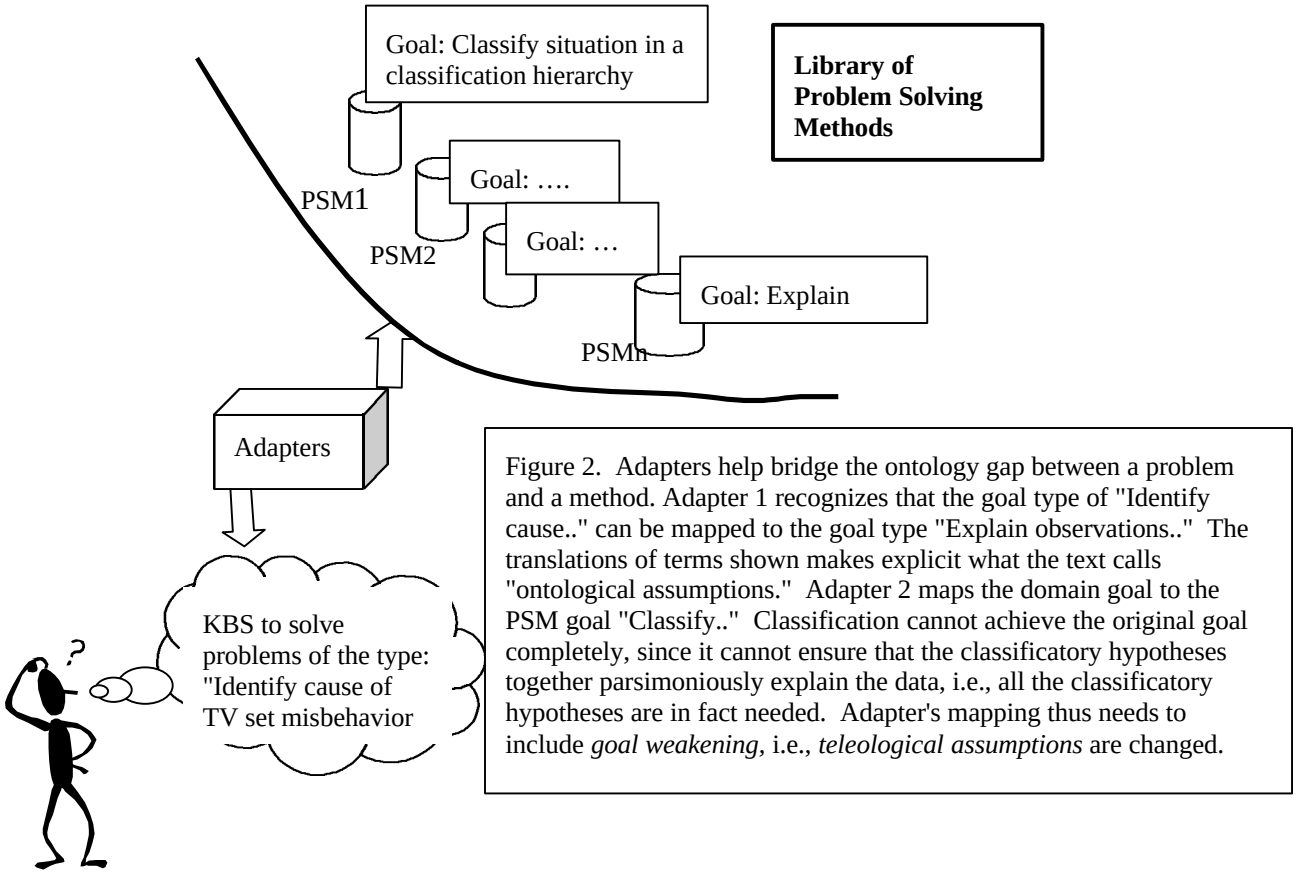

Figure 2. Adapters help bridge the ontology gap between a problem and a method. Adapter 1 recognizes that the goal type of "Identify cause.." can be mapped to the goal type "Explain observations.." The translations of terms shown makes explicit what the text calls "ontological assumptions." Adapter 2 maps the domain goal to the PSM goal "Classify.." Classification cannot achieve the original goal completely, since it cannot ensure that the classificatory hypotheses together parsimoniously explain the data, i.e., all the classificatory hypotheses are in fact needed. Adapter's mapping thus needs to include *goal weakening,* i.e., *teleological assumptions* are changed.

Term in KBS Task Domain Ontology	**Adapter 1: Mapping to PSM with goal "Explain observations."** **Term in PSM Ontology**	**Adapter 2: Mapping to PSM with goal "Classify observations."** **Term in PSM Ontology**
Identify cause..	Explain observation	Classify observations into classes (Change in *Teleological assumption*: Classes are not necessarily the most parsimonious account of causes. Only a weaker goal can be achieved.)
Symptoms	Abnormal observations	Abnormal observations
Device behavior data	Observations	Observations
Malfunction causes	Explanatory hypotheses	Malfunction hypothesis classes
<Malfunction> can cause <Behavior data>	<Hypothesis> can explain<Observation>	<Observations> categorized as <Class Hypothesis>

The Situated Cognition (SC) movement (*Winograd & Flores, 1987*) raises issues that challenge the hope for reusing knowledge. SC argues that knowledge is constructed in response to the situation, rather than retrieved and reused. For one situation, ``A causes B" might be the relevant knowledge, while for another, ``A causes B after a delay of 1 sec", and 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 essentially 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 also be made about the representation of problem-solving methods. Additionally, given the rather unsettled state of the philosophical foundations of upper ontology – as evidenced by the quote from a conference announcement earlier in the paper – there are additional reasons to be concerned about the vision of massive reuse of knowledge.

There are a number of things that make us optimistic, in spite of these concerns. For one thing, in these 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 changes in ontology will be needed less often than changes to a knowledge base. Secondly, situational change in knowledge for classes of well-defined tasks and domains is likely to be relatively controllable. For example, knowledge sharing for engineering design in the domain of operational amplifiers is much less subject to the changes in knowledge that SC proponents point to. Thirdly, knowledge sharing is not an all-or-nothing proposition. Knowledge may be imported from a library and modified to meet the needs of a specific task. The work needed for modifying knowledge that is close to what one needs would as a rule be much less than the work needed for identifying and encoding knowledge from scratch for each system. Exactly how much knowledge can be shared for what task domains remains an empirical issue, and not one that can be settled by a priori arguments. 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 suggests that 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.

Method sharing raises additional issues, some of that we discussed. Situations are described by the concrete task that a system-builder wishes his knowledge system to solve. As we saw, there will often be gaps between the way a system-builder conceived the task and the way methods would be indexed in a library. Such a gap can probably never be completely eliminated. However, some of the current ideas in the field that we discussed – assumptions that are attached to the methods and adaptors that use these assumptions – can help to span this conceptual gap.

Acknowledgment

This material is based upon work supported by The Office of Naval Research, grant number N00014-96-1- 0701, DARPA order number D594. Richard Benjamins is supported by the Netherlands Computer Science Research Foundation with financial support from the Netherlands Organisation for Scientific Research (NWO).

Bibliography

Barros, L. Nunes de *et al.*, 1996

Barros, L. Nunes de , Valente, A., & Benjamins, V. R. (1996). Modeling planning tasks. In *Third International Conference on Artificial Intelligence Planning Systems, AIPS-96*, pp. 11--18. American Association of Artificial Intelligence (AAAI).

Benjamins, 1993

Benjamins, V. R. (1993). *Problem Solving Methods for Diagnosis*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands.

Benjamins *et al.*, 1996

Benjamins, V. R., Fensel, D., & Straatman, R. (1996). Assumptions of problem-solving methods and their role in knowledge engineering. In Wahlster, W., (Ed.) , *Proc. ECAI--96*, pp. 408--412. J. Wiley & Sons, Ltd.

Beys *et al.*, 1996

Beys, P., Benjamins, V. R., & van Heijst, G. (1996). Remedying the reusability-usability tradeoff for problem-solving methods. In Gaines, B. R. & Musen, M. A., (Eds.) , *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pp. 2.1--2.20, Alberta, Canada. SRDG Publications, University of Calgary. http://ksi.cpsc.ucalgary.ca:80/KAW/KAW96/KAW96Proc.html.

Bylander & Mittal, 1990

Bylander, T. & Mittal, S. (1990). CSRL a language for classificatory problem solving and uncertainty handling. *AI Magazine*, 7:66--77.

Chandrasekaran, 1986

Chandrasekaran, B. (1986). Generic tasks in knowledge based reasoning: High level building blocks for expert system design. *IEEE Expert*, 1(3):23--30.

Chandrasekaran, 1990

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

Chandrasekaran, 1994

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

Clancey, 1985

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

Coelho & Lapalme, 1996

Coelho, E. & Lapalme, G. (1996). Describing reusable problem-solving methods with a method ontology. In Gaines, B. R. & Musen, M. A., (Eds.) , *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pp. 3.1--3.20, Alberta, Canada. SRDG Publications, University of Calgary. http://ksi.cpsc.ucalgary.ca:80/KAW/KAW96/KAW96Proc.html.

Fensel, 1997

Fensel, D. (1997). The tower-of-adapters method for developing and reusing problem-solving methods. In Plaza, E. & Benjamins, V. R., (Eds.) , *Knowledge Acquisition, Modeling and Management*, pp. 97--112. Springer-Verlag.

Fensel & Benjamins, 1996

Fensel, D. & Benjamins, V. R. (1996). Assumptions in model-based diagnosis. In Gaines, B. R. & Musen, M. A., (Eds.) , *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pp. 5.1--5.18, Alberta, Canada. SRDG Publications, University of Calgary. http://ksi.cpsc.ucalgary.ca:80/KAW/KAW96/KAW96Proc.html.

Fensel *et al.*, 1997

Fensel, D., Motta, E., Decker, S., & Zdrahal, Z. (1997). Using ontologies for defining tasks, problem-solving methods and their mappings. In Plaza, E. & Benjamins, V. R., (Eds.) , *Knowledge Acquisition, Modeling and Management*, pp. 113--128. Springer-Verlag.

Fridman-Noy & Hafner, 1997

Fridman-Noy, N. & Hafner, C. D. (1997). The state of the art in ontology design. *AI Magazine*, 18(3):53--74.

Gennari *et al.*, 1994

Gennari, J. H., Tu, S. W., Rotenfluh, T. E., & Musen, M. A. (1994). Mapping domains to methods in support of reuse. *International Journal of Human-Computer Studies*, 41:399--424.

Gruber, 1993

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

Guarino, 1995

Guarino, N. (1995). Formal ontology, conceptual analysis and knowledge representation. *International Journal of Human-Computer Studies*, 43(5/6):625--640. Special issue on The

Role of Formal Ontology in the Information Technology.

Johnson, 1991

Johnson, T. (1991). *Generic Tasks in the Problem-Space Paradigm: Building Flexible Knowledge Systems while Using Task-Level Constraints*. PhD thesis, The Ohio State University, Ohio.

Josephson & Josephson, 1994

Josephson, J. & Josephson, S., (Eds.) (1994). *Abductive Inference, Computation, Philosophy, Technology*. Cambridge, Cambridge University Press.

Klinker *et al.*, 1991

Klinker, G., Bhola, C., Dallemagne, G., Marques, D., & McDermott, J. (1991). Usable and reusable programming constructs. *Knowledge Acquisition*, 3:117--136.

Laird *et al.*, 1987

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). SOAR: an architecture for general intelligence. *Artificial Intelligence*, 33:1--64.

Lenat & Guha, 1990

Lenat, D. B. & Guha, R. V. (1990). *Building large knowledge-based systems. Representation and inference in the Cyc project*. Reading, Massachusetts, Addison-Wesley.

Marr, 1982

Marr, D. (1982). *Vision: A computational investigation into the human representation and processing of visual information*. San Francisco, W.H. Freeman.

McCarthy & Hayes, 1969

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

Mizoguchi *et al.*, 1995

Mizoguchi, R., van Welkenhuysen , J., & Ikeda, M. (1995). Task ontology for reuse of problem solving knowledge. In Mars, N. J. I., (Ed.) , *Towards very large knowledge bases*. IOS Press.

Musen, 1989

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

Neches *et al.*, 1991

Neches, R., Fikes, R. E., Finin, T., Gruber, T. R., Senator, T., & Swartout, W. R. (1991). Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36--56.

Newell, 1982

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, 18:87--127.

Schreiber & Birmingham, 1996

Schreiber, A. T. & Birmingham, W. P. (1996). The Sisyphus-VT initiative. *International Journal of Human-Computer Studies*, 44. Editorial special issue.

Schreiber *et al.*, 1994

Schreiber, A. T., Wielinga, B. J., de Hoog, R., Akkermans, J. M., & Van de Velde , W. (1994). CommonKADS: A comprehensive methodology for KBS development. *IEEE Expert*, 9(6):28--37.

Smith, 1991

Smith, Douglas, R. (1991). Structure and design of problem reduction generators. In Möller , B., (Ed.) , *Constructing Programs from Specifications*. North Holland.

Sowa, 1997

Sowa, J. F. (1997). *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Book draft.

Steels, 1990

Steels, L. (1990). Components of expertise. *AI Magazine*, 11(2):28--49.

Tate, 1998

Tate, A. (1998). Roots of SPAR - shared planning and activity representation. *The Knowledge Engineering Review*, pp. to appear. Special Issue on Ontologies.

van Heijst *et al.*, 1997

van Heijst , G., Schreiber, A. T., & Wielinga, B. J. (1997). Using explicit ontologies in KBS development. *International Journal of Human-Computer Studies*, 46(2/3):183--292.

Wielinga & Breuker, 1986

Wielinga, B. J. & Breuker, J. A. (1986). Models of expertise. In *Proceedings ECAI--86*, pp. 306--318.

Wieringa & de Jonge , 1995

Wieringa, R. & de Jonge , W. (1995). Object identifiers, keys and surrogates -- object

identifiers revisited. *Theory and Practice of Object Systems (TaPOS)*, 1.

Winograd & Flores, 1987

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

Footnotes

note-1

Call for papers for a special issue on ``Temporal Parts'' for The Monist, An International Quarterly Journal of General Philosophical Inquiry.

note-2

The term ``task'' has been used in two related, but still distinct, ways, causing some confusion. In one usage, task is a set or sequence of things to do: that is, task is something you do. Another usage has task as a term for a goal type, leaving open what the steps are in achieving it. We use it in the latter sense; that is, tasks are goal types, what needs to be accomplished.

note-3

Our examples throughout are chosen mainly for clarity in making the conceptual points rather than for accuracy or completeness in describing the knowledge. In particular, in this example, the knowledge for establishing hypotheses is usually substantially more complex than the example.

note-4

HPKB URL is http://www.teknowledge.com:80/HPKB/.

note-5

URL IBROW3 is http://www.swi.psy.uva.nl/projects/IBROW3/home.html.