Representing Capabilities of Problem Solving Methods

Bill Swartout
USC
Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
swartout@isi.edu

Yolanda Gil
USC
Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
gil@isi.edu

Andre Valente
USC
Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
valente@isi.edu

Abstract

In order to develop and use shared libraries of problem-solving methods, it is of paramount importance to provide adequate descriptions of their capabilities and competence. Methods must be indexed and organized based on their capabilities so that they can be retrieved when their capability is adequate for the task at hand. This paper describes the approach taken in EXPECT for representing method capabilities and argues that it has important features that should be used for describing methods in shared libraries. EXPECT's capability representation is tightly coupled with the domain ontologies in the knowledge base, can express task-related parameters explicitly, and is based on case grammars. This representation allows the system to reason about the capability descriptions through class subsumption and reformulation. The benefits of this approach include self-organizing method libraries, reuse, and support for explanation. The representation has already been used extensively within EXPECT to express a wide range of method capabilities, ranging from abstract to specific, small to large, and domain-dependent to general-purpose methods. The paper also discusses some of the additional features that we anticipate will be useful to structure shared method libraries.

1 Introduction

Libraries of problem solving methods could facilitate the construction and adaptation of knowledge based systems [Chandrasekaran 1986; Eriksson et al. 1995; MacDermott 1988; Breuker and Van de Velde 1994]. Rather than building a system from scratch, as is current practice, system builders would assemble a knowledge based system from reusable components drawn from shared ontologies and libraries of problem solving methods. By reusing components, this approach should allow knowledge-based systems to be constructed more rapidly. Further, the resulting systems should be more error free since they will be constructed from existing (and presumably debugged) resources. Finally, because the emphasis in system construction will be on assembling existing components, rather than building things from scratch, it should be possible for less experienced individuals to build knowledge based systems successfully.

But how should these libraries of problem solving methods be organized? How can the capabilities of problem solving methods be represented? This approach to system construction will only work if people (and machines) can easily find the methods that are capable of addressing the problem at hand. Other approaches, such as CommonKADS, use a functional specification of method capabilities. However, the matching of these capabilities with problem goals in e.g. the CommonKADS Library [Valente et.al., 1998] are meant to be done by a human that analyzes a semi-formal method description. The library is organized "by design", for example using typologies of methods and explicit collections of related decompositions.

In contrast, our approach aims to build a library of problem-solving methods that is self-organizing, in the sense that we can automatically find the right place for a new method in the library and have tools that can use the library to build a problem solver for a specific problem. We believe such a library will enhance reusability. Also, we want to have capabilities that are amenable to produce explanations – both of the methods and the systems constructed using these methods. To achieve these goa ls, we need a rich and expressive specification of method capabilities that allows the

The copyright of this paper belongs to the papers authors. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage.

Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5) Stockholm, Sweden, August 2, 1999
(V.R. Benjamins, B. Chandrasekaran, A. Gomez-Perez, N. Guarino, M. Uschold, eds.)

http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-18/

system to reason about the capability descriptions through class subsumption and reformulation.

In this paper we describe the approach we have been using to represent problem solving method capabilities in EXPECT, our knowledge based system framework that supports knowledge acquisition [Swartout and Gil 1995; Gil 1994; Gil and Melz 1996]. We begin with a brief overview of the EXPECT framework, followed by a discussion of a set of desiderata that motivated the design of our representation for method capabilities. We then discuss the representation we use in detail.

2 The EXPECT Framework

A major goal of the EXPECT framework is to allow domain experts to change and add knowledge to a knowledge based system. EXPECT keeps track of the interdependencies in a knowledge based system, such as what factual knowledge must be present to support the problem solving methods that the KBS uses, and how factual knowledge is used in problem solving. For instance, in a configuration system that uses propose-and-revise as its problem solving method, each constraint must have one or more associated “fixes” that are used in problem solving to resolve a violation of the constraint when it occurs. This is an example of a dependency that arises between the domain representation of constraints and the problem solving method. EXPECT captures such dependencies in an *interdependency model* which is specific to each knowledge based system built in EXPECT. When new knowledge is added to a knowledge based system, EXPECT examines the interdependency model to determine what additional knowledge must be provided to make the new knowledge usable by the problem solving methods currently employed in the system. Similarly, if one of the problem solving methods is modified, EXPECT rederives the interdependency model to determine if any of the dependencies have changed. If so, it will request the needed additional information from the user. In this way, EXPECT helps a user modify and adapt a knowledge based system while freeing him from the need to understand the details of the implementation. Figure 1 shows the architecture of EXPECT.

EXPECT uses Loom [MacGregor 1991] to represent domain facts and the domain ontology. Loom is a description logic-based representation. Like other description logics, Loom is based on a semantic network approach to knowledge representation. *Concepts* in Loom are descriptions of objects (which may or may not actually exist) while Loom *instances* represent objects that do exist. Concepts can have *roles* which may be used to specify attributes of the concept. A distinguishing feature of description logics like Loom is that they provide a way of precisely defining the meaning of a concept, that is, what it denotes. Loom provides a *classifier*, which is a reasoner that uses concept definitions to determine whether one concept subsumes another concept. Specifically, a concept A is said to subsume a concept B if all the possible entities that could be described by B are also necessarily described by A. For example, ``a man who only drinks beer'' subsumes ``a man who only drinks imported beer.'' The classifier can determine whether all the instances that could possibly be described by one concept are also necessarily described by another based on the definitions of the two concepts. As a result, it is possible to automatically organize concepts into an AKO (a kind of) lattice based only on their definitions.

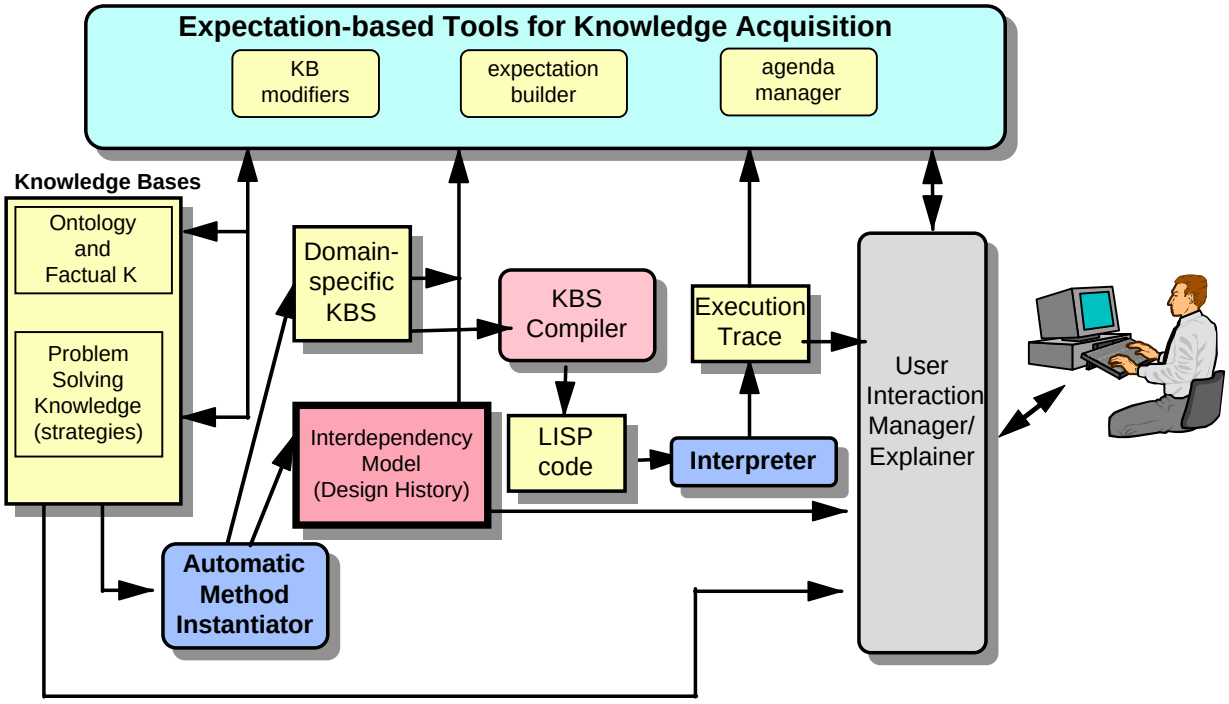

Figure 1: EXPECT Architecture

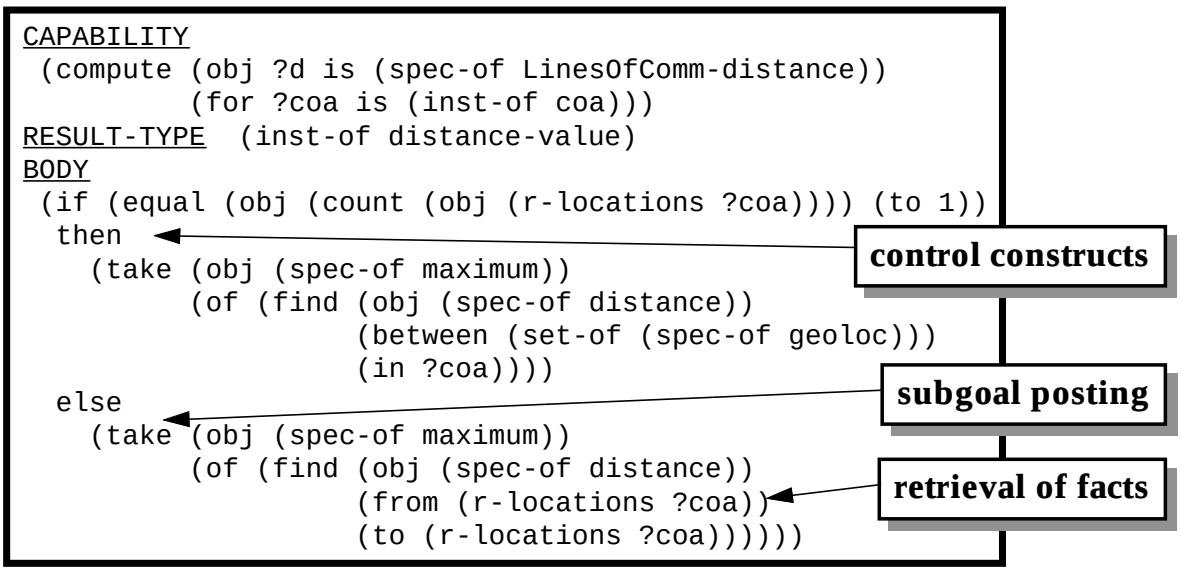

Figure 2: An EXPECT Method

In EXPECT, a *goal* represents a task to be done or a problem to be solved. Problem solving methods are used to accomplish these goals. Each problem solving method has a *capability description* which describes the kinds of problems the method can solve and a *method body* which consists of step(s) for achieving the method's capability. These steps may include subgoals. Figure 2 shows an example of a problem-solving method in EXPECT. The method computes the distance of lines of communication in a course of action (COA).

As we described above, EXPECT's interdependency model captures part of the design of the knowledge based system. EXPECT creates an interdependency model for a knowledge based system by synthesizing the system from a set of abstract problem solving methods and knowledge about a domain. As it synthesizes the system, EXPECT records how different parts of the system depend on each other in the interdependency model.

EXPECT uses a form of partial evaluation and hierarchical decomposition to create a knowledge based system. Starting with an initial high level goal that specifies what the knowledge based system is supposed to do, EXPECT looks for a method whose capability description matches the goal. When a method is found, its method body is instantiated by replacing variables in the method with corresponding instances in the goal. The instantiation of the method body may result in the posting of subgoals, in which case the process recurs. If no method can be found for a goal, EXPECT attempts to reformulate the goal into a new goal or set of goals that are semantically equivalent to the original. It then attempts to achieve these new goals, as we describe below. During this entire process, EXPECT records in the interdependency model how specific factual information is used in expanding the problem solving methods, thus capturing how different parts of the system depend on each other.

3 Desiderata for the Representation of Capability Description and Goals

Linking up goals and problem-solving methods is a critical part of EXPECT's approach to knowledge acquisition, and it places a number of demands on the representation of goals and the capability descriptions of problem solving methods. In this section we outline the desiderata that led to EXPECT's representation for capability descriptions and goals.

- *A representation tied to the domain ontology.* We wanted a representation that was tied to the ontologies used in EXPECT so that goals could be defined using terms from the domain. Further, integrating the representation for capability descriptions and goals with the domain ontology assures us that the semantics of the two representations will be consistent.
- *A broad spectrum representation.* Some problem solving methods are very general, while others are especially tuned to work in highly specialized situations. We believe that problem-solving method libraries need to include both kinds of methods. General methods will provide broad coverage and allow us to build robust systems, while highly specific methods will substantially enhance efficiency in specific situations. We wanted a representation that would allow us to describe the capabilities of both very general and highly domain-specific methods.
- *Support for "loose coupling" between goals and method capabilities.* Reuse of problem solving methods will be increased if the capability

descriptions of the methods don't have to match goals exactly. We wanted a representation that would allow the system to find methods that could work for a particular goal, even if they did not match it exactly.

- *Support for reformulation.* Loose coupling and reuse can be further increased if goals can be reformulated. Reformulation involves mapping a goal into a new goal or set of goals that is semantically equivalent to the original goal. Reformulation allows the system to find a way of achieving a goal even if a problem-solving method could not be matched against the original goal. To be able to automatically reformulate a goal, the semantics of the individual terms that comprise the goal must be well specified so that they can be mapped into new terms to create equivalent goals.
- *Understandable by users.* Since a goal of the EXPECT effort is to support knowledge acquisition from domain experts, we wanted a representation that could be easily understood or paraphrased into English.
- *Self-organizing.* In our view, problem solving method libraries are likely to become quite large in the future. Further, both AI experts and non-experts will contribute to these libraries and use methods from them. For these reasons we felt it was important to have a representation for method capabilities that would support self-organization, that is, that would allow us to organize the methods into a hierarchy automatically based on their capability descriptions. This would allow either a machine or person to find methods that were applicable to a particular problem easily.

In the next section, we describe our representation for goals and method capabilities that helps us achieve the desiderata outlined above.

4 Representing Goals and Capabilities

EXPECT uses a structured representation for goals that arise during problem solving and the capabilities of methods that can be used to achieve those goals. Goals and capabilities are represented as verb clauses using a case-grammar style formalism [Fillmore 1968]. Each goal consists of a verb, which specifies what is to be done, and a number of roles, or slots, which specify the parameters to be used in the action. The parameters use terms that are defined in the domain ontology. For example, the goal of estimating the closure date of particular transportation movement would be specified roughly as:

```
estimate OBJ closure-date OF
  transportation-movement-1
```

Here, *estimate* is the verb, and the roles are indicated in upper case. Roles are filled by Loom concepts and instances taken from the ontology, which couples our representation with the ontology.

In EXPECT, roles can be filled in several different ways, which allows considerable flexibility in specifying a task to be done. A role can be filled by a specific *instance*:

```
add OBJ 3 TO 5
```

which allows us to specify particular instances that are to be used in an action. A role can be filled by a concept:

```
compute OBJ (spec-of factorial) of 7
```

In this case, the concept `factorial` is used to specify the kind of task that is to be done. The data required to perform the computation are specified as parameters (in this case the number 7), while these additional task parameters allow us to express what needs to be done with that data in an explicit way and are not strictly necessary to perform the computation itself. The fact that roles can be used both to specify the parameters or objects that will be involved in a task *and* to further describe or specify the task itself is one of the key capabilities that our representation supports, providing us with a rich language for specifying goals.

Roles can be a type of an instance, as in:

```
divide OBJ (inst-of number) BY 2
```

This results in a generic goal that can be instantiated with any elements of that type.

Roles can also be filled by extensional sets as in:

```
find OBJ (spec-of maximum) OF (42 2
                               99)
```

or they can be filled by intensional sets, where the set is described by a concept:

```
find OBJ (set-of
          (spec-of
              violated-constraint))
     IN (inst-of configuration)
```

Finally, it is possible to use descriptions (which are similar to the definitions of Loom concepts) in roles:

```
estimate OBJ support-personnel
         IN (and location
                 (exactly 0
seaports))
```

This is a goal to estimate the support personnel in a location with no seaports.

This approach provides us with a rich language for specifying behaviors. The use of a case grammar formalism makes it relatively straightforward to paraphrase the goals into natural language helping to make them more understandable [Swartout et al. 1991].

Capability descriptions for methods are specified in a similar way, except that variables may appear in the capability descriptions. These are bound when the capability descriptions are matched with goals. Figure 2 shows an example of a method and its capability.

As we just showed, EXPECT's language to describe goals and capabilities is very expressive. An important aspect of EXPECT is how it reasons about method capabilities with this representation, exploiting *subsumption* and *reformulation* as we describe next. Further details can be found in [Swartout and Gil 1995; Gil and Gonzalez 1996].

4.1 Creating LOOM Descriptions of Goals and Capabilities

We described earlier how EXPECT relies on LOOM's classifier to automatically organize concepts in an AKO lattice. EXPECT also relies on the LOOM classifier to reason about what goals and capabilities subsume others. This is achieved by turning goals and capabilities into LOOM descriptions. EXPECT has a core set of Loom definitions that are used for this, and include **action name** (its subclasses are essentially verbs), **action role** (its subclasses are OBJ and any parameter name), **goal**, and **capability**. Action roles are relations whose domain is an action name, and whose range can be any existing concept in the domain (ex: ship, number) qualified by its parameter type (set or element, concept or instance, extensional or intensional). For example, the goal to compute the factorial of a number is expressed in EXPECT as:

```
(compute
     (obj (spec-of factorial)
     (of (inst-of number)))
```

The corresponding Loom definition that is created is:

```
(defconcept CM20
  :is (:and compute
        (:the obj (:and concept-desc
                        factorial))
        (:the of (:and instance-desc
                       number))))
```

LOOM's classifier is now able to reason with this definition. Every term used in the parameters have their own definitions, provided in the ontologies, and LOOM will use them in reasoning about goal subsumption. Notice that these terms and their definitions can be domain independent (e.g., `violated-constraints`, `maximum`) or domain dependent (e.g., `location`, `closure-date`).

4.2 Self-Organizing Method Libraries

Using the techniques just described, EXPECT creates Loom definitions for the capabilities of all the methods that are defined in the knowledge base. Loom's classifier reasons about these definitions and places them in a lattice, where more general definitions subsume more specific ones. Notice that this subsumption reasoning uses the definitions of the domain terms and ontologies that are part of EXPECT's knowledge bases. As a result, the capability of a method to "move cargo with a vehicle" will subsume one to "move cargo with an aircraft", because according to the domain ontologies vehicle subsumes aircraft. This is illustrated in the method hierarchy shown in Figure 3. As a result, EXPECT's methods are *automatically organized according to their capabilities*, and their capabilities can be compared based on their place in the

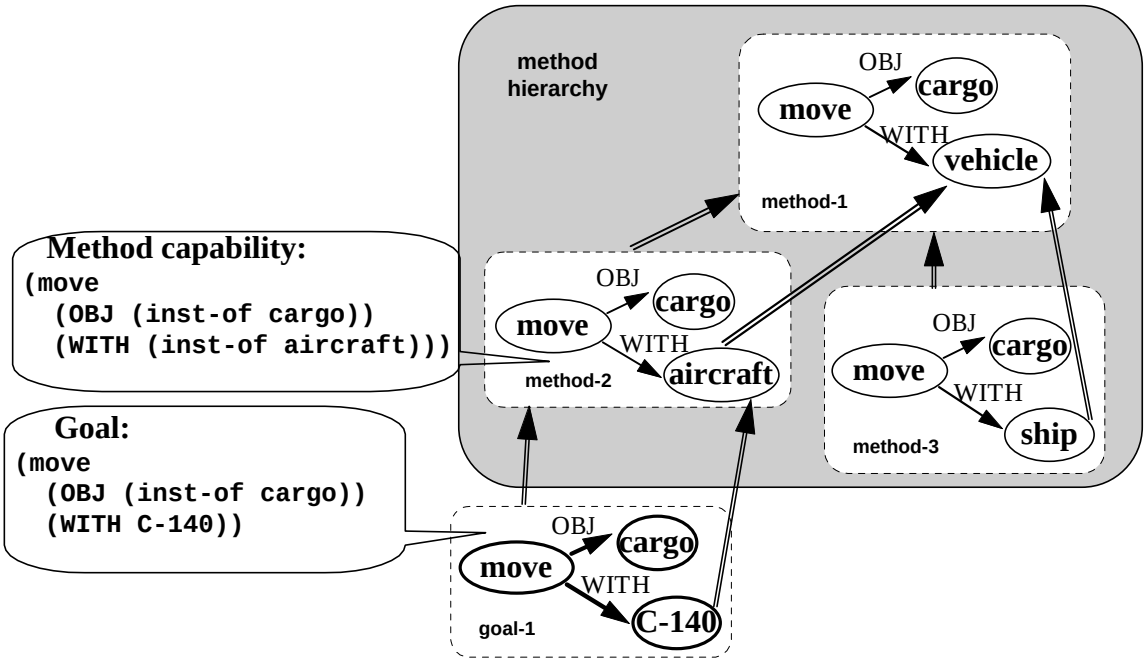

Figure 3: Translating Goals and Capabilities to Loom to organize and retrieve methods

lattice.

4.3 Matching Goals and Capability Descriptions

EXPECT also exploits the representation of goals and capabilities for matching method capabilities with the goals that arise during problem solving. EXPECT's matcher first translates the posted goal into a Loom concept, and then invokes the Loom classifier in order to find methods whose capability descriptions subsume the posted goal. Figure 3 illustrates this matching process for the goal of moving some cargo with a C-140 (which is a particular kind of aircraft).

Once the match has been made using the Loom representation for the goal and capabilities, the original representation is used to bind parameters in the goal to corresponding variables in the capability description. This is necessary since Loom does not support variables in concepts.

4.4 Reformulating Goals

When a goal is posted while EXPECT is synthesizing a knowledge based system and no method can be found with a matching capability, EXPECT attempts to reformulate the goal by transforming it into a new goal (or set of goals) that is equivalent to the original goal, but expressed in different terms. EXPECT then tries to find methods for achieving these new goals. This automatic reformulation allows EXPECT to reuse methods in a broader range of circumstances than would be possible if EXPECT required an exact match for goals and methods. EXPECT supports several types of reformulations.

- A **covering reformulation** is a form of divide and conquer. It transforms a goal into a set of goals that partition the original goal. If all the goals in the set are achieved, the intent of the original goal is achieved. Figure 4 shows an example covering reformulation. A goal of estimating support personnel has been posted, but no applicable methods have been found. Because EXPECT's ontology (as shown on the left in Figure 4) indicates that the concept support personnel is partitioned into unloading personnel, seaport support personnel and airport support personnel, EXPECT can reformulate the original goal into three new goals as indicated on the right in Figure 4.
- A **set reformulation** is like a covering reformulation except that it involves a goal over a set of objects which is reformulated into a set of goals over individual objects.
- An **input reformulation** is somewhat similar to the support that some languages provide for polymorphic operators. This kind of reformulation occurs when a goal is specified with a general parameter and no single method is available at a sufficiently general level to handle the parameter. In that case, EXPECT attempts to reformulate the goal into cases based on the subtypes of the parameter given in the ontology. EXPECT also creates dispatching code so that once the knowledge based system has been synthesized and is being run, the code will dispatch to the appropriate subcase based on the actual type of the parameter that is passed in at runtime.

Goal reformulations allow us to state the description of method capabilities more independently from the statement the descriptions of the goals that are posted by other methods or by the user. The benefit is a more loosely coupling between methods and tasks, i.e., between what is to be accomplished and what are possible ways to accomplish it. Goal reformulations also illustrate how method libraries can leverage from domain ontologies and their structure.

5 Related Work

Several groups have proposed approaches for representing PSM capabilities. In CommonKADS methodology and related work [Schreiber et al 1994, Valente et al 1998], method capabilities are represented in a functional way, i.e., by specifying inputs and outputs, plus the knowledge used in the process (called static knowledge). Part of the semantics was also

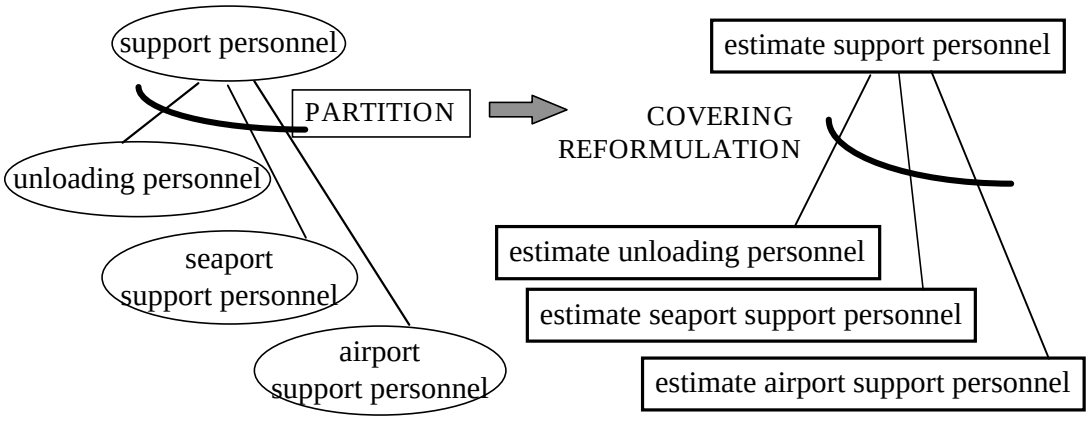

Figure 4: A covering reformulation

expressed by relating the method to an element of a typology of methods, typically at the lowest grain size level (the so-called canonical inferences, see [Aben, 1993]) or at the highest grain-size level (e.g. the suite of problem types by [Breuker, 1997]). Despite the fact that EXPECT also models inputs and outputs of methods, there are many differences between the two approaches. First, EXPECT uses the case frame representation to establish a hierarchy of types of goals, while there is no such notion in CommonKADS. Second, because the EXPECT framework is based on the idea of deriving or finding what knowledge is used by a method in the construction of a problem solver, it is able to derive (instead of requiring the user to specify) the static knowledge used by a method. Third, while the terms used in specifying the input and output roles in CommonKADS are basically arbitrary, EXPECT relies on an ontology to find interrelations between them and reason about them in constructing a problem solver. In this regard, the EXPECT approach is closer to the approach used in the Role-Limiting Methods or in PROTÉGÉ, where there is a method ontology that characterizes input, output and static knowledge of a method. An interesting difference, however, is that EXPECT does not force the user to separate the method ontology from the domain ontology, because the system is able to find out automatically what knowledge is referenced by the method capability specifications. In summary, EXPECT finds the roles that knowledge will play when the knowledge-based system is derived by the method instantiator, while these roles are pre-specified by most other approaches.

Another important line of research about representing method capabilities is the work on specifying assumptions of PSMs [Fensel et al, 1996]. EXPECT represents some assumptions in the way the Loom knowledge base is put together. For instance, it can represent a completeness assumption about descriptions of ports by defining them to have at least one berth. This is exploited by the Loom reasoning engine: if an instance of port does not have a berth, Loom will classify it as incoherent because it contradicts the definition of the concept port. Other assumptions are derived during the matching process. For instance, assumptions about knowledge availability can be derived by analyzing a method and concluding that, for example, the capacity of the C-140 needs to be known so that the method can calculate whether it can move a certain cargo using a C-140.

6 Summary

We have described the approach that is used in EXPECT to describe and reason about goals and method capabilities. The main features of the approach are:

- the method representation is *tightly integrated with ontologies* as a model of the objects that the methods reason about. Ontologies may be domain-specific or high-level ontologies.
- a *wide range of parameter types*, including intensional sets and generic instances
- method capabilities *state explicitly information about the type of computation* that the method does, not just which data it uses.
- a *case-frame representation* is used that is understandable by users and supports explanation.
- a *broad spectrum of methods* can be represented, ranging from small domain-specific methods to very general domain-independent methods (such as propose-and-revise)
- *goals can be reformulated* into more specific subgoals by using domain knowledge stated in the domain ontologies.

There are several advantages of this approach that method libraries can benefit from:

- a **loose-coupling between goals and method capabilities**, which facilitates reuse.
- **self-organizing method libraries**, where key features of the method (in our case their capabilities) are used to automatically determine how they relate to one another.
- **understandable by users**, since they are structured as case frames that can be easily paraphrased.

An important additional feature of EXPECT is that the method body, i.e., the description of the procedure and subtasks that accomplish the method's capability, is also expressed explicitly. This is important for reuse, since it allows adaptation of the methods by using EXPECT's knowledge acquisition tools. It is also important because it allows users to look at the method body and get first-hand information about how the method works (as opposed to informal or formal descriptions created separately from the actual code).

We are planning several extensions to our current approach in order to make it more suitable for describing capabilities of methods in shared libraries.

One set of extensions is motivated by our work on representing role-limiting methods in EXPECT [Gil and Melz 1996]. We find that the knowledge roles used in the method should be expressed explicitly, and that EXPECT can derive them automatically by looking at interdependencies that it derives. We found the need for an extensive range of types of knowledge roles, including classes to be defined in the domain ontologies and method stubs to be mapped to domain-dependent methods. We would like to be able to express additional types of parameters in goals and method capabilities, such as relations and method classes. Finally, we would like to be able to express how methods work together to form larger macro-methods.

Another set of extensions is motivated by our participation in DARPA's High Performance Knowledge Bases Program [Cohen et al. 1998], where one of our goals is to develop with others a shareable, distributed

library of implemented problem-solving methods that can be used in conjunction with large-scale ontologies to rapidly create knowledge based systems. In order to organize these method libraries, in addition to their capability we would like to represent and reason more explicitly about the assumptions that they make on ontologies, the subtasks that they pose, the submethods that they use, and other information about the method's implementation.

Acknowledgements

We would like to thank all the past and present members of the EXPECT project for their contributions to this work. We gratefully acknowledge the support of DARPA with contract DABT63-95-C-0059 as part of the DARPA/Rome Laboratory Planning Initiative and with grant F30602-97-1-0195 as part of the DARPA High Performance Knowledge Bases Program.

Bibliography

[Aben 1993] Aben, M. "Formally specifying re-usable knowledge model components". *Knowledge Acquisition*, 5:119--141, 1993.

[Breuker 1997] Breuker, J. "Problems in indexing Problem Solving Methods". In R. Benjamins and D. Fensel, editors: *Proceedings of the IJCAI'97 Workshop on Problem Solving Methods*, 1997.

[Chandrasekaran 1986] Chandrasekaran, B. "Generic tasks in knowledge-based reasoning". *IEEE Expert* , 1(3):23-30, 1986.

[Eriksson et al. 1995] Eriksson, H., Shahar, Y., Tu, S. W., Puerta, A. R., and Musen, M. A. "Task modeling with reusable problem-solving methods". *Artificial Intelligence* 79(1995):293--326.

[Cohen et al. 1998] Cohen, P.; Schrag, R.; Jones, E.; Pease, A.; Lin, A.; Starr, B.; Gunning, D.; and Burke, M. " The Darpa High-Performance Knowledge Bases Project". *AI Magazine*, 19(4), 1998.

[Fensel et al, 1996] Fensel, D and Benjamins, R. "Assumptions in Model Based Diagnosis". In Gaines, B. and Musen, M., editors: *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*. Banff, 1996.

[Gil and Gonzalez 1996] Gil, Y. and Gonzalez, P. "Using Description Logics to Match Goals", In Proceedings of the 1996 International Workshop on Description Logics (DL-96), November 2-4, 1996, Boston, MA.

[Gil and Melz 1996] Gil, Y., and Melz, E. "Explicit representations of problem-solving strategies to support knowledge acquisition". In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 1996.

[Gil 1994] Gil, Y. Knowledge Refinement in a Reflective Architecture. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-94)*, 1994.

[MacGregor 1991] MacGregor, R. "The evolving technology of classification-based knowledge representation systems". In Sowa, J., ed., *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. San Mateo, CA: Morgan Kaufmann.

[Marcus 1988] Marcus, S. "SALT: a knowledge-acquisition tool for propose-and-revise systems," in *Automating Knowledge-acquisition for expert systems* S.Marcus (ed), pp. 81-121. Kluwer Academic Publishing. 1988

[McDermott 1988] McDermott, J, "Preliminary steps toward a taxonomy of problem solving methods," in *Automating Knowledge-acquisition for expert systems* S.Marcus (ed), Kluwer Academic Publishing. 1988

[Musen 1992] Musen, M. A. "Overcoming the limitations of role-limiting methods," *Knowledge Acquisition*, 4(2):165--170. 1992.

[Musen and Tu 1993] Musen, M. A., and Tu, S. W. Problem-solving models for generation of task-specific knowledge acquisition tools. In J. Cuena (Ed.), *Knowledge-Oriented Software Design*, Elsevier, Amsterdam, 1993.

[Schreiber et al,1994] Schreiber, A., Wielinga, B., Akkermans. J., Van de Velde, W. and de Hoog, R. "CommonKADS: A comprehensive methodology for KBS development". IEEE Expert, 1994.

[Swartout and Gil 1995] Swartout, B. and Gil, Y. "EXPECT: Explicit Representations for Flexible Acquisition". In *Proceedings of the Ninth KnowledgeAcquisition for Knowledge-Based Systems Workshop (KAW'95)* Banff, Canada, February 26-March 3, 1995.

[Swartout et al 1991] Swartout, W. R., Paris, C. L., and Moore, J. D. "Design for Explainable Expert Systems". *IEEE Expert* 6(3):58-64, 1991.

[Valente et al, 1998] Valente, A., Breuker, J. and Van de Velde, W. "The CommonKADS Library in Perspective". *International Journal of Human-Computer Studies*, 49: 391—416, 1998.