Methods to Solve the Context Dependency Problem of Problem-Solving Methods

Dieter Fensel[1] and Arno Schönegge[2]

[1]University of Karlsruhe, Institute AIFB, 76128 Karlsruhe, Germany, dfe@aifb.uni-karlsruhe.de
[2]University of Karlsruhe, Institute LDK, 76128 Karlsruhe, Germany, schoenegge@ira.uka.de

Abstract. Context dependency of knowledge models causes several problems: unreliability of knowledge-based systems, maintainability effort and limitations for sharing and reuse. Problem-solving methods are knowledge models of the reasoning process of knowledge-based systems. In the paper we present two complementary methods to deal with the situatedness of problem-solving methods. First, we present a method that helps to investigate the context dependency of a method by making underlaying assumptions explicit. We use failed proof attempts as a search method for assumptions and analysis of these failures for constructing and refining assumptions. Second, we present a structured approach for developing and adapting problem-solving methods to different contexts. We view this process as stepwise combination process of methods we adapters. Starting from very generic strategies with very general data structures and add adapters that refine the states of the problem-solving process, that refine state transitions, and that add assumptions necessary to link the competence of a method with given problem definitions and domain knowledge.

1 Introduction

During the last years, Problem-solving methods (PSMs) have become quite successful in describing the reasoning behavior of knowledge-based systems (KBSs). Despite the strong agreement on the usefulness of PSMs and the large body of documented PSMs there is still a lack of clear methodological support in developing PSMs and in (re-)using them. Recent work [AWS93], [BA], [Fen95a], [WAS95], [BG96], [BFS96], [FB96], [FEM+96], [FS96], [HS96], [MZ96], [Bre97], [FS97a], and [Tei97] provide in-depth analysis of the essence and main rationales of some PSMs. There seems to be common agreement that the *assumptions* underlying a reasoning process are central in characterizing and developing PSMs. Some of the papers outline general steps that have to be taken in developing PSMs. However, it still remains rather unclear how to develop PSMs, how to adapt PSMs to given problems and domain-specific circumstances and how to select PSMs from a library, i.e. how to organise such a library. We provide two complementary solutions for these problems.

Assumption Detector Method. PSMs for KBSs need to make assumptions to provide effective and efficient problem solving: assumptions about the scope of the problem they should solve and assumptions about the domain knowledge they can use as a resource for their reasoning process [FS96], [BFS96]. If these assumptions are made explicit they can improve the reusability of PSMs by guiding the refinement process of PSMs for a given application and by defining goals for the acquisition process of domain knowledge. However, making the underlying assumptions explicit is not an easy task. The goal of our paper is to provide methods for solving this problem. The main idea is to construct mathematical proofs and to analyse their failure as a systematic means for formulating assumptions. Tool support is provided by adapting the Karlsruhe Interactive Verifier (KIV) [Rei95] for our purpose. KIV is an interactive theorem prover that returns with open goals if a proof could not be completed. These open goals can be used to derive the assumptions we are looking for.

Adapter Method. We show a principled way of developing and adapting PSMs and provide a new

concept on how to organise a library of PSMs to support their reuse. This is mainly achieved by using *adapters* as a means to express the refinement of PSMs. Adapters were originally introduced in [FSG+96], [FG97] to allow the independent specifications of problem definitions, PSMs, and domain knowledge. Building KBSs from reusable elements require adapters that properly link these elements and adapt them to the application-specific circumstances. Because these elements should be reusable, they must abstract from application-specific circumstances and because they are specified independently from each other there is a need to introduce their mappings. Introduced as glue that brings other elements together, we will give adapters a much more prominent role during this paper. They will play a central role in refining PSMs. Actually, a refined version of a PSMs is achieved by combining it with an adapter.

The paper is organised as follows. In section two we discuss the problems our approach tries to solve. We sketch the problem of context dependency of PSMs. In section three and four we discuss two different methods that deal with context dependency of PSMs. First we introduce the *assumption detector method* that supports the process of making a context of a method explicit. Second we introduce the *adapter method* that supports the modification of a PSMs to a new or changed context. Conclusions, related and future work are discussed in section five.

2 The Problem

Cyc is a prominent and long-term research project aiming for a large knowledge base enabling common-sense reasoning [GL90]. To achieve this goal large amounts of human common sense knowledge were encoded in their representation formalism. However, they encountered a serious problem when formalizing human knowledge. Knowledge is *situated* and its usefulness for different situations is limited [MC]. Knowledge can be understood as a model of the reality serving specific (probably implicit) purposes [AFH94]. Trying to represent this situated knowledge creates recursively the same problem. A representation of this knowledge, i.e. a model, reflects a point of view taken by the modeller [Cla93]. The point of view he takes reflects implicitly or explicitly the intended use of the model. [Guh93] enumerates three aspects of a representation that reflect the situatedness of a knowledge model:

- the *vocabulary* used to formulate the model,
- the *granularity and accuracy* of the model, and
- the *assumptions* that underlay the model.

[McC93] and [Guh93] present *context logic* as a means to deal with the problem.[1] The notion of context is reified within first-order logic by introducing terms that denote a context, i.e. that provide context names. Simplified, a literal $P(a)$ becomes extended by a context name c as an second argument $P(a,c)$ that states that $P(a)$ is assumed to be true in context c. That allows to express that $P(a)$ may be true in one context c and false in another context c'.

However, providing a notion for context dependency (i.e., a language that allows to express situatedness) is only halve of the work. The second problem is *how to get aware of the context dependency of a model*. That is, can we provide methodological support in building a model of the context dependency of our knowledge model. Making the context of a (knowledge) model explicit is a tricky problem that is unsolvable in principle however (heuristically) solvable in practice. It is unsolvable in principle because its solution would require to solve a problem of infinite regress. Making a context explicit needs a perspective that is used as point of reference for this activity. Clearly this creates a new context dependency of the model. “As a consequence, there doesn’t seem to be any certain knowledge on which to stop and stand ... that doesn’t rely on unproven assumptions“ [AFH94]. However, this does not imply that there is no pragmatical solution at all. [AFH94] claim that a notion of purpose realized by a social selection process comparable to the effect of the evolution process in nature solves the problem in practice. Some assumptions remain if they coincide with the desired purpose and its efficient achievement and other get rejected inquiring deeper search for a foundation.

1. See [AS96] for a survey on formalization formalisms for contexts.

Making the context of a knowledge model explicit is an infinite process. We cannot expect to get a solid ground where we can built our knowledge model on top. Therefore, the only thing we can provide are shovels and dredgers that can be used to deepen the fundament and to rebuilt the house if required. What can be provided and what is needed is active support in *context explication* and in *changing knowledge or its underlying context assumptions* in the case we have to adapt it to a new context (i.e., we got aware that it does not fit well to a context). This are precisely the goals of our contribution:

- First, we will discuss a method that allows to explicate hidden assumptions of a knowledge model and
- second, we will discuss a method that allows the adaptation of a knowledge model to a changed contexts by transforming the vocabulary and granularity of the model.

The context dependency of knowledge models is more than just a “philosophical“ problem. First, when developing a knowledge model for a single application the developer may have a good intuition about which assumptions can be made to deal with his problem adequately. In this case, hidden assumptions get aware in cases, where the system fails. The KBS may not be able to proceed a given input or it returns a result that is not the solution as it is required. Using error situations and system breakdowns is the most common (implicit) search method for assumptions. However, this “method“ may also cause significant damage. Reliable systems require the explication of their assumptions as explicit part of their development process. Second, contexts have the problematic feature that they change over time. As a consequence the KBS must be maintained [Men97]. The notion of the context can be used as a guideline for this process to answer the questions whether it is necessary to change the system and how this can be done without losing other necessary properties of it. Third, the problem of context-dependency is immediately present for knowledge models that should be sharable and reusable. In this case they cannot be designed to intuitively fit to a given context because they should be applicable to a broad range of problems not known beforehand.

In this paper we do not investigate knowledge models in general. Instead we focus our attention on a specific knowledge type that is called *problem-solving methods*. The concept of *problem-solving method* (PSM) is present in a large part of current knowledge-engineering frameworks (e.g. Generic Tasks [CJS92]; Role-Limiting Methods [Mar88], [Pup93]; CommonKADS [SWA+94]; the Method-To-Task approach [EST+95]; Components of Expertise [Ste90]; GDM [THW+93]; and MIKE [AFS96]).

In general a PSM describes in a domain-independent way which reasoning steps and strategy (i.e., control) and which types of knowledge are needed to perform a task. In addition, [vdV88] and [AWS93] define the competence of a PSM in addition to the description of the reasoning behaviour. The competence of a PSM should describe which goals a method can achieve without referring to *how* these goals are achieved. Libraries of PSMs are described in ([Ben95], [BvV94], [CJS92], [MZ96]), and [Bre94] describes a library of problem types. PSMs are used in a number of ways in knowledge engineering: as a guideline to acquire problem-solving knowledge from an expert, as a description of the main rationale of the reasoning process of the expert and the KBS, as a skeletal description of the design model of the KBS, and to enable flexible reasoning by selecting methods during problem solving.

On the one hand, there seems to be a strong consensus on the usefulness of PSMs as knowledge level models of reasoning processes. On the other hand, there is still no solid theoretical background in characterising the precise competence of PSMs and in providing guidelines for developing reusable PSMs and for adapting these PSMs to application specific circumstances. In [Fen95a] we wanted to specify the competence of the PSM *propose & revise for parametric design* [SB96]. However we run into two significant problems:

- There is not only one *propose & revise* method but there is a large number of slightly different variants and there is no justification to select one of them as the golden standard.
- Each of this variants uses slightly different assumptions about what the precise problem is and about the strength of the domain knowledge that it can use for its reasoning process. In general, the competence of a PSM cannot be described independent of these assumptions.

One could wonder whether it would be a solution to choose the variant of a PSM making as less

assumptions as possible. However, this misses the essence of PSMs. In general, most problems tackled with KBSs are inherently complex and intractable (cf. [Byl91], [BAT+91], [Neb96]). Efficient reasoning is only possible by introducing assumptions. These assumptions are necessary to reduce the complexity of the reasoning task [FS96] and the complexity of the development process of the reasoning system [HS96]. They either formulate requirements on domain knowledge used by the PSM or restrictions on the size of the problem that is solved by the PSM [BFS96]. Therefore, *making assumptions for efficiency reasons* is an essential feature of PSMs. As more assumptions a PSM makes as more efficient reasoning power it can provide.

There are two consequences that have be drawn for PSMs. There is neither a golden standard for a PSM nor can we expect that we will ever find a useful assumption-free method. Therefore, what we need to provide are generic schemes of PSMs and methodological support in adapting such schemes to problem-specific and domain-specific circumstances. Such a support has to deal with two aspects (see Figure 1):

- We have to provide a method that can be used to make context dependency explicit.
- We have to provide a method that supports adaptation of a PSM to a given context.

In the following, we will sketch our contribution to both requirements on a development method for PSMs.

2.1 Context Explication

We already argued that making strong assumptions on available domain knowledge and problem restrictions is not a bug but a feature of PSMs that should enable efficient reasoning. [FB96] reviews the work done on PSMs in model-based diagnosis for their assumptions. We collected a large number of assumptions and related them to the different subtasks of diagnostic problem solving. This assumption list can be used to check given domain knowledge, to define goals for knowledge acquisition, to restrict the size of the problem that must be solved by the KBS or to select a PSM that fits to the given context as characterized by the established assumptions. However, what we did was a kind of post-analysis. We collected the results of an around twenty years long common research project. Searching for hidden assumptions was a kind of implicit activity of this community during developing problem solvers for diagnosis. In this paper, we will present a method that allows a pre-analysis for hidden assumptions. We provide support for an active and explicit search process for these assumptions (see section 3).

2.2 PSM Adaptation

The context problem of PSMs were already encountered during the early work on *role-limiting methods* [Mar88] and *generic tasks* [Cha86]. The implemented methods fit well for the application the were developed for. However, trying to apply them to similar problems with slightly different goals and domain knowledge failed because of the brittleness of these methods. [KBD+91] rephrased this as the usability-reusability trade-off of PSMs. On the one hand as more assumptions and commitments to a specific problem type are made as stronger is the support of the method in developing a solution for this problem. On the other hand as more commitments to the specifity of the problem are made as less reusability could be expected.

The *mismatch* problem of PSMs and problem types is another way how context-dependency of PSMs are realized. The literature on knowledge engineering is full of examples that show that different methods can be applied to solve the same problem and a PSM can be applied to solve very different

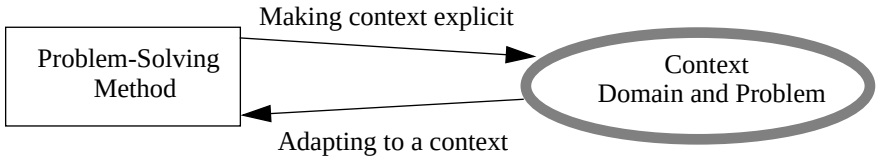

Fig. 1. The two directions of the context problem.

types of tasks. A method like *propose & revise* was developed for parametric design problems but there is no real reason why it should not be applicable to more difficult design problems or, for example, planning tasks. Recently, [vHA96] and [BBH96] have proposed that PSMs should be described not only in a domain-independent, but also in a complete task-independent way, so that they can become more broadly reusable. The mismatch of problems and PSMs can be solved in this way. However, this decontextualization of PSMs significantly reduces the usefulness of these methods that they usually provide with their task-specific vocabulary, assumptions, and granularity. In their approach, there remain only abstract algorithmic schemas without any relation to the problem that can be solved with it. As a consequence, the mapping between such an algorithmic schema and a problem-, domain-, and application-specific problem solver is immense complex and little support in knowledge acquisition is provided.

All these discussions deal with the problem of adapting a PSMs to a new or modified context. In this paper we will show methodological support for this process. We show a principled way of developing and adapting PSMs. This is mainly achieved by using *adapters* ([FSG+96], [FG97]) as a means to express the adaptation of PSMs. Refined versions of PSMs are achieved by combining them with adapters.Therefore PSMs can be used and reused at an arbitrary level of contextualization.

3 Making a Context Explicit: The Assumption Detector

> *"Science does not rest upon solid bedrock. The bold structure of its theories rises, as if were, above a swamp. It is like a building erected on piles. The piles are driven down from above into the swamp, but not down to any natural or 'given' base; and if we stop driving the piles deeper, it is not because we have reached firm ground. We simply stop when we are satisfied that the piles are firm enough to carry the structure, at least for the time being." [Pop59]*

Our method consists of two main steps: (1) Establishing a notion of the competence of a method in dependence of assumptions and (2) relating a competence of a method with a problem definition that introduces a general notion in which the PSM can be applied. Both steps are assumption search and construction processes and both rely on the *same principle*, however play *different conceptual roles* and require *different techniques*. We use failed proof attempts as a search method for assumptions and analysis of these failures for constructing and refining assumptions. A mathematical proof written down in a text book explains why a lemma is true under some preconditions (i.e., assumptions and other sublemmas). The proof establishes the lemma by using the preconditions and some deductive reasoning. Taking a view at the proof *process* we get a different picture. Usually first proof attempts fail running into improvable subgoals. These stucked proof attempts point to necessary features that are not present from the beginning. Actually they make aware of further assumptions that have to be made in order to succeed the proof. Taking this perspective a proof process can be viewed as a search and construction process of assumptions. Gaps that can be found in a failed proof provide already first characterizations of missing assumptions. They appear as sublemmas that were necessary to proceed with the proof. An assumption that implies such a sublemma closing the gap in the proof is a possible candidate we are looking for. That is, formulating this sublemma as an assumption is a first step in finding and/or constructing assumptions that are necessary to ensure that a PSM behaves well in its context. Using an open goal of a proof directly as an assumption normally leads to very strong assumptions. That is, these assumptions are *sufficient* to guarantee the correctness of the proof, but they are often neither *necessary* for the proof nor *realistic* in the sense that application problems will fulfil them. Therefore, further work is necessary to find improved characterizations for these assumptions. This is achieved by a precise analysis of their role in the completed proof to retrace unnecessary properties.

Such proofs can be done semiformal in a textbook style as proposed by *Evolving algebras*[2] community

2. Meantime also called Abstract State Machines (ASMs).

[Bör95], a recent specification framework in software engineering. However, providing specification formalisms with formal syntax and formal semantics allows (semi-)automated proof support. The large amount of details that arises when proving properties of software (and each PSM finally has to be implemented) indicates the necessity of such mechanization. Therefore, we provide a formal notion for PSMs and semi-automatic proof support by adapting the Karlsruhe Interactive Verifier (KIV) [Rei95] for our purpose.[3] KIV was originally developed for the verification of procedural programs but it can also be applied to formal specifications of KBSs (see [FS97b]). Three main reasons justify our choice:

- KIV integrates specification of dynamics in dynamic logic with functional specifications in an abstract data type style. Therefore, KIV can express the dynamic of a reasoning process as well as its static competence.
- KIV provides structuring operations for specification that can be used to represent tasks, PSMs, adapters, domain models and their relationships.
- KIV uses an *interactive* tactical theorem prover that makes it suitable for hunting hidden assumptions. We expect many proofs to fail. Using a theorem prover that returns with such a fail adds nothing. Instead of returning with a failure KIV returns with open goals that could not be solved during its proof process. In addition, KIV provides support in counter examples generation. These are precisely the kind of support we are looking for finding and constructing assumptions.
- Further important support is provided by correctness management and reuse facilities. Because the development process of the appropriate PSM and its assumptions is an iterative and reversible process, one has to keep track of (repeated) changes of lemmas, assumptions and proofs. Support in generating and managing different proof obligations and reuse of proofs for slightly modified specification, is essential in making proofs practicable.

During the following, we will illustrate our method by discussing small toy examples that present our ideas clearly and that are easy to understand. First, we present *hill-climbing* and relate the competence of this local search method with a problem of finding a global optimum. Second, we discuss a simple version of abductive diagnosis and the kind of assumptions that can be found when trying to solve this problem with a local search method. The reader may argue that we do not discuss PSMs for KBS but simple search methods. However, we would like to make three points:

- The algorithmic core of PSMs are simple search methods. Take *propose & revise* ([MM89], [SB96]) as an example. It is a simple local search method with two different modes: Proposing extensions of a state and revising a state if constraints violations occur. Therefore, our results can immediately applied to this type of methods.
- PSMs gain their “intelligence“ by using strong domain heuristics for the search process, by restricting the size of the problem they deal with properly, and by ontological commitments that make them immediately applicable to a specific problem type. In section 4 we will explain how the search methods that we discuss can be adapted to richer contexts constituting task-specific (or problem-type specific) PSMs.
- In the following we will derive rather strong assumptions providing local search techniques with the power to solve global search problems. However, we will sketch in the last part of section 3 how more realistic assumptions reflecting the heuristic nature of PSMs can be found.

3.1 First Example: A Local Search Method

Our methods consists of two main steps: (1) Establishing a notion of the competence of a method in dependence of assumptions and (2) relating a competence of a method with a problem definition that introduces a general notion in which the PSM can be applied. Each step uses the same principle but rely on a very *different conceptual and technical* background. Establishing a competence of an operational algorithm specification requires proof techniques of dynamic logic. We have to relate a procedural specification with a first-order specification of its assumptions and competence. For the second step we have to relate two declarative specifications. The gap between a competence specification and a

3. This formal framework is used in the paper but is not its subject, cf. for more details [FG96], [FG97], [FS97b].

problem definition has to be closed via assumptions. The specifications and proofs remain within first-order logic by proving equivalence of first-order formulas.

3.1.1 Establishing a Competence of a PSM in Dependence on Assumptions

Hill-climbing is a local search algorithm that stops when it has found a local optimum. The control flow is defined in Figure 2. The method works as follows: First, we select a start object. Then we recursively generate the successors of the current object, and select a successor if we find a better one. Otherwise we terminate and return the current object that does not have better successors. The functions *select-start*, *generate*, and *select-a-best* correspond to elementary inference actions in CommonKADS [SWA+94].

The main requirements on *domain knowledge* that are introduced by *hill-climbing* (and by other local search methods for an optimum) is the existence of a *preference* relationship and a *successor* relationship between the objects. The former is used for selection and the latter is used to enable the local search process. A third requirement is a selection criterion for the start object of the search process. The performance and competence of the method depends on the properties of these three relations.

With KIV, we tried to prove that *hill-climbing* always terminates and that it has the competence to find a local optimum (see Figure 3).[4] KIV automatically generates all proof obligations in dynamic logic that are necessary to ensure termination and competence of an algorithmic specification. In our case it generates the following proof obligations (see [FS97b]):

$\vdash$ <*hill-climbing*(*input*)> true, i.e. termination

$\vdash$ <*hill-climbing*(*input*)> $output \in input$

$\vdash$ <*hill-climbing*(*input*)> $\neg\exists x\,.\,(successor(output,x) \wedge x \in input \wedge output < x)$.

```
operational spec hill-climbing
    hill-climbing(input)
        begin
            current := select-start(input);
            output := recursion(current)
        end
    recursion(X)
        begin
            successors := generate(X);
            new := select-a-best(X,successors)
            if X = new
                then output := X
                else recursion(new)
            endif
        end
    /* select-start must select an element of input and uses a selection criterion. */
        select-start(x) ∈ x ∧ select-start(x) ∈ select-criterion(select-start(x),x)
    /* generates selects input elements that are in successor relation with the current object.*/
        x ∈ generate(y) ↔ x ∈ input ∧ successor(y,x)
    /* select-a-best selects the current object if no better successors exist or a successor if a better successor
    exists. In the latter case the selected successor must be better than the current object and there need not to be
    another successor that is better than the selected successor. */
        ¬∃z . (z ∈ {y} ∪ y´ ∧ select-a-best(y,y´) < z)
        ¬∃z . (z ∈ y´ ∧ y < z) → select-a-best(y,y´) = y
        ∃z . (z ∈ y´ ∧ y < z) → select-a-best(y,y´) ∈ y´ ∧ y < select-a-best(y,y´))
endoperational spec
```

Fig. 2. The operational specifiction of of *hill-climbing*.

4. See http://www.aifb.uni-karlsruhe.de/WBS/dfe/ijcai.html and http://www.aifb.uni-karlsruhe.de/WBS/dfe/eurovav.html for details of the proof processes.

competence *hill climbing*
input requirement
$\exists x.\ (x \in input \wedge x \in select\text{-}criterion)$
knowledge requirement
$\neg(x < x)$
$x < y \wedge y < z \rightarrow x < z$
post condition
$output \in input$
$\exists x\ .\ (x \in input \wedge select\text{-}criterion(x, input) \wedge (x < output \vee x = output))$
$\neg\exists x\ .\ (successor(output, x) \wedge x \in input \wedge output < x)$
endcompetence

Fig. 3. The competence theory of *hill-climbing*.

Tool support is provided in unfolding these proof obligations and in applying proof tactics like inductive proofs. In general the user has to select proof tactics and heuristics from a menu. In our case, user interaction is needed to select axioms of a specification as supporting lemmas for the proof and in selecting the kind of induction (see [FS97b]). We run into a number of open goals during the proofs:

- We had to ensure that *select-criterion* retrieves an object for each possible input. Otherwise we cannot guaranty that *hill-climbing* provides any output.
- We had to ensure that the preference we use (<) is a *partial order.* We have to be sure of irreflexivity and transitivity to ensure that hill-climbing cannot be catched in circles (imagine $a < b$ and $b < a$ and $a \in successor(b)$ and $b \in successor(a)$). Also we have to ensure finiteness of the input set.

Based on these assumptions that were necessary to complete the proofs we could establish a competence of *hill-climbing* as shown in Figure 3. We can ensure that *hill-climbing* finds a *local* optimum of the input.

To give an impression of how to work with KIV, Figure 4 is a screen dump of the KIV system when proving the competence of *hill-climbing*. The *current proof* window on the right shows the partial proof tree currently under development. Each node represents a sequent (of a sequent calculus for dynamic logic); the root contains the theorem to prove. In the *messages* window the KIV system reports its ongoing activities. The *KIV-Strategy* window is the main window, which shows the sequent of the current goal, i.e. an open premise (leaf) of the (partial) proof tree. The user works either by selecting (clicking) one proof tactic (the list on the left) or by selecting a command from the menu bar above. Proof tactics reduce the current goal to subgoals and thereby make the proof tree grow. Commands include the selection of heuristics, backtracking, pruning the proof tree, saving the proof, etc.

3.1.2 Relating the Competence of a PSM with a General Problem Definition

A *problem definition* specifies the *goals* that should be achieved by the KBS. It establishes an explicit notion of the context the PSM is applied in. A general context *hill-climbing* can be applied in is the search for a (global) optimum. Figure 5 provides the definition for our running example. The goal describes what an optimum must fulfil. However, the PSM *hill-climbing* has only the competence to find a local optimum in a graph. Again, we start the interactive proof process knowing that it will lead us to further assumptions because in general *hill-climbing* does not have the competence to find a global optimum. Two main problems arise during the proof:

1) We would have to prove that the selected start object is always connected with a global optimum (there may exist several global optima because we do not require a total order). Otherwise, the global optimum is not reachable by the recursive search of hill-climbing.
2) Even if we could prove (1) we may get stucked at the case distinction
   if $X = new$
   where hill-climbing stops for a local optimum.

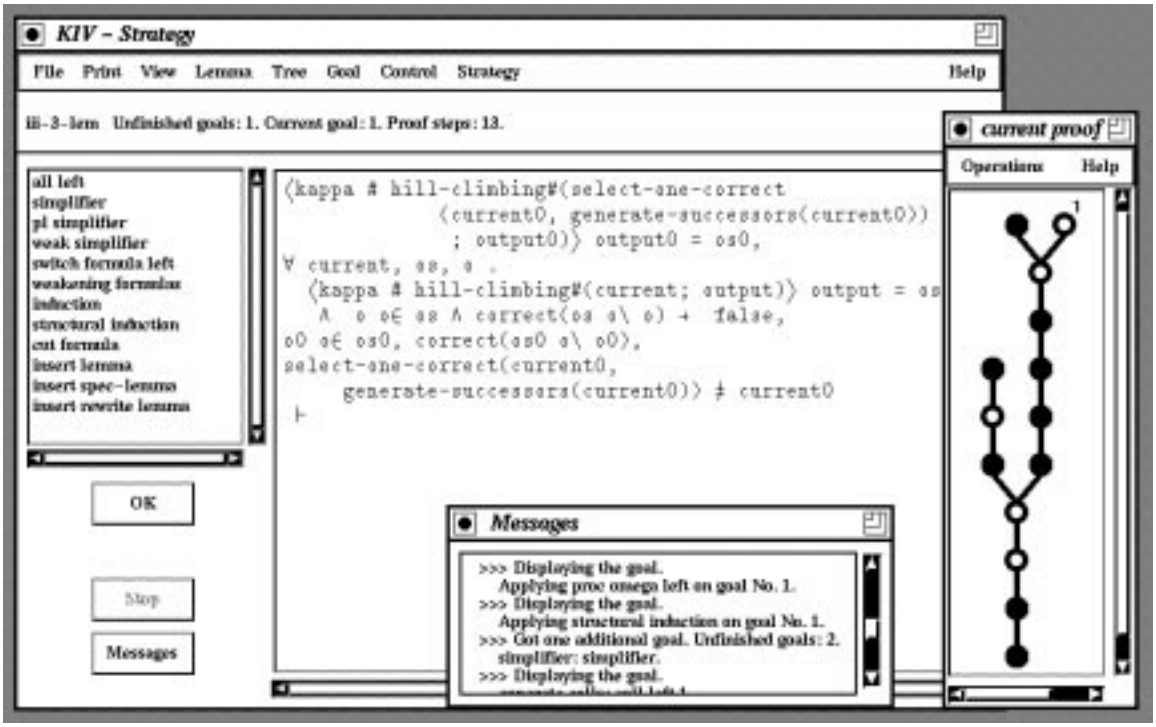

Fig. 4 Verifying the PSM with KIV.

A trivial assumption that close both gaps in the proof is to require that each object is directly connected with each object.

totally-connected assumption: *successor*(*x*,*y*)

However this is not a very meaningful assumption. In this case, *hill-climbing* collapses to a complete search in one step because all objects are successors of each possible start object. A less drastic assumption is to require that each object (except a global optimum) has a successor with a higher preference.

better-successor assumption:

$$x \in input \rightarrow (\exists y\,.\,(y \in input \wedge successor(x,y) \wedge x < y) \vee \neg\,\exists z\,.\,(z \in input \wedge x < z))$$

This assumption is derived to close the gap in the case distinction of the recursion. If the recursion stops we have found a global optimum. We already know from the termination proof of *hill-climbing* that the PSM always terminates.

The question remains whether the assumptions is *minimal*. Here, *minimality* means that the assumptions is not only sufficient but also *necessary* to guarantee that the competence of the PSM implies the problem definition, formally,

$$(PSM_{competence} \rightarrow Problem\ Definition) \rightarrow Assumption$$

An assumption that is *minimal in the logical sense* (i.e., necessary) has the clear advantage that it maximizes the circumstances under which it holds. It does not require anything more than what is precisely required to close the gap between competence and problem. In fact, we have proven with KIV that the *better-successor assumption* is a minimal assumption in the logical sense.[5] However, besides

task *global optimum*
goal
$global\text{-}optimum \in input$
$\neg\exists x\,.\,(x \in input \wedge global\text{-}optimum < x)$
endtask

Fig. 5. The problem definition *global optimum*.

logical minimality other aspects like cognitive minimality (effort in understanding an assumption) or computational minimality (effort in proving an assumption) may influence the choice of assumptions, too. We will illustrate this in the following subsection. In general, minimizing (i.e., weakening) of assumptions can be achieved by analysing their sufficiency proof with KIV and eliminating aspects that are not necessary for continuing the proof.

It was also easy to prove with KIV that the *better-successor assumption* is weaker[6] than the *totally-connected assumption* but the former has the disadvantage that it is not only formulated in terms of domain knowledge but also in terms of the current case input (we have to ensure that the local better successor is always element of the input). Therefore, whether this assumption holds cannot be proven statically independent from the actual input.

The two assumptions that we have introduced yet are rather trivial. This is a consequence of our simplistic example. In the following section we will define a more complex task and PSM which will lead to more interesting assumptions.

3.2 Second Example: Finding an Abductive Explanation

In the following, we introduce a more complex task definition. We ask for explanations, where an explanation is a set of hypotheses that explains a set of observations. That is, we define a typical abductive problem.

3.2.1 The Task of Finding Complete and Parsimonious Explanations

[BAT+91] analyse the computational complexity of abduction. They define an abduction problem by a set of input data that must be explained and a set of hypotheses that can be used to construct explanations. A *complete explanation* must explain all input data (i.e., *observations*) and a *parsimonious* explanation must be minimal (that is, no subset of it explains all *observations*). Figure 6 provides the problem definition for our new example.

3.2.2 The Set-Minimizer Method

We use the simple PSM *set-minimizer* of [FG97] for our example. It receives a set of objects as input and tries to find a minimized version of the set that still fulfils a correctness requirement. The applied search strategy is one-step look ahead. The PSM-specification is provided in Figure 7. The main requirement on available knowledge and input is the existence of a predicate *correct* holding true for the input set. The method works as follows: First, we take the input. Then we recursively generate the *successors* of the current set and select one of its correct successors. If there is no new correct successor we return the current set. The *competence* states that *set-minimizer* is able to find a *local* minimal subset of the given set of objects. The three axioms state that it (1) finds a subset that is correct (2), and (3) each set containing one element less is not a correct set.

We skip all proofs that were necessary to establish this competence. Actually we could reuse the proofs that were done for hill-climbing based on our *adaptation method* (see section 4) and the proof reuse facilities of KIV. By providing a library of reusable PSMs and support in adaptation, their proofs can be reused, too. Therefore, this type of proofs that an operational specification of a PSM has some

task *complete and parsimonious explanation*
$goal(x) \leftrightarrow complete(x) \wedge parsimonious(x)$
$complete(x) \leftrightarrow expl(x) = observables$
$parsimonious(x) \leftrightarrow \neg\exists x'.\ (x' \subset x \wedge expl(x) \subseteq expl(x'))$
endtask

Fig. 6. The problem definition for *abduction*.

5. This proof process leads to several refinements of the original assumption as we encountered several wholes during the proof process.
6. *A* is *weaker* than *B* iff $B \models A$.

```
operational spec set-minimizer
    output := set-minimizer(input)
    set-minimizer(X)
        begin
            successors := generate(X);
            if ¬ ∃x . (x ∈ successors ∧ correct(x))
                then output := X
                else
                    new := select-a-correct(successors)
                    set-minimizer(new)
            endif
        end
    /* generate creates subsets that contain one element less.*/
        x ∈ generate(y) ↔ ∃z . (z ∈ y ∧ x = y \ {z})
    /* A selected successor has to be correct. */
        ∃x . (x ∈ y ∧ correct(x)) → (correct(select-a-correct(y)) ∧ select-a-correct(y) ∈ y)
endoperational spec
competence set-minimizer
        input requirement
            correct(input)
        post condition
            (1) output ⊆ input,
            (2) correct(output),
            (3) x ∈ output → ¬correct(output \ {x})
endcompetence
```

Fig. 7. The specifiction of *set-minimzer.*

competence (related to a fixed set of assumptions) has to be done only once when introducing the PSM into the library.

3.2.3 Relating Problem and Method

Problem and PSM become connected by providing the set of all hypotheses as input and identifying correct sets with complete explanation, i.e. *input* := {*x* | *x* is a hypothesis} and *correct*(*x*) ↔ *complete*(*x*).

Again we have to investigate the circumstances that ensure that the PSM achieves the goal as introduced by the problem definition. We use the same procedure again. We try to prove:

goal(*output*)

This immediately splits into two subgoals:

- *complete*(*output*)
- *parsimonious*(*output*)

Completeness. The completeness of output follows directly from axiom (2) of the competence of the PSM (see Figure 7). However, it is based on the input requirement of the method. We have to strengthen the problem definition of abduction. The set of all hypothesis must be a complete explanation. This puts a strong restriction on the abductive problem: The function *expl* has to be defined in a way that adding hypotheses to a set of hypotheses must not destroy the explanatory power of the set. Actually, we will establish the latter property during the proof of parsimony.

Parsimonity. The competence of our method ensures local parsimonity. Our method *set-minimizer* finds a local-minimal set that is parsimonious in the sense that each subset that contains one element less is not a complete explanation. However, we cannot guaranty that it is parsimonious in general. There may exist smaller subsets of it that are complete explanations. The reader may already have realized the similarity with the problem of *hill-climbing* that finds only local optimal elements. However, with the *better-successor assumption* it is able to find global optima. A natural way to close our gap is therefore to reformulate the *better-successor assumption* and *global optimum* in terms of the

new problem. Instantiating the *better-successor assumption* requires the definition of a preference and of a successor relation:

$x < y :\leftrightarrow expl(x) \subseteq expl(y)$

$successor(x,y) :\leftrightarrow \exists z\ .\ (z \in x \wedge y = x \setminus \{z\})$

The *better-successor assumption* is now reformulated as:

$x \subseteq hypotheses \rightarrow$
$(\exists y\ .\ (y \in x \wedge expl(x) \subseteq expl(x \setminus \{y\})) \vee \neg\ \exists z\ .\ (z \subseteq hypotheses \wedge expl(x) \subseteq expl(z))$

However, this is a minimal but not very intuitive assumption. Again we apply our failed-proof technique. We use it now to *generalize* an assumption. The idea is to prove that the *better-successor assumption* holds based on the problem definition, the input requirements and competence of the set-minimizer method, and the way we defined the preference. KIV returns with open goals that would be necessary to complete the proof. These open goals are generalizations of the *better-successor assumption* because they imply its truth. This proof attempt with KIV is straightforward and gets stuck in the following subgoal:

$y \subset x \wedge expl(x) \subseteq expl(y) \rightarrow \exists z\ .\ (expl(x) \subseteq expl(x \setminus \{z\})$

This assumptions requires that if there are smaller subset of *x* with larger explanatory power than there must be another subset that differs only in one element from *x* and also has larger explanatory power. A simple generalization of this implication is to negate its premise. That is, we select the strengthening tactic:

$\neg a \vdash\!\!\!- a \rightarrow b$

This tactics leads to

$\neg(y \subset x \wedge expl(x) \subseteq expl(y))$,

i.e.,

$y \subset x \rightarrow \neg\ expl(x) \subseteq expl(y)$

This assumption requires that for any set of hypotheses there may not exist a subset that has a larger explanatory power. Deleting a hypothesis from the set of hypotheses may never lead to a superset of explained observations. Actually this assumption plays a prominent role in the literature on abductive reasoning and model-based diagnosis.

A strengthened version of this assumption is used by [BAT+91] to define polynomial subclasses of abduction. In general, abduction is NP-hard in the number of hypotheses. However, with their monotonic abduction assumption

$y \subset x \rightarrow expl(y) \subseteq expl(x)$

[BAT+91] proves that it is possible to find a complete and parsimonious explanation in polynomial time. The assumption they use requires that a superset of hypotheses explains also a superset of observations. The assumption is used to restrict the worst-case effort of a method.

[KMR92] examine their role in model-based diagnosis. The assumption holds for applications, where no knowledge that constrains fault behaviour of devices is provided or where this knowledge respects the *limited-knowledge-of-abnormal behaviour assumption*. This is used by [KW87] as *minimal diagnosis hypothesis* to reduce the *average-case effort* of finding all parsimonious and complete explanations with GDE. A syntactical way to ensure this assumption (i.e., to formulate it as a requirement on the domain knowledge) is the restriction of the domain theory to Horn clauses constraining only the correct behaviour of devices, cf. [KMR92].

It is interesting to see how the very generic *better-successor assumption* transforms into such intuitive and broadly used task-specific assumptions.

3.3 Heuristic Assumptions

The assumptions introduced so far ensure that a local method solves a global problem. They ensure that a local search method has the same competence as a global search method. However, that is not what

one is usually looking for. Often these assumptions are too strong. In addition, we mentioned that GDE uses the monotonic-abduction problem to reduce the average-case behaviour of the problem solver. However, we have not provided measurements nor proofs of these aspects. In this subsection, we will sketch how both can be integrated.

3.3.1 Domain-specific Reformulation and Weakening of Assumptions

Propose & revise ([MM89], [SB96]) is a local search method consisting of two substeps: *Propose* extend a current state and *revise* modifies the state if constraint violations occur. Both activities are iterated until a complete and correct state is reached. In the general case, this cannot be guaranteed because *propose & revise* does not include any backtracking mechanism that would allow to escape a death-end of the solution process. Completeness of the search method can only be guaranteed if we introduce strong assumptions about the domain knowledge that is used by the *propose* and by the *revise* step (see Figure 8).

[ZM96] provide an interesting and in-depth analysis of the PSM *propose & revise* and its application to the vertical transportation (VT) domain ([MM89], [SB96]). The goal is to design an elevator that meets several requirements and constraints. [ZM96] identify two key parameter in this domain that influence the difficulty of the problem: the *capacity* and the required *speed* of the elevator. As larger the values of these two parameters are as more difficult it is to find a solution. They investigate how *propose & revise* behaves based on the available domain knowledge for different combinations of these two parameter values. The instantiated method failed for some of the simple cases and not surprisingly for many of the difficult ones. Some of the difficult ones could be solved by a complete search method however it required large amount of storage size and computation time. Failures of *propose & revise* in these cases can be accepted because we are looking for a heuristic problem solver that gains efficiency by restriction to the simple cases. Aiming for a complete and efficient problem solver for the general case would define an unsolvable problem. However, that *propose & revise* also fails for some of the simple cases must be viewed as gaps in the provided domain knowledge. It should be strong enough to enable *propose & revise* to find a solution for these cases.

Based on this domain analysis, we could weaken our assumptions of Figure 8. Instead of requiring that we always have a *propose* and *revise* step available that find an optimal successor state we could restrict these requirements to the more simple cases. That is, we include boundaries on the values of the key parameters in the assumptions. For difficult cases we have to use either a more complex search method with higher demands on storage and time or we have to ask the human expert to solve the tricky cases.

competence *propose & revise*
knowledge requirements
/* The propose knowledge never fails and monotonically extends the state. */
$\neg$ *Complete*(*s*) $\rightarrow$ *Partial completeness*(*s*) < *Partial completeness*(*propose*(*s*))
/* The application of a propose leads to an optimal state. */
$\neg$ *Complete*(*s*) $\rightarrow$
$\neg\exists$ *s´* . (*s´* $\in$ *State* $\wedge$ *Correct*(*s´*) $\wedge$ *propose*(*s*) < *s´* $\wedge$
Partial completeness(*s*) < *Partial completeness*(*s´*) = *Partial completeness*(*propose*(*s*))
/* The revise knowledge never fails. */
$\neg$ *Correct*(*s*) $\rightarrow$ *Correct*(*revise*(*s*))
/* The application of revise does not change the completeness of a state. */
Partial completeness(*revise*(*s*)) = *Partial completeness*(*s*)
/* The application of revise leads to an optimal state. */
$\neg$ *Correct*(*s*) $\rightarrow$
$\neg$ $\exists$ *s´* . (*s´* $\in$ *State* $\wedge$ *Correct*(*s´*) $\wedge$
Partial completeness(*revise*(*s*)) = *Partial completeness*(*s´*) $\wedge$ *revise*(*s*) < *s´*)
post condition
/* The output is a complete, correct and optimal state. */
Complete(*output*) $\wedge$ *Correct*(*output*) $\wedge$
$\neg\exists$ *s* . (*s* $\in$ *State* $\wedge$ *Complete*(*s*) $\wedge$ *Correct*(*s*) $\wedge$ *output* < *s*)
endcompetence

Fig. 8. The competence of *propose & revise* (see [FMD+]).

3.3.2 Specification and Verification including Thresholds

We motivated the assumptions of PSMs with the goal to improve the efficiency of the assumption-based reasoning process compared to a reasoning process using less assumptions. However, we have not yet provided means to specify and verify the efficiency of PSMs. [Sha89], [SB95] describe means that can be integrated into our framework. [Sha89] includes counters and boundaries for the values of these counters into the specification of real-time software. Precisely the same can be done for PSMs. In the case of *hill-climbing* we can add a counter for the number of successors that are derived and compared in one step and a second counter on the number of steps. Then, we can formulate boundaries for their combined value and formulate this in terms assumptions that either introduce requirements on the domain knowledge or restrict the set of problems that are solved by *hill-climbing* (i.e., *hill-climbing* terminates after it has consumed its computational time).[7]

In general, such refinements of specifications by measurements allow more refined assumptions on the search graph that is used by the local search methods. [Ste95] provides informal examples for such refined assumptions for different variants of different PSMs in what he calls symbol-level analysis of PSMs. For example, he discusses for classification methods under which circumstances data-directed search, solution-directed search and opportunistic search are preferable. Each search type requires some knowledge types but more important some properties of this knowledge in order to perform well (i.e., effectively *and* efficiently).

3.4 Weakest Preconditions

Above we discussed two roles of assumptions. First, they are necessary to ensure that a PSM has a specific competence. For example, without the assumption that the preference relation is a partial order the termination proof of hill-climbing would not be possible. Second, they are necessary to ensure that the competence of a PSM is able to achieve a goal as introduced by a problem definition. For example, with the *better-successor assumption* we can prove that the competence of *hill-climbing* is strong enough to find a global optimum. Both roles differ conceptually and technically.

The first aspect has to be examined when establishing a PSM in a library of reusable elements. Such elements must be reliable in the sense that they guarantee some competence and the conditions necessary to provide this competence. Technically, it is a proof that concerns the algorithmic structure of the method. Therefore, we use dynamic logic to specify the algorithm and the proof obligations are formulas in dynamic logic.

Assumptions → <Operational Specification> true

Assumptions → [Operational Specification] Competence

Finding weakest preconditions for an algorithm has a long tradition in software engineering (cf. the wp calculus [Dij75] and predicate transformers [CS95]). Since the dynamic logic we use can be regarded as a generalization of Hoare-triples we can employ methods and techniques developed in this area (cf. e.g. the B-Toolkit [Wor96] or Z/EVES [MS96]).

However, the second aspect has a different purpose and requires different techniques. Here we are looking for assumptions that ensure that the competence of the method implies the problem definition, i.e.

Assumptions → (Competence → Problem Definition)

That is, we do not need to refer to the algorithmic structure of the PSM and neither specification nor proof rely on dynamic logic. We relate two declarative specifications, the functionality of the system and the specification of the required functionality. Assumptions are used to split the required functionality into two parts. One part that is solved by the competence of the PSM (in the case the assumptions holds) and one part that is only assumed to be solved. That is, this part is either be solved by the domain knowledge, by an external possible human agent, or it must be viewed as a restriction of the class of solvable problems. We argued that our PSMs can only provide a limited fragment of the entire functionality because of the intractability of typical problems they are applied to. In software

7. Using this direction, work on *anytime* algorithms [Zil96] could be integrated into the work on PSMs.

engineering, usually a different point of view is taken. One assumes that the specification of the functionality of the system is also the specification of the required functionality. In this setting, our assumption hunting method makes no sense because the distinction between the two different specifications and their relationship are not present.

4 Switching a Context: The Tower-of-Adapter Method

In the following, we show a principled way of developing and adapting PSMs to different contexts. This is mainly achieved by using *adapters* as a means to express the refinement of PSMs. *Adapters* were originally introduced in [FSG+96],[FG97] to allow the independent specifications of problem definitions, PSMs, and domain knowledge. Building KBSs from reusable elements requires adapters that properly link these elements and adapt them to the application-specific circumstances. Because these elements should be reusable, they must abstract from application-specific circumstances and because they are specified independently from each other there is a need to introduce their mappings. Introduced as glue that brings other elements together they will now play a central role in refining PSMs. Actually, a refined version of a PSM is achieved by combining it with an adapter.

The stepwise introduction of adapters can be used to stepwise refine generic PSMs. Three processes are supported by our approach:

- the *terminological structure of states* of a method can be refined by introducing ontological commitments;
- the terminological structure of states can be used to refine *state transitions* of a method; and
- *assumptions* can be introduced to link the competence of a method with problem definitions and domain knowledge.

Again we use simple and self-containing examples to illustrate our ideas. First, we sketch the general specification framework. Then we introduce a generic algorithmic schema for local search. First this schema becomes refined to *hill-climbing* via an adapter. A second adapter is used to refine *hill-climbing* into the *set-minimizer* method. The general search strategy always remains the same but the ontological commitments of the methods become refined. States and state transitions are described with an enriched vocabulary. A further adapter transforms this method into a method for abductive diagnosis (by adding additional ontological requirements). Besides adding ontological commitments, it is necessary to add assumptions to close the gap between a method and a problem. We discuss adapters that ensure that these methods can be applied to problems that define a global optimum. We show how this leads to a refinement of problem definitions and assumptions similar to the refinement of PSMs. Finally, we discuss the refinement of control.

4.1 The Four Component Model

[FSG+96],[FG97] propose a four component model for the specification of KBSs built up from reusable elements. (see Figure 9): a *problem definition* that introduces the problem that should be solved by the KBS; a *PSM* that defines the reasoning process of a KBS; and a *domain model* that describes the domain knowledge of the KBS. A fourth element of a specification of a KBS is an *adapter* that is necessary to adjust the three other (reusable) parts to each other and to the specific application problem. It is used to introduce assumptions and to map the different terminologies.

The Problem Definition. The problem definition specifies *goals* that should be achieved in order to solve a given problem. A second part of this specification is the definition of *requirements* on domain knowledge. For example, a task that defines the selection of the maximal element of a given set of elements requires a preference relation as domain knowledge. Axioms are used to define the requirements on such a relation (e.g. transitivity, connexitivity, etc.).

The Problem-Solving Method. The description of the *reasoning process* of a KBS by a PSM consists of three elements in our framework: a competence description, an operational specification, and requirements on domain knowledge. The definition of the functionality of the PSM introduces the *competence* of a PSM independent from its dynamic realization. An *operational description* defines the

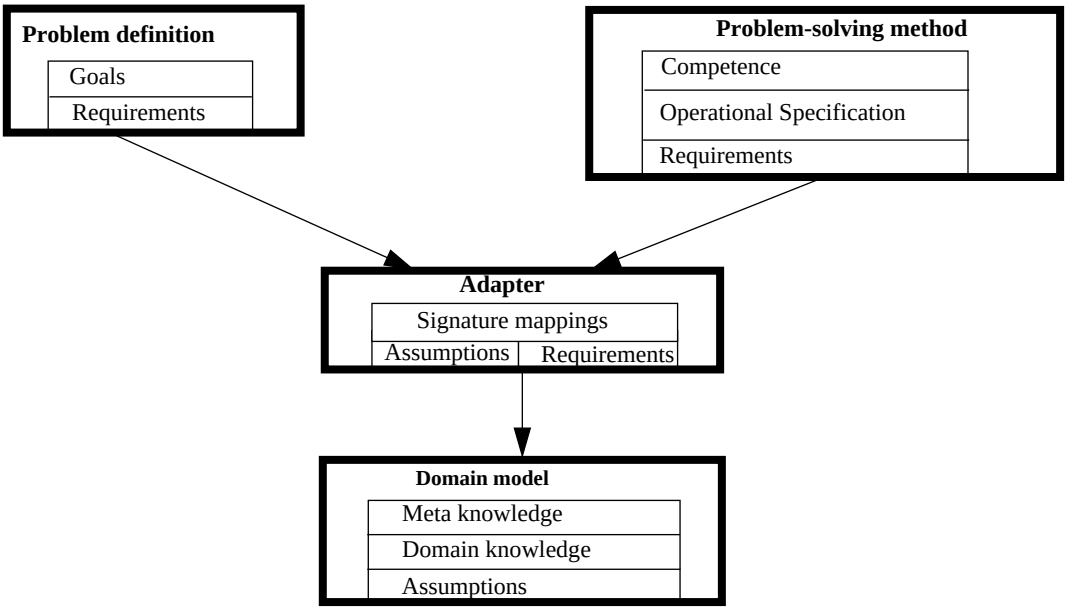

Fig. 9 The four elements of a specification of a KBS.

dynamic reasoning of a PSM. Such an operational description explains how the desired competence can be achieved. It defines the main reasoning steps (called *inference actions*) and their dynamic interaction (i.e., the knowledge and control flow) in order to achieve the functionality of the PSM. The third element of a PSM are *requirements* on domain knowledge.

The Domain Model. The description of the *domain model* introduces the domain knowledge as it is required by the PSM and the task definition. Our framework provides three elements for defining a *domain model*: a meta-level characterization of properties, the domain knowledge, and assumptions of the domain model. The *meta knowledge* characterizes properties of the domain knowledge. It is the counter part of the requirements on domain knowledge of the other parts of a specification. The *domain knowledge* is necessary to define the task in the given application domain and necessary to proceed the inference steps of the chosen PSM. *External assumptions* relate the domain knowledge with the actual domain. They can be viewed as the missing pieces in the proof that the domain knowledge fulfil its meta-level characterizations.

The Adapter. The description of an *adapter* maps the different terminologies of task definition, PSM, and domain model and introduces further requirements and assumptions that have to be made to relate the competence of a PSM with the functionality as it is introduced by the problem definition (cf. [Fen95a], [BFS96]). Because it relates the three other parts of a specification together and establishes their relationship in a way that meets the specific application problem they can be described independently and selected from libraries.

4.2 Refining Problem-Solving Methods

We start describing the development process of PSMs with a very generic search schema. During the following sections this schema will become refined. However, we do not make a commitment to this top-down like development process. The process can start at any level of specialization and can take the direction of specialization or generalization because we provide a library containing these generic schemas and their adaptations. *Specialization* is achieved by adding an adapter to an existing PSM-adapter combination and *generalization* is achieved by deleting an adapter from an existing PSM-adapter combination.

[SL90] present a theory of search algorithms to support the transformation of problem definitions into implementations. Figure 10 shows their hierarchy of search methods providing local search as an instance of *generate & test* like approaches working on local structures. The general algorithmic structure of a local search graph can be described by an initialization and a recursion that parses the

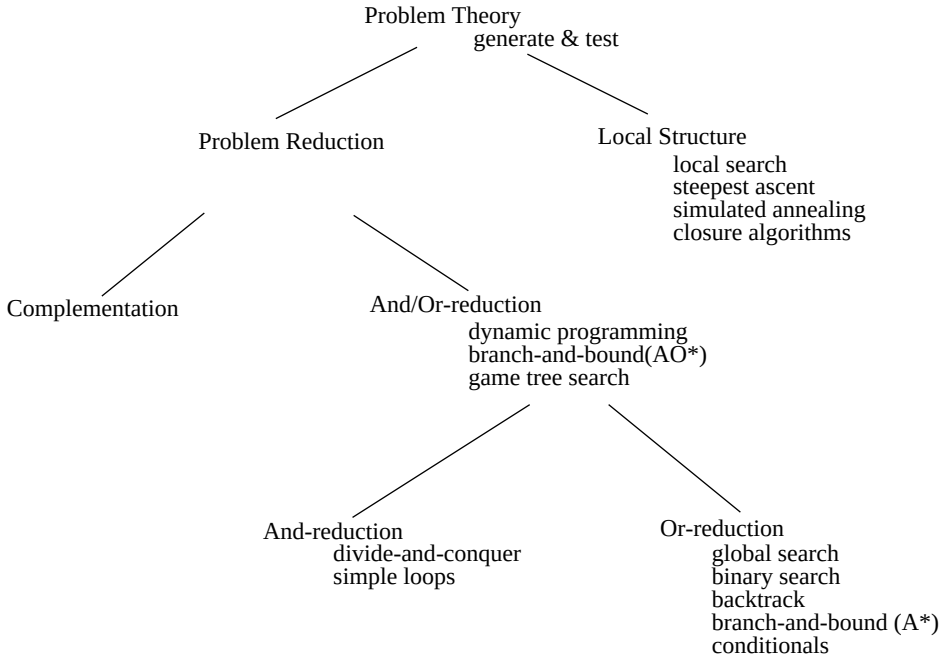

Fig. 10. Refinement hierarchy of algorithm theories [SL90].

local search structure. Figure 11 provides the definition of such a generic search strategy. It can be used to describe breadth-first search, depth-first search, hill climbing, beam search etc. This generic specification will be the backbone of all our examples in the following. All other refined versions will be achieved by combining it with adapters.

4.3 Hill-Climbing

Figure 2 already provided an operational specification of *hill-climbing*. However, we can achieve the same effect by defining an adapter for the generic local search schema of Figure 11. This adapter has to refine the definitions of the elementary state transitions (i.e., inference actions in terms of CommonKADS) of the method. Axioms have to be added to the definitions of these relations. The adapter that achieves this refinement is provided in Figure 12.

Hill-climbing refines the generic local search strategy. The output is a local optimal element, however, this method is still very generic and can be applied to nearly any type of task. In the next step we will specialize this method to the *set-minimizer method*.

4.4 Set-minimizer

We already presented *set-minimizer* that can be used to find a minimal but still correct subset of a given set (see Figure 7). It returns a correct set that is local minimal in the sense that there is no correct subset that has one element less. This method is obviously a local search method specialized for a specific type of problems. *Set-minimizer* refines *hill-climbing* with the following refinements:

- An generic object of *hill-climbing* is a set in *set-minimizer*. That is, *set-minimizer* adds additional ontological commitments used to characterize states of the search process.
- The successor relationship is hard-wired in *set-minimizer*. A set is a successor of another set if it is a subset with one element less. The ontological commitment used to characterized states is used to refine the definition of state transitions.
- A preference on entities is defined implicitly. Smaller sets are preferred if they are still correct.

Set-minimizer describes only one of several possible problem-specific adaptations of *hill-climbing*. Traditionally for each variant the specification has to be re-done, all termination and correctness proofs of the method have to be re-done, and the method has to be re-implemented. Our approach provides

operational specification *local search*
output := local search(*input*)
local search(*X*)
begin
currents := $select_1(X)$;
output := *resursion*(*currents*)
end
recursion(*X*)
begin
successors := *generate*(*X*);
new := $select_2(X,successors)$
if $X = new$
then *output* := $select_3(X)$
else *recursion*(*new*)
endif
end
/* $select_1$ must select elements of *input*. */
$select_1(x) \subseteq x$
/* *generates* selects input elements that are in successor relation with the current objects.*/
$x \in generate(y) \leftrightarrow x \in input \wedge \exists z\ .\ (z \in y \wedge successor(z,x))$
/* $select_2$ selects a subset of objects from the union of two sets. The new object set must be constructed in a way that there remain no better objects unselected. That is, if we select x and there is a y that is better than x we also have to select y. $select_2$ selects at least one object. */
$select_2(y,y') \subseteq y \cup y'$
$\neg\exists x,z\ .\ (z \in ((y \cup y') \setminus select_2(y,y)) \wedge x \in select_2(y,y) \wedge x < z)$
$y \cup y' \neq \emptyset \rightarrow \exists x\ .\ (x \in select_2(y,y))$
/* $select_3$ behaves as $select_2$ but specialized local search variants may refine them differently. */
$select_3(y) \subseteq y$
$\neg\exists x,z\ .\ (z \in (y \setminus select_3(y)) \wedge x \in select_3(y) \wedge x < z)$
$y \neq \emptyset \rightarrow \exists x\ .\ (x \in select_3(y))$
endoperational spec

Fig. 11 The specifications of *local search*.

adapters as means to add the problem-specific refinement to *hill-climbing*, allowing to keep both aspects separate. Therefore, the complete specifications, proofs, and implementations of *hill-climbing* can be reused. Only the problem-specific aspects have to be specified, proven and implemented by an adapter. Figure 13 provides the definition of such an adapter. Its main proof obligation is that the way the preference is defined fulfils the requirement on such a relation.

By keeping the problem-specific refinement separate from the generic core of the method it is easy to overcome what was viewed as the usability/reusability trade-off of PSMs [KBD+91]. The original version of hill-climbing can reused for different problems requiring different kinds of refinement. The combination of *hill-climbing* and the *set-minimizer* adapter can be used for problems that can be expressed in terms of minimizing sets. This combined version is less reusable however much more usable for cases it can be applied to. For achieving a problem-specific variant of a method it is not necessary to change the method itself. Instead, a problem-specific adapter is added. These adapters can also be stapled to increase the problem specificity of methods. We will show this in the following

PSM refinement adapter *local search -> hill-climbing*
/* $select_1$ must select one element. */
$|select_1(x)| = 1$
/* $select_2$ selects the current object if no better successors exist or a better successor if such a successor exists. */
$\neg\exists z\ .\ (z \in y' \wedge y < z) \rightarrow select_2(\{y\},y') = \{y\}$
$\exists z\ .\ (z \in y' \wedge y < z) \rightarrow select_2(\{y\},y') \in y')$
$|select_2(\{y\},y')| = 1$
endPSM refinement adapter

Fig. 12 The adapter *local-search -> hill-climbing*.

PSM refinement adapter *hill-climbing -> set-mimimizer*
/* The input set must be correct. */
correct(*input*)
/* $select_1$ must select the input set. */
$select_1(x) = \{x\}$
/* Successors are subsets that contain one element less.*/
$successor(x,y) \leftrightarrow \exists z\ .\ (z \in x \wedge y = x \setminus \{z\})$
/* We prefer smaller sets if they are still correct. */
$x < y \leftrightarrow correct(y) \wedge y \subset x$
endPSM refinement adapter

Fig. 13 The adapter *hill-climbing -> set-minimizer.*

subsection where we adapt *set-minimizer* to abductive diagnosis.

4.5 Abductive Diagnosis

In section 3, we introduced the problem *abductive diagnosis* that receives a set of observations as input and delivers a complete and parsimonious explanation (see Figure 6). We already described the problem-specific refinement of *set-minimizer* in section 3. The adapter that actually realizes this refinement is given in Figure 14. The set of all hypotheses is the set that must be minimized and correctness is defined in terms of completeness. Assumptions ensure that the output of set-minimizer is a complete and parsimonious explanation. First, we have to require that the input of the method is a complete explanation. Second, as was shown the *monotonicity assumption* (cf. Figure 14) is sufficient to prove that global parsimonity follows from its local parsimonity.

4.6 Resume

We showed in three steps the derivation of a refined PSM for abductive problems from a generic local search frame via adapters:

- *hill-climbing* := $\text{Adapter}_{\textit{local-search -> hill-climbing}}$(*local-search*)
- *set-minimizer* := $\text{Adapter}_{\textit{hill-climbing -> set-mimimizer}}$(*hill-climbing*)
- *abductive-method* := $\text{Adapter}_{\textit{set-mimimizer -> abductive-method}}$(*set-minimizer*)

The first adapter refines mainly the definition of state transitions of the method. The second adapter refines the notion of states to sets and state transition via defining a successor relationship between sets. The third adapter adds some simple terminological mappings that express the method in terms of abduction and adds assumptions that guarantee that the methods achieves the goal as it is introduced by the problem definition.

In the same way we refined PSMs we can also refine problem definitions and assumptions necessary to link PSMs and problem definitions. In section 3 we showed that the monotonicity assumption is a problem-specific refinement of an assumption that is necessary and sufficient to prove that *hill-climbing* finds a global optimum. Therefore, it is not only possible to refine PSMs but also problem definitions and assumptions. Figure 15 summarizes our problem definitions, assumptions and methods. We could have refined *hill-climbing* to a method for global optima, refined this method to a method that finds global-minimal but correct sets and this method refined to a method for abduction. For both paths, the dimension of this refinement is the *ontological commitments* made by problem definition, assumptions and PSMs. Notice that these specializations are kept separate via adapters. Therefore, it is

PSM refinement adapter *set-mimimizer -> abduction-method*
$correct(x) = complete(x)$;
$input = \{h \mid h \text{ is } hypothesis\}$;
$H_1 \subseteq H_2 \rightarrow expl(H_1) \subseteq expl(H_2)$
endPSM refinement adapter

Fig. 14 The adapter *set-minimizer -> abduction-method.*

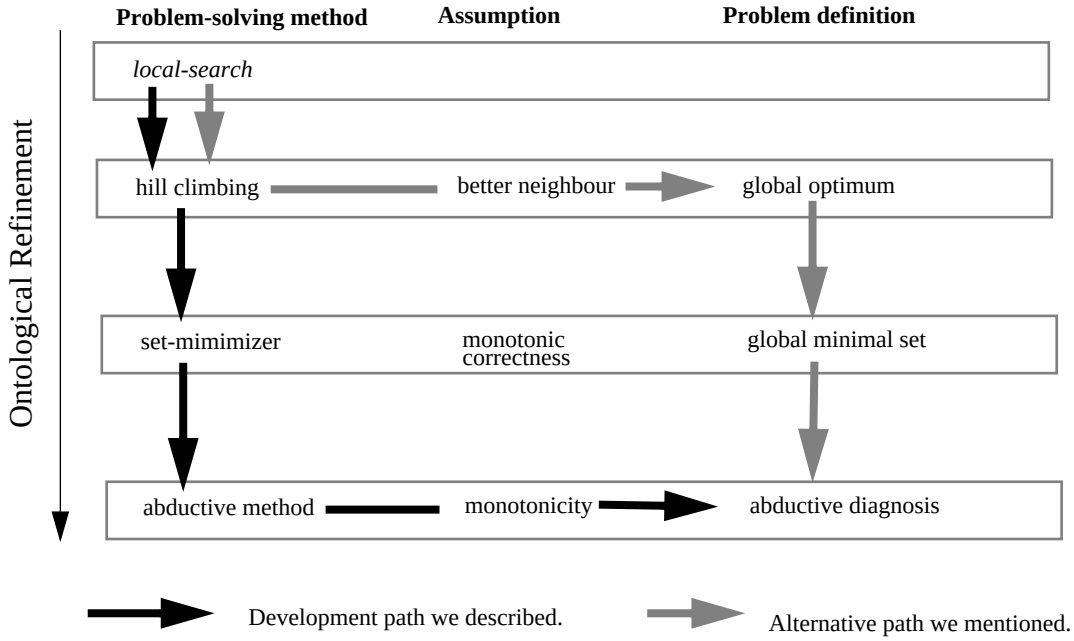

Fig. 15 Refining PSMs, assumptions and problem definitions.

always possible to reuse very specific or very generic entities of our library. We get an unified library of problem definitions, assumptions, and PSMs where the refinements have a *virtual* existence via adapters.

4.7 Refinement of Control

We view PSMs as task or problem-specific refinements of generic search strategies. However this refinement concerns the characterizations of states, elementary state transitions and assumptions over specific properties of concepts and relations used to characterize states and transitions. Take the *boardgame method* [EST+95] as an example. It refines the generic search method *chronological backtracking* for one-player board games. In [FEM+96] a formal specification of both methods is analysed for their differences. It is interesting to note that the *boardgame method* does not change the overall control, i.e. the algorithmic structure of *chronological backtracking*. It refines the notion of states in terms of board positions (chronological backtracking does not make commitments to the internal structure of the states it searches through) and uses this refined notion to define elementary state transitions in terms of *moves* that changes board positions. As a consequence, it can be easier (i.e., more efficiently) applied to tasks it is well-suited for like the Sisyphus-I assignment problem [Lin94].

Our approach should support precisely these kinds of refinement processes. Developing new generic search procedures or gaining efficiency by algorithmic optimization techniques applied to its overall control structure is beyond its scope. We restrict our attention to assumptions that formulate requirements on domain knowledge or that restrict the problem size and on ontological commitments that improve the efficiency in applying the method to a given application domain and problem by providing refined data structures. Both is usually beyond the scope of approaches that look for generic optimization because these assumptions and commitments cannot be made in the generic case. Such assumptions reflect specific domain-type or problem-type specific circumstances.

Our approach is complementary to approaches that investigate generic control regimes (see [Bun90] for a survey on search techniques). Such work can be used by our approach to introduce different starting points for our refinement process like the local search schema we used in this paper. Our approach is complementary algorithmic optimization methods like the KIDS/SPECWARE approach that refines the control, i.e. the algorithmic structure, to gain efficiency [Smi96], [JSL96]. They provide valuable support for refining a specification of a PSM into an efficient implementation.

5 Conclusions, Related and Future Work

PSMs are connected with a context like every other knowledge model. This situatedness may appear in three types of problems:

- The context a PSM is developed for may cause an error because the developers were not aware of an implicit assumption that does not hold in this context.
- The initial context of a PSM may change over time and this causes the typical maintainage problems.
- A PSM shall be shared and reused in a new context and it is hard to decide whether it fits into the new context and whether and how it has to be adapted.

We provided two contribution to tackle with the situatedness of PSMs and the problems this may cause. We provide methodological support in context *explication* and in context *adaptation*.[8]

In [FS96] we proposed the idea of characterising and developing PSMs by their underlying assumptions. However, the problem arose how to find such assumptions. Now we present the idea of the failed proof and its implementation by an interactive theorem prover. We developed and adapted PSMs by introducing assumptions that close the gap between the problem definition and the competence of a PSM. KIV has shown to be an excellent tool for our purpose. Its concepts of proof modularity and proof reuse made the development and adaptation process of PSM, that is highly iterative and reversive, tractable. The interactive theorem prover could be used to identify assumptions as open goals in partial proofs. However, more work has to be done to integrate our conceptual models used for the specifications directly into the generic module concept of KIV and to provide proof tactics and proof engineering facilities that make use of this conceptual model.

In the second part of the paper, we have shown how to use adapters for developing PSMs. The development process of PSMs is viewed as a refinement process that:

- introduces ontological commitments used to characterize initial, intermediate and terminal states of the method;
- uses ontological commitments to specialize the state transitions of a method; and
- introduces assumption to bridge the gap between competence of a method and a problem definition.

All these refinement were achieved by adding adapters to existing elements. [BBH96], [vHA96] have proposed that PSMs should be described not only in a domain-independent, but also task-independent, so that they can become more broadly reusable. However, there is a known trade-off between usability and reusability [KBD+91]. With our approach this dilemma disappears because PSMs can either reused in their generic or more problem-specific variant as the latter does not modify the former but adds only an external description to it. Therefore we also overcome the problems of [Ben95], [THW+93] that express a PSM immediately in problem-specific terms (like symptom detection, hypothesis generation, hypothesis discrimination, etc.) whereas we describe general algorithmic schemas that become instantiated to a specific class of problem via adapters. Therefore we can discuss these algorithmic schemas of PSMs independent from specific problems reflecting appropriate the fact that the same PSM (or better the same algorithmic schema) can be applied to different problem classes.

Most existing approaches for developing PSMs either stop at the level of the competence of the methods [Abe93], [AWS93], [WAS95], [Tei97] or view PSM development as a process of hierarchically refining inference actions [THW+93], [Ben95], and [BA]. The former deal only with a very limited aspects of the methods as a method is essentially a description of *how* to achieve some goals. The latter assume that adapting a PSM to a given problem is an activity of decomposing a problem in subproblems and defining control over the solution of the subproblems (and recursively refining the subproblems). However it has often been reported that different control regimes can be applied to solve the same problem type and the same control regime can be applied to very different problems [Bre97]. Therefore, we think that adapting a control regime is neither the only nor the central

8. Precisely spoken we do not adapt the context but the PSM.

point in adapting a PSM to a task.

Acknowledgements. We would like to thank Richard Benjamins, Joost Breuker, Rix Groenboom, Tim Menzies, Enrico Motta, Annette ten Teije, Frank van Harmelen, and Bob Wielinga for inspiring discussions on issues related to the paper.

References

[Abe93] M. Aben: Formally Specifying Re-Usable Knowledge Model Components, *Knowledge Acquisition*, 5, 1993.

[AFH94] N. M. Agnew, K. M. Ford, and P. J. Hayes: Expertise in Context: Personally Constructed, Socially Selected, and Reality-Relevant?, *International Journal of Expert Systems*, 7(1), 1994.

[AFS96] J. Angele, D. Fensel, and R. Studer: Domain and Task Modelling in MIKE. In A. Sutcliffe et al. (eds.), *Domain Knowledge for Interactive System Design*, Chapman & Hall, 1996.

[AS96] V. Akman and M. Surav: Steps Toward Formalizing Context, *AI Magazine*, 17(3), 1996.

[AWS93] J. M. Akkermans, B. Wielinga, and A. Th. Schreiber: Steps in Constructing Problem-Solving Methods. In N. Aussenac et al. (eds.): *Knowledge-Acquisition for Knowledge-Based Systems*, LNAI 723, Springer-Verlag, 1993.

[BAT+91] T. Bylander, D. Allemang, M. C. Tanner, and J. R. Josephson: The Computational Complexity of Abduction, *Artificial Intelligence*, 49, 1991.

[BBH96] P. Beys, R. Benjamins, and G. van Heijst: Remedying the Reusability-Usability Tradeoff for Problem-solving Methods. In *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´96)*, Banff, Canada, November 9-14, 1996.

[BA] R. Benjamins and M. Aben: Structure-Preserving KBS Development through Reusable Libraries: a Case Study in Diagnosis. To appear in International Journal on Human-Computer Studies.

[Ben95] R. Benjamins: Problem Solving Methods for Diagnosis And Their Role in Knowledge Acquisition, *International Journal of Expert Systems: Research and Application*, 8(2):93—120, 1995.

[BFS96] R. Benjamins, D. Fensel, and R. Straatman: Assumptions of Problem-Solving Methods and Their Role in Knowledge Engineering. In *Proceedings of the 12. European Conference on Artificial Intelligence (ECAI-96)*, Budapest, August 12-16, 1996.

[BG96] R. Benjamins and C. Pierret-Golbreich: Assumptions of Problem-Solving Method. In N. Shadbolt et al. (eds.), *Advances in Knowledge Acquisition*, LNAI 1076, Springer-Verlag, Berlin, 1996.

[Bör95] E. Börger: Why Use Evolving Algebras for Hardware and Software Engineering. In M. Bartosek et al. (eds.), *SOFSEM '95: Theory and Practice of Informatics*, LNCS 1012, Springer-Verlag, 1995.

[Bre94] J. Breuker: Components of Problem Solving. In L. Steels et al. (eds.), *A Future of Knowledge Acquisition*, Springer-Verlag, LNAI, 1994.

[Bre97] J. Breuker: Problems in Indexing Problem Solving Methods. In *Proceeedings of the Workshop on Problem-Solving Methods during the IJCAI-97*, Japan, August 24, 1997.

[Bun90] A. Bundy (ed.): *Catalogue of Artificial Intelligence Techniques*, 3rd ed., Springer-Verlag, Berlin, 1990.

[BvV94] J. Breuker and W. Van de Velde (eds.): *The CommonKADS Library for Expertise Modelling*, IOS Press, Amsterdam, The Netherlands, 1994.

[Byl91] T. Bylander: Complexity Results for Planning. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, Sydney, Australia, August 1991.

[Cha86] B. Chandrasekaran: Generic Tasks in Knowledge-based Reasoning: High-level Building Blocks for Expert System Design. *IEEE Expert*, 1(3): 23—30, 1986.

[CJS92] B. Chandrasekaran, T. R. Johnson, and J. W. Smith: Task Structure Analysis for Knowledge Modeling, *Communications of the ACM*, 35(9): 124—137, 1992.

[Cla93] W. J. Clancey: The Knowledge Level Reinterpreted: Modeling Socio-Technical Systems, *The International Journal of Intelligent Systems*, (8)2, 1993.

[CS95] K. Mani Chandy and Beverly A. Sanders: Predicate Transformers for Reasoning about Concurrent Programs, Science of Computer Programming, 24, 1995.

[Dij75] E. W. Dijkstra: Guarded Commands, Nondeterminacy, and Formal Derivation of Programs, *Communication of the ACM*, 18:453-457, 1975.

[EST+95] H. Eriksson, Y. Shahar, S. W. Tu, A. R. Puerta, and M. A. Musen: Task Modeling with Reusable Problem-Solving Methods, *Artificial Intelligence*, 79(2):293—326, 1995.

[FB96] D. Fensel and R. Benjamins: Assumptions in Model-based Diagnosis. In *Proceedings of the 10h Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´96)*, Banff, Canada, November November 9-14, 1996.

[Fen95a] D. Fensel: Assumptions and Limitations of a Problem-Solving Method: A Case Study. In *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´95)*, Banff, Canada, January 26 - February 3, 1995.

[FEM+96]D. Fensel, H. Eriksson, M. A. Musen, and R. Studer: Developing Problem-Solving by Introducing Ontological Commitments, *International Journal of Expert Systems: Research & Applications*, 9(4), 1996.

[FG96] D. Fensel and R. Groenboom: MLPM: Defing a Semantics and Axiomatization for Specifying the Reasoning Process of Knowledge-based Systems. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, Budapest, August 12-16, 1996.

[FG97] D. Fensel and R. Groenboom: Specifying Knowledge-Based Systems with Reusable Components. In *Proceedings of the 9th International Conference on Software Engineering & Knowledge Engineering (SEKE-97)*, Madrid, Spain, June 18-20, 1997.

[FMD+] D. Fensel, E. Motta, S. Decker, and Z. Zdrahal: The Use of Ontologies For Specifying Tasks and Problem-Solving Methods: A Case Study. Submitted to EKAW-97.

[FS96] D. Fensel und R. Straatman: The Essence of Problem-Solving Methods: Making Assumptions for Efficiency Reasons. In N. Shadbolt et al. (eds.), *Advances in Knowledge Acquisiiton, Lecture Notes in Artificial Intelligence (LNAI)*, no 1076, Springer-Verlag, Berlin, 1996.

[FS97a] D. Fensel and A. Schönegge: Assumption Hunting as Developing Method for Problem-Solving Methods, In *Proceedings of the Workshop on Problem-Solving Methods for Knowledge-Based Systems during the 15th International Joint Conference on AI (IJCAI-97)*, Nagoya, Japan, August 23-30, 1997.

[FS97b] D. Fensel and A. Schönegge: Specifying and Verifying Knowledge-Based Systems with KIV. In *Proceedings of the European Symposium on the Validation and Verification of Knowledge Based Systems EUROVAV-97*, Leuven Belgium, June 26-28, 1997.

[FSG+96] D. Fensel, A. Schönegge, R. Groenboom, and B. Wielinga: Specification and Verification of Knowledge-Based Systems. In *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´96)*, Banff, Canada, November 9-14, 1996.

[GL90] R. V. Guha and D. B. Lenat: Cyc: A Midterm Report, *AI Magazine*, 11:32-59, 1990.

[Guh93] R. V. Guha: *Context Dependence of Representations in Cyc*, MCC Technical Report, CYC 066-93, 1993.

[HS96] K. O'Hara and N. Shadbolt: The Thin End of the Wedge: Efficiency and the Generalized Directive Model Methodology. In N. Shadbolt (eds.), *Advances in Knowledge Acquisition*, LNAI 1076, Springer-Verlag, Berlin, 1996.

[JSL96] R. Juellig, Y. Srinivas, and J. Liu: SPECWARE: An Advanced Environment for the Formal Development of Complex Software Systems. In M. Wirsing et al. (eds.), *Algebraic Methodology and Software Technology*, LNCS 1101, Springer-Verlag, Berlin, 1996.

[KBD+91] G. Klinker, C. Bhola, G. Dallemagne, D. Marques, and J. McDermott: Usable and Reusable Programmin Constructs, *Knowledge Acquisition*, 3, pp. 117-136, 1991.

[KW87] J. de Kleer and B. C. Williams: Diagnosing Multiple Faults, *Artificial Intelligence*, 32:97-130, 1987.

[KMR92] J. de Kleer, K. Mackworth, and R. Reiter: Characterizing Diagnoses and Systems, *Artificial Intelligence*, 56, 1992.

[Lin94] M. Linster (ed.) (1994). Sisyphus ´91/92: Models of Problem Solving, *International Journal of Human Computer Studies (IJHCS)*, 40(3).

[Mar88] S. Marcus (ed.). *Automating Knowledge Acquisition for Experts Systems*, Kluwer Academic Publisher, Boston, 1988.

[McC93] J. McCarthy: Notes on Formalizing Context. In *Proceedings of the 13th International Conference on Artificial Intelligence (IJCAI-93)*, Chamberry, France, 1993.

[MC] T. Menzies and W. J. Clancey (eds.): The Challenge of Situated Cognition for Symbolic Knowledge

Based Systems, *Special Issue of the International Journal of Human-Computer Studies (IJHCS)*, to appear.

[Men97] T. Menzies: *35 Kinds of Knowledge Maintenance*, Technical Report TR97-03, The University of New South Wales, 1997.

[MM89] Marcus, S. and McDermott, J. SALT: A Knowledge Acquisition Language for Propose and Revise Systems, *Artificial Intelligence*, 39(1), pp. 1-37. 1989.

[MS96] Irwin Meisels and Mark Saaltink: The Z/EVES Reference Manual, ORA Canada, 1996. Available via http://www.ora.on.ca/z-eves/.

[MZ96] E. Motta and Z. Zdrahal: Parametric Design Problem Solving. In *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´96)*, Banff, Canada, November 9-14, 1996.

[Neb96] B. Nebel: Artificial Intelligence: A Computational Perspective. In G. Brewka (ed.), *Principles of Knowledge Representation*, CSLI publications, Studies in Logic, Language and Information, Stanford, 1996.

[Pop59] K. R. Popper: *The Logic of Scientific Discovery*, London, 1992 (reprint of 1959).

[Pup93] F. Puppe: *Systematic Introduction to Expert Systems: Knowledge Representation and Problem-Solving Methods*, Springer-Verlag, Berlin, 1993.

[Rei95] W. Reif: The KIV Approach to Software Engineering. In M. Broy and S. Jähnichen (eds.): *Methods, Languages, and Tools for the Construction of Correct Software*, LNCS 1009, Springer-Verlag, Berlin, 1995.

[Sha89] A. Shaw: Reasoning About Time in Higher Level Language Software, *IEEE Transactions on Software Engineering*, 15(7):875—889, 1989.

[SB95] R. Straatman and P. Beys: A Performance Model for Knowledge-based Systems. In M. Ayel and M. C. Rousset (eds.): *EUROVAV-95 European Symposium on the Validation and Verification of Knowledge Based Systems*, pages 253—263. ADEIRAS, Universite de Sovoie, Chambery, June 26-28, 1995.

[SB96] A. TH. Schreiber and W. P. Birmingham (eds.): Special issue on Sisyphus-VT, *International Journal on Human-Computer Studies (IJHCS)*, 44, 1996.

[Smi96] D. R. Smith: Towards a Classification Approach to Design. In M. Wirsing et al. (eds.), *Algebraic Methodology and Software Technology*, LNCS 1101, Springer-Verlag, Berlin, 1996.

[SL90] D. R. Smith and M. R. Lowry: Algorithm Theories and Design Tactics, *Science of Computer Programming*, 14:305—321, 1990.

[Ste90] L. Steels: Components of Expertise, *AI Magazine*, 11(2), 1990.

[Ste95] M. Stefik: *Introduction to Knowledge Systems*, Morgan Kaufman Publ., San Fransisco, 1995.

[SWA+94] A. TH. Schreiber, B. Wielinga, J. M. Akkermans, W. Van de Velde, and R. de Hoog: CommonKADS. A Comprehensive Methodology for KBS Development, *IEEE Expert*, 9(6):28—37, 1994.

[Tei97] A. ten Teije: *Automated Configuration of Problem Solving Methods in Diagnosis*, PhD thesis, University of Amsterdam, Amsterdam, NL, 1997.

[THW+93] P. Terpstra, G. van Heijst, B. Wielinga, and N. Shadbolt: Knowledge Acquisition Support Through Generalised Directive Models. In M. David et al. (eds.): *Second Generation Expert Systems*, Springer-Verlag, 1993.

[vHA96] G. van Heijst and A. Anjewerden: Four Propositions concerning the specification of Problem-Solving Methods. In *Supplementary Proceedings of the 9th European Knowledge Acquisition Workshop EKAW-96*, Nottingham, England, May 14-17, 1996.

[vdV88] W. van de Velde: Inference Structure as a Basis for Problem Solving. In *Proceedings of the 8th European Conference on Artificial Intelligence (ECAI-88)*, Munich, August 1-5, 1988.

[WAS95] B. Wielinga, J. M: Akkermans, and A. TH. Schreiber: A Formal Analysis of Parametric Design Problem Solving. In *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´95)*, Banff, Canada, January 26 - February 3, 1995.

[Wor96] John B. Wordsworth: *Software engineering with B*, Addison-Wesley, 1996.

[Zil96] S. Zilberstein: Using Anytime Algorithms in Intelligent Systems, *AI Magazine*, 17(3), 1996.

[ZM96] Z. Zdrahal and E. Motta: Improving Competence by Integrating Case-Based Reasoning and Heuristic Search. In *Proceedings of the 10h Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´96)*, Banff, Canada, November 9-14, 1996.