Thesis, University of Amsterdam, Amsterdam, The Netherlands, 1997.

[49] 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.

[50] J. Top and H. Akkermans: Tasks and Ontologies in Engineering Modeling, *International Journal of Human-Computer Studies*, 41:585—617, 1994.

[51] 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.

[52] F. van Harmelen and M. Aben: Structure-preserving Specification Languages for Knowledge-based Systems, *Journal of Human Computer Studies*, 44:187—212, 1996.

[53] G. van Heijst, A. T. Schreiber, and B. J. Wielinga: Using Explicit Ontologies in Knowledge-Based System Development, *International Journal of Human-Computer Interaction (IJHCI)*, 46(6), 1997.

[54] M. Wirsing: Algebraic Specification. In J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, Elsevier Science Publ., 1990.

[30] D. Harel: Dynamic Logic. In D. Gabby et al. (eds.), *Handook of Philosophical Logic*, vol. II, *Extensions of Classical Logic*, Publishing Company, Dordrecht (NL), 1984.

[31] S.A. Kripke: A Completeness Theorem in Modal Logic, *Journal of Symbolic Logic*, 24:1—14, 1959.

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

[33] E. Motta and Z. Zdrahal: Parametric Design Problem Solving. In *Proceedings of the 10h Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´96)*, Banff, Canada, November 9-15, 1996.

[34] A. Newell: The Knowledge Level, *Artificial Intelligence*, 18:87—127, 1982.

[35] J. Penix and P. Alexander: Toward Automated Component Adaption. In *Proceedings of the 9th International Conference on Software Engineering & Knowledge Engineering (SEKE-97)*, Madrid, Spain, June 18-20, 1997.

[36] J. Penix, P. Alexander, and K. Havelund: Declarative Specifications of Software Architectures. In *Proceedings of the 12th IEEE International Conference on Automated Software Engineering (ASEC-97)*, Incline Village, Nevada, November 1997.

[37] C. Pierret-Golbreich and X. Talon: An Algebraic Specification of the Dynamic Behaviour of Knowledge-Based Systems, *The Knowledge Engineering Review*, 11(2), 1996.

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

[39] W. Reif: Correctness of Generic Modules. In Nerode & Taitslin (eds.), *Symposium on Logical Foundations of Computer Science*, LNCS 620, Springer-Verlag, 1992.

[40] 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, 1995.

[41] G.R. Renardel de Lavalette, R. Groenboom, E.P. Rotterdam, F. van Harmelen, and A. ten Teije: Formalisation of anaesthesiology for decision support, to appear in *Artificial Intelligence in Medicine*.

[42] 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.

[43] M. Shaw and D. Garlan: *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[44] J. W. Spee and L. in ‘t Veld: The Semantics of $K_{BS}SF$: A Language For KBS Design, *Knowledge Acquisition*, vol 6, 1994.

[45] J.M. Spivey: *The Z Notation. A Reference Manual*, 2nd ed., Prentice Hall, New York, 1992.

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

[47] R. Studer, V. R. Benjamins, D. Fensel: Knowledge Engineering: Methods and Principles, to appear in *Data and Knowledge Engineering*, 1998.

[48] A. ten Teije,*Automated Configuration of Problem Solving Methods in Diagnosis*, PhD

[15] D. Fensel: The Tower-of-Adapter Method for Developing and Reusing Problem-Solving Methods. In E. Plaza et al. (eds.), *Knowledge Acquisition, Modeling and Management*, Lecture Notes in Artificial Intelligence (LNAI), 1319, Springer-Verlag, 1997.

[16] D. Fensel and R. Groenboom: MLPM: Defining 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.

[17] D. Fensel, R. Groenboom, and G. R. Renardel de Lavalette: MCL: Specifying the Reasoning of Knowledge-based Systems, to appear in *Data and Knowledge Engineering (DKE)*.

[18] D. Fensel and E. Motta: Dimensions for Method Refinement, submitted.

[19] D. Fensel and A. Schönegge: Assumption Hunting as Development Method for Knowledge-Based Systems. 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.

[20] D. Fensel and A. Schönegge: Using KIV to Specify and Verify Architectures of Knowledge-Based Systems. In *Proceedings of the 12th IEEE International Conference on Automated Software Engineering (ASEC-97)*, Incline Village, Nevada, November 1997.

[21] 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 9th - November 14th, 1996.

[22] D. Fensel and R. Straatman: The Essence of Problem-Solving Methods: Making Assumptions to Gain Efficiency, to appear in *International Journal on Human-Computer Studies*.

[23] D. Fensel and F. van Harmelen: A Comparison of Languages which Operationalize and Formalize KADS Models of Expertise, *The Knowledge Engineering Review*, 9(2), 1994.

[24] E. Gamma, R. Helm, R. Johnson, and J. Vlissides: *Design Patterns*, Addison-Wesley Pub., 1995.

[25] D. Garlan and D. Perry (eds.), Special Issue on Software Architecture, *IEEE Transactions onSoftware Engineering*, 21(4), 1995.

[26] F de Geus and E. Rotterdam: *Decision Support in Anaesthesia*, Decision, PhD thesis, University of Groningen, 1992.

[27] R. Groenboom: *Formalizing Knowledge Domains - Static and Dynamic Aspects*, PhD thesis, University of Groningen, Shaker Publ., 1997.

[28] R. Groenboom and G.R. Renardel de Lavalette: Reasoning about Dynamic Features in Specification Languages. In D.J. Andrews et al. (eds.), *Proceedings of Workshop in Semantics of Specification Languages,* October 1993, Utrecht, Springer Verlag, Berlin, 1994.

[29] T. R. Gruber: A Translation Approach to Portable Ontology Specifications, *Knowledge Acquisition*, 5:199—220, 1993.

this conceptual model.

Acknowledgement. We thank Richard Benjamins, Stefan Decker, Arno Schönegge, Remco Straatman, Rudi Studer, Annette ten Teije, Frank van Harmelen, Maarten van Someren, Bob Wielinga, and Mark Willems for helpful comments on drafts of the paper.

References

[1] 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*, Lecture Notes in AI, no 723, Springer-Verlag, 1993.

[2] J. Angele, D. Fensel, and R. Studer: Developing Knowledge-Based Systems with MIKE, to appear in *Journal of Automated Software Engineering*, 1988.

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

[4] 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.

[5] 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.

[6] M. Bidoit, H.-J. Kreowski, P. Lescane, F. Orejas and D. Sannella (eds.): *Algebraic System Specification and Development*, Lecture Notes in Computer Science (LNCS), no 501, Springer-Verlag, 1991.

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

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

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

[10] R. Davis: Diagnostic Reasoning Based on Structure and Behavior, *Artificial Intelligence*, 24: 347—410, 1984.

[11] 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.

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

[13] 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, February 26 - February 3, 1995.

[14] D. Fensel: Formal Specification Languages in Knowledge and Software Engineering, *The Knowledge Engineering Review*, 10(4), 1995.

necessary to solve the problem. A local search method need a local structure that is not necessary to define the problem but to define the problem-solving process. This type of knowledge is not present as a parameter in her framework.

In consequence, we think [48] presents rather an interesting framework for a parameterized specification of model-based diagnosis *tasks* than problem-solving methods.

8 Conclusions and Future Work

In the paper, we introduce a formal and conceptual framework for specifying and verifying knowledge-based systems. One can specify tasks, problem-solving methods, domain models, and adapters and can verify whether the assumed relationships between them are guaranteed, i.e., which assumptions are necessary for establishing these relationships. Such an architecture improves the understandability of specification and verification. The modularization reduces the effort in specification and verification by defining smaller contexts and enabling reuse of smaller parts in new contexts. The idea of an adapter allows to combine and adapt reusable elements without being forced to modify them. The specification of the problem-solving method is decomposed in external and internal aspects. The specification of the competence of the problem-solving method provides all necessary aspects for relating it with the task that must be solved. When specifying a reusable problem-solving method it must be proven one time whether the operational specification specifies a computational process that has the specified competence. When reusing the method, it is possible to abstract from all details of the internal operationalization and refer only to the external specification of the competence. In the case of the domain model, such an encapsulation is not possible because task and problem-solving method need access to meta-knowledge and domain knowledge. In the case of the task, such an encapsulation is not necessary because it does not own an internal implementation. Its implementation is described by the problem-solving method.

In addition to modelling concepts, formal development of knowledge-based systems requires tool support for modularisation of specifications and programs and for constructing, analysing, and reusing proofs. [21] and [19] report successful case studies in applying KIV to the verification of knowledge-based systems. The KIV system (Karlsruhe Interactive Verifier) (see [40]) is an advanced tool for the construction of provably correct software. [21] and [19] show how the verification of conceptual and formal specifications of knowledge-based systems can be performed with it. KIV was originally developed for the verification of procedural programs but it serves well for verifying knowledge-based systems. Its specification language is based on abstract data types for the functional specification of components and dynamic logic for the algorithmic specification. It provides an interactive theorem prover integrated into a sophisticated tool environment supporting aspects like the automatic generation of proof obligations, generation of counter examples, proof management, proof reuse etc. Such a support is essential for making the verification of complex specifications feasible. Currently we are working on problems stemming from differences of the formalization languages of KIV and MCL, on integrating our conceptual models directly into the generic module concept of KIV, and on proof tactics that make use of

7 Related Work

Recently, the knowledge level [34] has been encountered in Software Engineering (cf. [25], [43]). Work on software architectures establishes a higher level to describe the functionality and the structure of software artefacts. The main concern of this new area is the description of generic architectures that describe the essence of large and complex software systems. Such architectures specify specific classes of application problems instead of focusing on the small and generic components from which a system is built up. Our conceptual model fits nicely in this recent trend. It describes an architecture for a specific class of systems: knowledge-based systems.

Usually, architectures are described by their components and connectors that establish the proper relationships between the former. In our case, we have three types of components (tasks, problem-solving methods, and domain models) and adapters that connect them. However, adapters do not deal with communication aspects as it is often the case for connectors. Each problem-solving method could hierarchically refine the architecture by introducing a set of new subtasks through the functional specification of its elementary inference steps.

Work on formalizing software architectures characterizes the functionality of architectures in terms of assumptions over the functionality of its components [35], [36]. This shows strong similarities to our work where we define the competence of problem-solving method in terms of assumptions over domain knowledge (which can be viewed as one or several components of a knowledge-based systems) and the functionality of elementary inference steps. However, [35], [36] abstract from the operational specification of the architecture and keep its specification and verification separate. In our framework, it is treated as an integrated piece of the specification of the entire architecture. This is the reason why we rely on a combination of abstract data types and dynamic logic for specification and verification whereas [35], [36] use only abstract data types.

An interesting architecture for the specification of problem-solving methods in the area of model-based diagnosis is presented by [48]. The specification of the competence of a problem-solving method is parameterized by a fixed set of component types. Changing a different functionality of a component by selecting a different instantiation for one of the component types modifies the competence of the entire method. Being a very interesting approach we wonder whether it is really useful for specifying problem-solving methods. In our opinion, two key features of problem-solving methods are missing:

- Problem-solving methods describe how a problem is solved and the operational strategy is not covered by the declarative specification of its competence.
- A task introduces knowledge requirements to define a problem in domain-specific terms. A problem-solving method introduces additional knowledge requirements that are

2. For example, a more serious assumption would be the single-fault assumption (cf. [10]). Formulating it as a requirement on domain knowledge enforces that each possible fault combination is represented as a single fault by the domain knowledge. Therefore, it is often used as an assumption that limits the scope of the problems that can be handled correctly by the system. Cases, where a single fault is the actual cause can be solved correctly by the system. Situation with more complex error situations must be solved without support by the system. In general, formulating a property as a requirement increases the demand on domain knowledge and formulating a property as an assumption decreases the application scope of the system (cf. [5]).

adapter $TP_{Adapter}$
include *set-minimizer, complete and parsimonious explanation*
rename *set-minimizer* **by** *abduction*
object → *hypothesis, objects* →*hypotheses, correct* → *complete*
variables *x* : *datum, H,H´*: *hypotheses*
axioms *goal*(*Output*)
requirements
$\exists x$ ($x \in$ *observables*)
complete(*Init*)
assumptions $\forall$H ,*H´*(*H* ⊆ *H'* → *expl*(*H*) ⊆ *expl*(*H'*))
endadapter

Fig. 10 The intermediate version of the $TP_{Adapter}$.

its connection with the domain knowledge.

6.2 Connecting with the Domain Model

Finally, we have to link the domain model with the other components using the $D_{Adapter}$ (see Figure 11). We have to map the different terminologies, to define the logical relationships between domain knowledge and the other parts of the specification by axioms, and to prove the validness requirements on domain knowledge by the other parts. For our example, most of these requirements follow straight-forward from the meta-knowledge of the domain model. Therefore, the monotonicity of hypotheses can be stated as a requirement because it follows from the specification of the domain knowledge. If a requirement cannot be derived from the domain knowledge it must be stated as an assumption. In our example, the requirement

$\exists x$ ($x \in$ *observables*)

cannot be derived from the domain knowledge because it is concerned with the input. However, assuming an input for deriving a diagnosis is not a critical assumption.[2] It remains to ensure that *Init* delivers a correct set of hypotheses. An easy way to achieve this is to deliver the entire set of hypotheses (given the monotony of the problem), i.e. $\forall h \in$ *Init.*

adapter $D_{Adapter}$
include *anesthesiology*, $TP_{Adapter}$
rename *anesthesiology* **by** $TP_{Adapter}$
symptom → *datum,*
variables *h* : *hypothesis, x* : *datum, H,H´*: *hypotheses*
axioms
$\forall x$,*H* ($x \in$ *expl*(*H*) ↔ $\exists$ *h* (*h* ∈ *H* ∧ *causes*(*h*,*x*)))
proof obligation
$\exists x$ ($x \in$ *observables*)
complete(*Init*)
$\forall$H ,*H´*(*H* ⊆ *H'* → *expl*(*H*) ⊆ *expl*(*H'*))
endadapter

Fig. 11 The initial $D_{Adapter}$.

adapter $TP_{Adapter}$
include *set-minimizer, complete and parsimonious explanation*
rename *set-minimizer* **by** *abduction*
object → *hypothesis, objects* →*hypotheses, correct* → *complete*
variables *x* : *datum*
proof obligation *goal*(*Output*)
requirements
∃*x* (*x* ∈ *observables*)
complete(*Init*)
endadapter

Fig. 9 The initial version of the $TP_{Adapter}$.

purpose. First we demonstrate how to link task and PSM by the $TP_{Adapter}$. Then we discuss their relations with the domain model defined by the $D_{Adapter}$.

6.1 Connecting Task and PSM

Combining task and PSM requires three activities: establishing of syntactical links between different terminologies by mappings (see [39] for more details), establishing of semantic links between different predicates, and the introduction of new assumptions and requirements to establish that the goals of the task is implied by the output of the method.

In our case study, we have to link the sorts *object* and *objects* and the predicate symbol *correct* of the PSM by renaming (see Figure 9). The appropriate interpretation of predicates have to be ensured by axioms if they cannot linked directly. The necessity that the output of the method implies the goal of the task is stated as proof obligation (see Figure 9)

The $TP_{Adapter}$ contains the collection of the requirements of task and PSM. This includes (cf. Figure 9): any application problem provides at least one observation and the set of hypotheses delivered by *Init* must be a complete explanation of all observations (see Figure 9). These requirements must be fulfilled by the domain knowledge and the input to ensure that the task is well-defined and the inference steps of the PSM work proper.

Finally, we have to introduce new assumptions and requirements to ensure that the competence of the PSM implies the goal of the task (i.e., to fulfil the proof obligation of the adapter). We already know that *Output* contains a locally-minimal set. Each subset of it that contains one less element is not a complete explanation. Still this is not strong enough to guaranty parsimoniousness of the explanation in the general case. There may exist smaller subsets that are complete explanations. In [19], we have proven that the global-minimality of the task definition is implied by the local-minimality if we introduce the *monotonic-problem assumption* (see [3]):

$$H \subseteq H' \rightarrow expl(H) \subseteq expl(H')$$

For details on how to find such assumptions with an interactive theorem prover see [19], [20].

Figure 10 provides the intermediate adapter that contains the fulfilled proof obligation and the new introduced assumption. Whether the monotonicity property must be stated as an assumption or whether it can be formulated as a requirement on domain knowledge in the final version of the adapter can be decided when specifying the second aspect of the adapter,

domain model *anesthesiology*
signature
sorts *hypothesis, hypotheses* **set of** *hypothesis, symptom*
functions
HighHeartRate, HighPartm, ToolowCOP, WakingUp : *symptom*
Centralization, Pain, Edema, LowAnesthesia : *hypothesis*
predicates
causes: *hypothesis* x *symptom*
variables
h : *hypothesis*
s : *symptom*
H, H': *hypotheses*
meta-knowledge
there is a cause for each symptom
$\forall s \exists h\ causes(h, s)$
the fault knowledge is monotonic
$H \subseteq H' \rightarrow \{s \mid h \in H \wedge causes(h,s)\} \subseteq \{s \mid h \in H' \wedge causes(h,s)\}$
domain knowledge
causes(*LowAnesthesia,WakingUp*)
causes(*Centralization,HighPartm*)
causes(*Pain,HighPartm*)
causes(*Pain,WakingUp*)
causes(*Pain,HighHeartRate*)
causes(*Edema,HighHeartRate*)
causes(*Edema,ToolowCOP*)
assumption
complete fault knowledge
$\forall h,s((h \neq Centralization \vee h \neq Pain \vee h \neq Edema \vee h \neq LowAnesthesia) \vee$
$(s \neq HighHeartRate \vee s \neq HighPartm \vee s \neq ToolowCOP \vee s \neq WakingUp)$
$\rightarrow \neg causes(h,s))$.
enddm

Fig. 8 The domain model.

leads to a larger set of symptoms that can be explained. The *complete-fault-knowledge assumptions* guarantees that there are no other unknown faults like hidden diseases. Only under this assumption we can deductively infer causes from observed symptoms. However, it is a critical assumption when relating the output of our system with the actual problem and domain (cf. [12]).[1]

6 An Adapter

An adapter has to link the different signatures of task, PSM, and domain, and has to add further axioms to guaranty their proper relationships. We use abstract data types for this

1. Notice that we do not assume complete knowledge of symptoms.

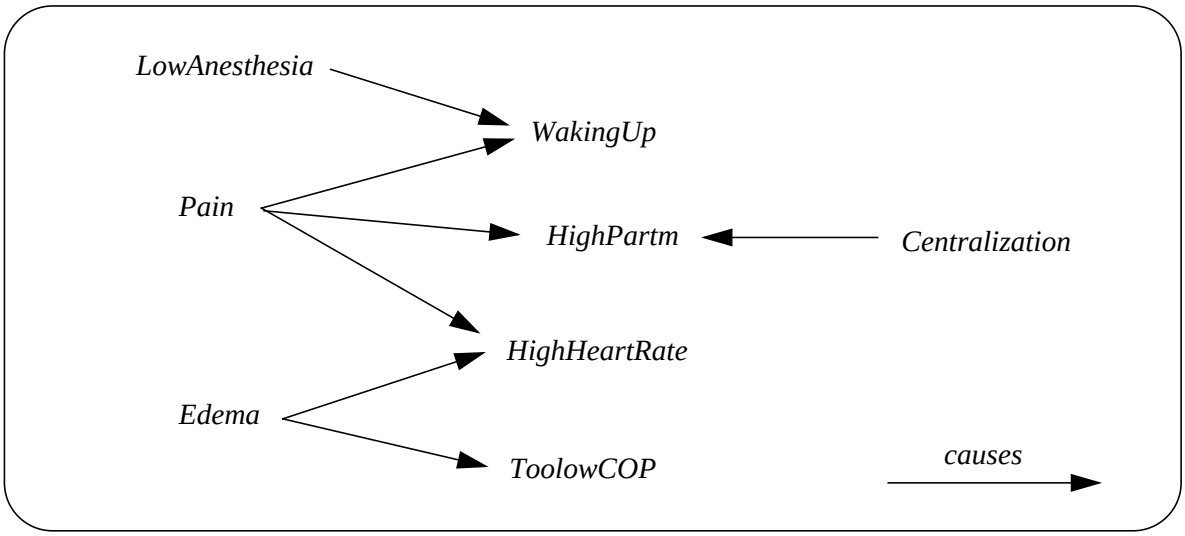

Fig. 7 The domain knowledge.

for details).

Some of the simplications we have made include:

- In this simplified domain we have a "one step" causal relation R. In practise we have the transitive and irreflexive closure R+. Note that we do not have the reflexive transitive closure R* since a symptom cannot be a hypothesis for itself. In a more complex version of this domain model, a symptom can be the hypothesis for another symptom. This kind of reasoning is left out for expository reasons (see [41] for details).
- To obtain a complete model, we restricted the number of possible hypotheses. Although this seems a major restriction, it is the same as we have to employ to the larger knowledge base. In consultation with the physician we restrict the number of diagnoses (hypotheses) we want the system to detect. Then we design a system to detect these hypothesis, leaving the final diagnosis to the physician. This is also the main reason why the system is a *support* system; the goal of the system is not to replace a physician only to support him.

A last simplification is the abstraction from time. Although the quantitative data is measures a certain time-points, we model causal-knowledge as non-temperal. The notion of time is handled elsewhere in the domain (see [27] and [41] for details).

In this domain we deal with abstract notions, derived from interpretations of measurements. The exact meaning of *HighPartm* (which stand for a High mean arterial blood-pressure) is defined elsewhere in the formal domain model (see [27]). Another technical term is *ToolowCOP*, which refers to a too low Cellular Oxygen Supply. Figure 7 sketeches the causal knowledge and Figure 8 defines the signature and axioms of our domain model. It contains hypotheses and symptoms and a causal relationship between them. The meta-knowledge ensures two properties of the domain knowledge: there is a cause for each symptom and hypotheses do not conflict. That is, different hypotheses do not lead to an inconsistent set of symptoms. In our domain this is guaranteed by the fact, that we do not have knowledge about negative evidence (i.e., a symptom may rule out an explanation). Assuming more causes only

```
Node := Init;
if Node = ∅
  then Output := ∅
  else
    repeat
      Nodes := λx.generate(x,Node);
      ∪x.((select(x,Node,z) ∧ Nodes(z))?; New node := λy.(x=y)
    until New node = Node
    Output := Node
endif
```

Fig. 6 The specification of dynamics.

A complete introduction to MCL is beyond the scope of this paper (see [17] for more details). We will only mention the features that are used in our example. Dynamic logic is often presented with the variable assignments $x:=t$ as its atomic programs. In MCL, due to the richer state representation, the atomic programs are

- $f:=\lambda x.t$ and $p:=\lambda x.A$
  changing the interpretation of a function or predicate such that $f(x)=t(x)$ and $p(x)=A(x)$ resp.;
- NEW
  creation of a new object denoted by new; and
- $\cup\, x.\alpha$
  do α for a non-determinnistically chosen x for a given program α.

Besides the atomic programs, MCL has the normal imperative statements for sequential compostition, choice and repetition. In our example, we apply $p:=\lambda x.A$ to express that *Nodes* is updated by all successor sets of the set contained by *Node* and $\cup\, x.\alpha$ to express that *New node* is updated (non-deterministically) by one correct successor set if exists or the predecessor if not.

5 The Domain Model

A domain model consists of three main parts: the domain knowledge, its meta-level characterization, and its external assumptions. In addition, a signature definition is provided that defines the common vocabulary of the other three elements.

The medical domain model we have chosen for our example is a subset of a large case-study in the formalization of domain knowledge for an anesthesiological support system. The support system should diagnose (limited) number of hypothesis, based on real-time acquired data. This data is obtained from the medial database system Carola [26] which performs on-line logging of measurements. The formal model includes knowledge how to interpret these raw measurements (quantitative data) and causal relations between qualitative data (see [41]

problem-solving method. Our method *set-minimizer* requires knowledge about correct sets and an initial set. This is modelled by the static knowledge roles. Figure 5 provides the definitions of the two inference actions and of a dynamic and of a static knowledge role. Basically, *generate* derives all subsets that have one element less and *select* selects a successor (one of these reduced sets) if a correct successor exists. Otherwise it selects the original node.

4.2.2 Control Flow

In Software Engineering, the distinction between a functional specification and the design/ implementation of a system is often discussed as a separation of *what* and *how*. During the specification phase, *what* the system should do is established in interaction with the users. *How* the system functionality is realized is defined during design and implementation (e.g., which algorithmic solution can be applied). This separation—which even in the domain of Software Engineering is often not practicable in the strict sense—does *not* work in the same way for KBSs: A high amount of the problem-solving knowledge, i.e. knowledge about *how* to meet the requirements, is not a question of efficient algorithms and data structures, but exists as heuristics as a result of the experience of an expert. For many problems which are completely specifiable it is not possible to find an efficient algorithmic solution. Often they are easy to specify but it is not necessarily possible to derive an efficient algorithm from these specifications; heuristics and domain-specific inference knowledge are needed for the efficient derivation of a solution. One must not only acquire knowledge about what a solution for a given problem is, but also knowledge about how to derive such a solution in an efficient manner. Already at the knowledge level there must be a description of the domain knowledge and the problem-solving method which is required by an agent to solve the problem effectively and efficiently. In addition, the symbol level has to provide a description of efficient algorithmic solutions and data structures for implementing an efficient computer program. As in Software Engineering, this type of knowledge can be added during the design and implementation of the system. Therefore a specification language for KBSs must *combine non-functional and functional specification techniques*: On the one hand, it must be possible to express algorithmic control over the execution of substeps. On the other hand, it must be possible to characterize the overall functionality and the functionality of the substeps (i.e., the inference actions) without making commitments to their algorithmic realization.

Therefore, the operational description of a PSM is completed by defining the control flow (see Figure 6) that defines the execution order of the inference actions. The specification in Figure 6 uses the *Modal Logic of Change (MCL)* [17] which was developed in to combine functional specification of substeps with procedural control of them. MCL is a generalized version of the Modal logic of Creation and Modification (MCML, see [28], [27]) and the Modal Logic for Predicate Modification (MLPM [16]). Each of these languages are variants of dynamic logic. Dynamic logic [30] was developed to express states, state transitions, and procedural control of these transitions in a logical framework. Dynamic logic uses the *possible-worlds* semantics of [31] for this purpose. A state is represented by a possible world through the value assignments of the program variables. MCL extends the representation of a state. A state is represented by an *algebra* following the states-as-algebras paradigm of evolving algebras (i.e., abstract state machines). A state transition is achieved by changing the truth values of a predicate or the values of a term. MCL provides the usual procedural constructs such as sequence, if-then-else, choice, and while-loop to define complex transition.

structure of this method is given in Figure 4 following the conventions of CommonKADS [42]. It specifies the main inferences of a method (i.e., its substeps), the dataflow between the inferences (i.e., the knowledge flow and the dynamic knowledge roles), and the knowledge types (i.e., the static knowledge roles) that are required by them. In the following, we will define each of these elements in more detail. In addition, we will have to define the control flow between the inference steps. The latter introduces a strong new requirement on our means for formalization requiring a logic of changes.

4.2.1 Inference Actions and Knowledge Roles

Again we use algebraic specifications to specify the functionality of inference actions and knowledge roles. Dynamic knowledge roles (dkr) are means to represent the state of the reasoning process and axioms can be used to represent state invariants. They introduce dynamic signature and correspond roughly to state schemas in Z [45]. The interpretation of constants, functions and predicates may change during the problem-solving process. Static knowledge roles (skr) are means to include domain knowledge into the reasoning process of a

inf *generate*
sorts *object, objects* **set of** *object*
variables z : *object*, x,y : *objects*
axioms
$generate\ (x,y) \leftrightarrow \exists z\ (z \in y \wedge x = y \setminus \{z\})$
endinf

inf *select*
sorts *object, objects* **set of** *object*
variables x,z : *object*, y : *objects*
axioms
$\forall x,y,z\ (\exists z'\ (z' \in z \wedge correct(z')) \rightarrow select(x,y,z) \wedge correct(x) \wedge x \in z)$
$\forall x,y,z\ (\neg\exists y'\ (y' \in y \wedge correct(y')) \rightarrow select(x,y,z) \wedge x = y)$
endinf

---

dkr *New node*
sorts *object, objects* **set of** *object*
constant *New node* : *objects*
enddkr

dkr *Node*
sorts *object, objects* **set of** *object*
constant *Node* : *objects*
enddkr

dkr *Nodes*
sorts *object, objects* **set of** *object*
predicate *Nodes* : *objects*
enddkr

skr *Correct*
sorts *object, objects* **set of** *object*
predicates *correct* : *objects*
endskr[1]

skr *Init*
sorts *object, objects* **set of** *object*
constant *Init* : *objects*
endskr

1. Sets can be either represented by constants or predicates.

Fig. 5 The specification of the inference actions and the knowledge roles.

competence *set-minimizer*
sorts *object*, *objects* **set of** *object*
predicates *correct* : *objects*
variables *x* : *object*
constants *Input*, *Output* : *objects*
axioms
$correct(Output)$
$\neg \exists x \, (\, x \in Output \wedge correct(Output \setminus \{x\}))$
endcompetence

requirements *set-minimizer*
sorts *object*, *objects* **set of** *object*
predicates *correct* : *objects*
constants *Init* : *objects*
axioms
$correct(Init)$
endrequirements

Fig. 3 The competence and requirements of the PSM.

being provided by the domain knowledge, by a human expert or by another PSM. The method only minimizes this correct set.

4.2 The Operational Specification

Our method *set-minimizer* uses depth-first search through a search tree that is derived from set inclusion. The entire method is decomposed into the following two steps: The inference action *generate* generates all successor sets that contain one element less. The inference action *select* selects one correct set from the successors and the predecessor. The inference

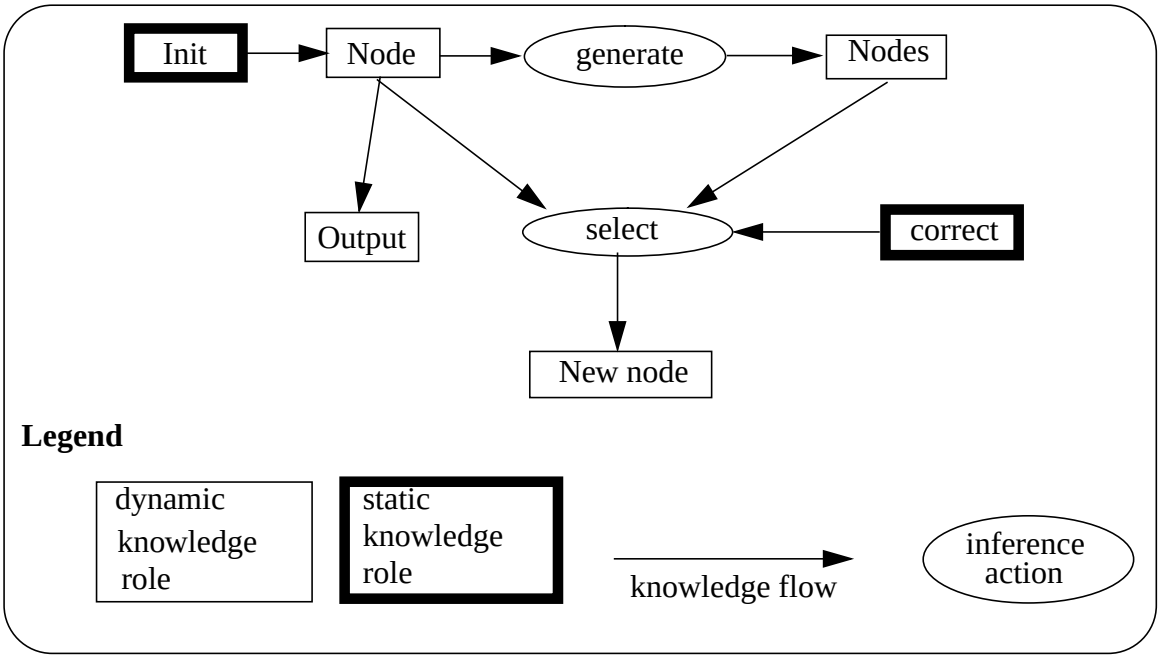

Fig. 4 Knowledge flow diagram of *set-minimizer.*

task *complete and parsimonious explanation*
sorts
datum, data : **set of** *datum,*
hypothesis, hypotheses : **set of** *hypothesis*
functions
expl: *hypotheses* → *data*
observables: *data*
predicates
goal : *hypotheses*
complete: *hypotheses*
parsimonious: *hypotheses*
variables
x : *datum*
H,H' : *hypotheses*
axioms
goal
$\forall H\ (goal(H) \rightarrow complete(H) \wedge parsimonious(H))$
$\forall H\ (complete(H) \leftrightarrow expl(H) = observables)$
$\forall H\ (parsimonious(H) \leftrightarrow \neg\exists H'\ (H' \subset H \wedge expl(H) \subseteq expl(H')))$
input requirement
$\exists x\ (x \in observables)$
endtask

Fig. 2 The task definition for *abduction*.

(cf. [15], [18] for more details). In the following, we first provide the black box specification of the method. That is, we specify the competence provided by the method and the knowledge required by the method. Then, we provide a white box specification of the operational reasoning strategy which explains how the competence can be achieved. The former is of interest during reuse of PSMs whereas the latter is required for developing PSMs.

4.1 The Black Box Description: Competence and Requirements

The competence description of the PSM as well as the task definition are declarative specifications. In consequence we apply the same formal means for their specifications: The task specifies the problem that should be solved by applying the KBS and the PSM specifies the actual functionality of the of the KBS (given that the domain knowledge fulfil the requirements of the PSM). A PSM introduces additional requirements on domain knowledge and may weaken the task definition. However, both aspects can directly be covered by algebraic data types.

The competence theory in Figure 3 defines the competence of a PSM that we call *set-minimizer*. *Set-minimizer* is able to find a correct and locally minimal set. Local minimality means, that there is no correct subset of the output that has only one element less. The method has one requirements: it must recieve a correct initial set (cf. Figure 3). The competence as well as the requirements illustrate the additional aspects that are introduced by the PSM:

- The task of finding a parsimonious set is reduced to local parsimonious sets.
- Constructing an initial correct set is outside the scope of the method. It is assumed as

- (b) In addition to the already existing requirements, an adapter may need to introduce new requirements on domain knowledge and assumptions (properties that do not follow from the domain model) to guaranty, that the competence of the PSM is strong enough to proceed the task.
- (c) We have to prove that the requirements of the adapter are implied by the meta knowledge of the domain model.

Notice that PO-i deals with the task definition internally, PO-ii deals with the PSM internally, and PO-iii deals with the domain model internally, whereas PO-iv deals with the external relationships between task, PSM, domain knowledge, and adapter. Thus a separation of concerns is achieved that contributes to the feasibility of the verification (cf. [52]). The conceptual model applied to describe KBSs is used to brake the general proof obligations into smaller pieces and makes parts of them reusable. As PSMs can be reused, the proofs of PO-ii does not have to be repeated for every application. These proofs have to be done only when a new PSM is introduced into the library. Similar proof economy can be achieved for PO-i and PO-iii by reusable task definitions and domain models. Application specific proof obligation is PO-iv.

Assumptions concerning the input cannot be verified during the development process of a KBS. However, their derivation is very important because they define pre-conditions for valid inputs that must be checked for actual inputs to guaranty the correctness of the system.

3 Formalizing Tasks

We use a simple task to illustrate our approach. The task *abductive diagnosis* receives a set of observations as input and delivers a complete and parsimonious explanation (see e.g. [3]). An explanation is a set of hypotheses. A *complete explanation* must explain all input data (i.e., *observations*) and a *parsimonious* explanation must be minimal (that is, no subset of hypotheses explains all *observations*). Figure 2 provides the task definition for our example. Any explanation that fulfils the *goal* must be *complete* and *parsimonious*. The input requirement ensures that there are *observations*.

The task does not introduce any requirements on domain knowledge by axioms but the domain model must provide sets to interpret the sorts *datum* and *hypothesis* and an explanation function *expl*. We will see how the signature mapping is achieved by the adapter.

4 The Problem-Solving Method

Finding a complete and parsimonious explanation is NP-hard in the number of hypotheses [3]. Therefore, we have to apply heuristic search strategies. In the following, we characterize a local search method which we call *set-minimizer*. The discussion whether other methods would be better suited or how we have selected this method is beyond the scope of this paper

are not captured by the model), the behavioural description of faults is complete (all fault behaviours of the components are modelled), the behavioural discrepancy that is provided as input is not the result of a measurement fault, etc. (cf. [12]).

2.1.4 The Adapter

The *adapter* maps the different terminologies of task definition, PSM, and domain model. Moreover, it gives further requirements adn asumptions that are needed to relate the competence of a PSM with the functionality given by the task definition (cf. [13], [5]). The use corresponds to the notion of *adapter pattern* in [24] where adapters are given as a design pattern to allow the reuse of object classes and the specification of re-usebale object classes. We already mentioned the fact that an adapter usually introduces new requirements or assumptions because in general, most problems tackled with KBSs are inherently complex and intractable. A PSM can only solve such tasks with reasonable computational effort by introducing assumptions that restrict the complexity of the problem or by strengthening the requirements on domain knowledge. Task, PSM, and domain model can be described independently and selected from libraries because adapter relate the three other parts of a specification together and establishes their relationship in a way that meets the specific application problem. Their consistent combination and their adaptation to the specific aspects of the given application—because they should be reusable they need to abstract from specific aspects of application problems—must be provided by the adapter.

2.2 The Main Proof Obligations

Following the conceptual model of the specification of a KBS, the overall verification of a KBS is broken down into four kinds of proof obligations (see Figure 1).

(PO-i) The consistence of the task definition ensures that a model exist. Otherwise, we would define an unsolvable problem. The requirements on domain knowledge are necessary to prove that the goal of the task can be achieved. Such a proof is usually done by constructing a model via an (inefficient) generate & test like implementation.

(PO-ii) We have to show that the operational specification of the PSM describes a PSM for that termination can be guaranteed and that the PSM has the competence as specified. This proof obligation recursively returns for each non-elementary inference action of a PSM. In addition to termination, one may also want to include some thresholds for the efficiency of the method by including it as part of the competence description (cf. [42]).

(PO-iii) We have to ensure internal consistency of the domain knowledge and domain model. The domain knowledge needs not to be overall consist but it must be possible to divide it into consistent parts. In addition, we have to prove that given its assumptions the domain knowledge actually implies its meta-level characterization.

(PO-iv) We have to establish the relationships between the different elements of the specification:

- (a) We have to prove that the requirements of the adapter imply the knowledge requirements of the PSM and the task.

requirements of a PSM.

The competence description of the PSM as well as the task definition are declarative specifications. The former specifies the actual functionality of the of the KBS (given that the domain knowledge fulfil the requirements of the PSM) and the latter specifies the problem that should be solved by applying the KBS. We make a distinction between both for two reasons:

- First, a PSM introduces requirements on domain knowledge in addition to the task definition. This knowledge is not necessary to define the problem but required to describe the solution process of the problem.
- Second, we cannot always assume that the functionality of the KBS is strong enough to completely solve the problem. Most problems tackled with KBSs are inherently complex and intractable (cf. [3], [33]). PSMs need to introduce assumptions that reduce the size of the problem they can deal with (see [22]). The later discussed adapters are the specification elements that contain the assumptions that have to be made to bridge the gap between both specifications.

A simple example may clarify these two points. The task of finding a global optimum is defined in terms of preference and an ordening relations. First, a PSM based on a local search technique requires in addition a local neighbour relation to guide the search process. This knowledge is not necessary to define the task but to define the problem-solving process and its competence. Depending on the properties of this neighbourhood relation different competencies of a method a re possible (cf. [24], [19]). Second, the task of finding an optimal solution could easily define a NP-hard problem. The PSM based on local search technique may provide solutions in polynomial time. However, it derives only a local optimum. Therefore, on must either put strong requirements on domain knowledge (each local optima must also be a global one) or one must weaken the task to local instead of global optima (cf. [5]) to establish the correspondence of PSM and task.

2.1.3 The Domain Model

The description of the *domain model* introduces the domain knowledge as it is required by the PSM and the task definition. Ontologies are proposed in knowledge engineering as a means to represent domain knowledge in a reusable manner (cf. [29], [50], [53]). 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. These external assumptions capture the implicit and explicit assumptions made while building a domain model of the real world. Technically they can be viewed as the missing pieces in the proof that the domain knowledge fulfils its meta-level characterisations. Some of these properties may be directly inferred from the domain knowledge whereas others can only be derived by introducing assumptions on the environment of the system and the actual provided input. For example, typical external assumption in model-based diagnosis are: the fault model is complete (no fault appears that

knowledge. For example, a task that defines the derivation of a diagnosis requires causal knowledge explaining observables as domain knowledge. Axioms are used to define the requirements on such knowledge. A natural candidate for the formal task definition are *algebraic specifications*. They have been developed in software engineering to define the functionality of a software artefact (cf. [6], [54]) and have already been applied by [44] and [37] for KBS. In a nutshell, algebraic specifications provide a signature (consisting of types, constants, functions and predicates) and a set of axioms that define properties of these syntactical elements.

2.1.2 The Problem-Solving Method

The concept *problem-solving method* (PSM) is present in a large part of current knowledge-engineering frameworks (e.g. GENERIC TASKS [8]; ROLE-LIMITING METHODS [32], [38]; KADS [7] and CommonKADS [42]; the METHOD-TO-TASK approach [11]; COMPONENTS OF EXPERTISE [46]; GDM [49]; MIKE [2]). Libraries of PSMs are described in [4], [7], [9], [33], and [38]. In general a PSM describes which reasoning steps and which types of knowledge are needed to perform a task. Besides some differences between the approaches, there is strong consensus that a PSM:

- decomposes the entire reasoning task into more elementary inferences;
- defines the types of knowledge that are needed by the inference steps to be done; and
- defines control and knowledge flow between the inferences.

In addition, [51] and [1] define the *competence* of a PSM independent from the specification of its operational reasoning behaviour. Proving that a PSM has some competence has the clear advantage that the selection of a method for a given problem and the verification whether a PSM fulfils its task can be done independently from details of the internal reasoning behaviour of the method.

The description of 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. As for task definitions, algebraic specifications can be used for this purpose.

An *operational description* defines the 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. We use a variant of dynamic logic (cf. [17]) to express procedural control over the execution of inferences. The definition of an inference step could recursively introduce a new (sub-)task definition. This process of stepwise refinement stops when the realization of such an inference is reearded as an implementation issue that is neglected during the specification process of the KBS.

The third element of a PSM are *requirements* on domain knowledge. Each inference step and therefore the competence description of a PSM require specific types of domain knowledge. These complex requirements on domain knowledge distinguish a PSM from usual software products. Pre-conditions on valid inputs are extended to complex requirements on available domain knowledge. Again, we will apply abstract data types for the specification of the

defines the reasoning process of a KBS; and a *domain model* that describes the domain knowledge of the KBS. Each of these three elements are described independently to enable the reuse of task descriptions in different domains (see [7]), the reuse of PSMs for different tasks and domains ([38], [7], [4]), and the reuse of domain knowledge for different tasks and PSMs (cf. [29], [50], [53]). Therefore, 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. This new introduced element is used to introduce assumptions and to map the different terminologies.

2.1.1 The Task

The description of a *task* specifies goals that should be achieved in order to solve a given problem. A second part of a task specification is the definition of requirements on domain

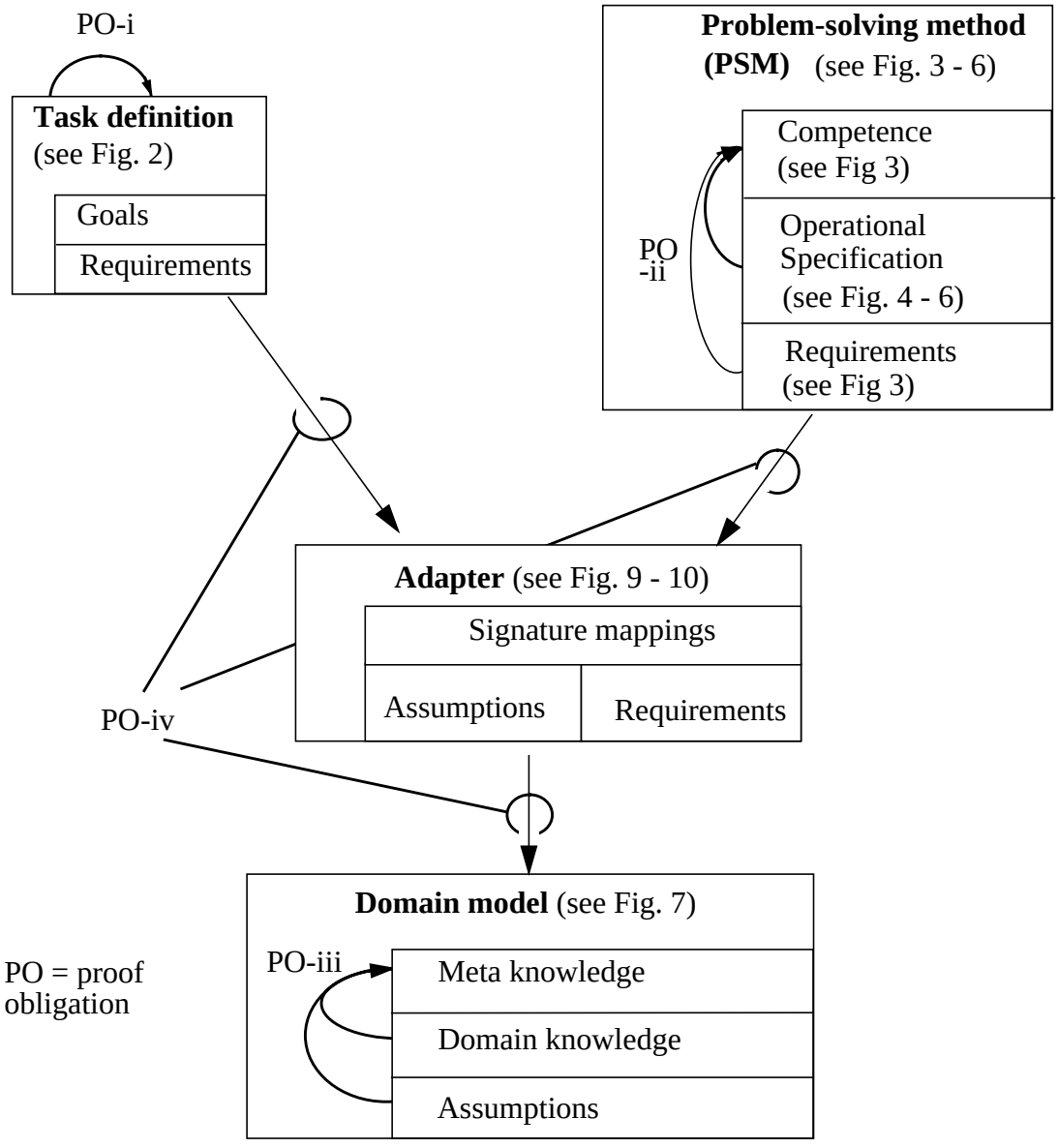

Fig. 1 The four elements of a specification of a KBS.

components it becomes an essential task to verify whether the assumptions of such a reusable building block fit to each other and the specific circumstances of the actual problem and knowledge.

In the paper, we discuss a conceptual and formal framework for the specification of KBSs. The conceptual framework is developed in accordance to the CommonKADS model of expertise (see [25], [42]) because this model has become widely used by the knowledge engineering community. The formal means applied are based on combining variants of algebraic specification techniques (see [6], [54]) and dynamic logic (see [30]). As a consequence of our modularized specification, we identify several proof obligations that arise in order to guarantee a consistent specification. The overall verification of a KBS is broken down into different types of proof obligations that ensure that the different elements of a specification together define a consistent system with appropriate functionality.

Our conceptual and formal model can be viewed as an software architecture for a specific class of systems, i.e. KBSs. A software architecture decomposes a systems into components and define their relationships (cf. [43]). This recent trend in software engineering work on establishing a more abstract level in describing software artefacts than it was common before. The main concern of this new area is the description of generic architectures that describe the essence of large and complex software systems. Such architectures specific classes of application problems instead of focusing on the small and generic components from which a system is built up. In the comparison section of this paper we will take a closer look on analogies and differences between our work and this recent line of research in software engineering.

The paper is organised as follows. In section 2, we discuss the different conceptual elements of a specification of a KBS and which kinds of proof obligation arise in their context. During section 3 until section 6, we introduce our formal means to specify the different elements. In each section, we use an example for illustrating these formalizations. Section 7 compares with related work and Section 8 summarizes the paper and defines objectives for future research.

2 A Formal Framework for the Specification of Knowledge-Based Systems

During the following, we first introduce the different elements of a specification. Then we discuss how they are related and which proof obligations arise from these relationships. In this paper, we focus on the specification of the different elements of our architecture. Actual proof activities and tool support with the interactive theorem prover environment KIV is discussed in [20].

2.1 The Main Elements of a Specification

Our framework for describing a KBS consists of four elements (see Figure 1): a *task* that defines the problem that should be solved by the KBS; a *problem-solving method* (PSM) that

An Architecture for Knowledge-Based Systems

Dieter Fensel[1] and Rix Groenboom[2]

[1] University of Karlsruhe, Institut AIFB, D-76128 Karlsruhe, Germany,
fensel@aifb.uni-karsruhe.de
[2] University of Groningen, Department of Computer Science, P.O. Box 800,
9700 AV Groningen, NL, rix@cs.rug.nl

Abstract. The paper introduces an architecture for the specification and verification of knowledge-based systems combining conceptual and formal techniques. We identify four elements of the specification of a knowledge-based system: a task definition, a problem-solving method, a domain model, and an adapter that relates the other elements. We present abstract data types and a variant of dynamic logic as formal means to specify and verify these different elements. As a consequence of our architecture we can decompose the overall specification and verification task of the knowledge-based systems into subtasks. We identify different subcomponents for specificationand proof obligations for verification. The use of the architecture in specification and verification improves understandability and reduces the effort for both activities. The modularization enables reuse of specifications and proofs. A knowledge-based system can be build by combining and adapting different *reusable* components.

1 Introduction

During the last years, several conceptual and formal specification techniques for knowledge-based systems (KBSs) have been developed (see [47], [23], [14] for surveys). The main advantage of these modelling or specification techniques is that they enable the description of a KBS independent of its implementation. This has two main implications. First, such a specification can be used as golden standard for the validation and verification of the implementation of the KBS. It defines the requirements the implementation must fulfil. Second, validation and verification of the functionality, the reasoning behavior, and the domain knowledge of a KBS is already possible during the early phases of the development process of the KBS. A model of the KBS can be investigated independently of aspects that are only related to its implementation. Especially if a KBS is built up from reusable