Role in Knowledge Engineering. In *Proceedings of the 12. European Conference on Artificial Intelligence (ECAI-96)*, Budapest, August 12-16, 1996.

[5] 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.

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

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

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

[9] 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.

[10] 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.

[11] 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.

[12] D. Fensel: *The Knowledge Acquisition and Representation Language KARL*, Kluwer Academic Publ., Boston, 1995.

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

[14] 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.

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

[16] 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.

[17] 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.

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

[19] 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.

[20] B. Nebel: Artificial intelligence: A Computational Perspective. In G. Brewka (ed.), *Essentials in Knowledge Representation*, Springer Verlag, 1996.

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

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

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

[24] 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.

[25] W. Reif and K. Stenzel: Reuse of Proofs in Software Verification. In Shyamasundar (ed.), *Foundation of Software Technology and Theoretical Computer Science,* LNCS 761, Springer-Verlag, 1993.

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

[27] 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.

[28] M. Sitaraman, L. R. Welch, and D. E. Harms: On Specification of Reusable Software Components, *International Journal of Software Engineering and Knowledge Engineering,* 3(2):207-229, 1993.

[29] J. W. Spee and L. in 't Veld: The Semantics of $K_{BS}$SF: A Language For KBS Design, *Knowledge Acquisition,* vol 6, 1994.

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

[31] 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.

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

[33] F. van Harmelen and J. Balder: $(ML)^2$: A Formal Language for KADS Conceptual Models, *Knowledge Acquisition,* 4(1), 1992.

[34] G. van Heijst: *The Role of Ontologies in Knowledge Engineering*, PhD thesis, University of Amsterdam, the Netherlands, 1995. To appear in International Journal on Human Computer Studies (IJHCS).

[35] C. A. Vissers, G. Scollo, M. van Sinderen, and E. Brinksma: Specification Styles in Distributed System Design and Verification, *Theoretical Computer Science*, 98:179-206, 1991.

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

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

$$\exists x \, (x \in \mathit{observables})$$

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.

For example, a more serious assumption would be the *single-fault assumption* (cf. [8]). 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. [4]).

## 7 Conclusions and Future Work

In the paper, we introduce a formal and conceptual framework for specifying and verifying KBSs. One can specify tasks, PSMs, 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 a conceptual model 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 PSM is decomposed in external and internal aspects. The specification of the competence of the PSM provide all necessary aspects for relating it with the task that must be solved. When specifying a reusable PSM it must be proven one time whether the operational specification specifies a terminating 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. This requires that the competence specification is complete for the signature that describes the input and the output of the method. In analogy to [28] who discuss the reuse of software components, we view the operational specification of a PSM as its implementation and the competence theory as its external black-box specification. The competence must therefore describe all relevant properties of the PSM to enable reuse without forcing to refer to internal details of the implementation (i.e., operationalization). In the case of the domain model, such an encapsulation is not possible because task and PSM 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 PSM.

In addition to modelling concepts, formal development of KBSs requires tool support for modularisation of specifications and programs and for constructing, analysing, and reusing proofs. The KIV system (Karlsruhe Interactive Verifier) (see [24]) is an advanced tool for the construction of provably correct software. It supports the design process starting from formal specifications (algebraic full first-order logic with loose semantics). Our aim is to adapt the KIV system, originally designed for conventional software engineering, for development and verification of KBSs. For this purpose the KIV system is quite attractive. KIV supports dynamic logic like the specification languages KARL [12], $(ML)^2$ [33], and MLPM [14]. KIV allows structuring of specifications and modularisation of software systems. Finally, the KIV system offers well-developed proof engineering facilities. Especially, automatic reuse of proofs that allows an incremental verification of corrected versions of programs and lemmas (see [25]) is important given the fact that system development is a process of steady modification and revision. [16] and [15] report successful case studies in applying KIV to the verification of KBSs. Currently we are working on problems stemming from differences of the formalization languages of KIV and MLPM, on integrating our conceptual models directly into the generic module concept of KIV, and on proof tactics that make use of this conceptual model.

**Acknowledgement.** We thank Arno Schönegge, Bob Wielinga, Frank van Harmelen, Richard Benjamins, Remco Straatman, Annette ten Teije, Maarten van Someren, Rudi Studer and Stefan Decker 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] T. Bylander, D. Allemang, M. C. Tanner, and J. R. Josephson: The Computational Complexity of Abduction, *Artificial Intelligence*, 49, 1991.

[3] 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.

[4] R. Benjamins, D. Fensel, and R. Straatman: Assumptions of Problem-Solving Methods and Their

problem and domain (cf. [10]).[3]

## 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 purpose. First we demonstrate how to link task and PSM by the $TP_{Adapter}$. Then we discuss their relation 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 [23] 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. In our case, we have to ensure that each possible hypothesis is regarded as input of the method. 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; a possible domain that it contains at least one hypothesis; and the union of all hypotheses of the domain knowledge must be a complete explanation of all observations of the input (see Figure 9). These requirements must be fulfilled by the domain knowledge 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 [15], 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 [2]):

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

For more details how to find such assumptions with interactive theorem provers compare [15].[4]

Whether this property must be stated as asssumption or whether it can be formulated as a requirement on domain knowledge can be decided when specifying the second aspect of the 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 10). 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 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.

```
adapter $TP_{Adapter}$
  include set-minimizer, complete and parsimonious explanation
  rename set-minimizer by abduction
    object → hypothesis, objects → hypotheses,
    Correct → complete
  variables h : hypothesis, x : datum, H,H´: hypotheses
  axioms
      $\forall h\ input(h)$
  proof obligation
      $\forall h\ (Output(h) \rightarrow goal(h))$
  requirements
      $\exists x\ (x \in observables)$
      $\exists h\ Input(h)$
      $\forall H (\forall h\ (h \in H) \rightarrow complete(H))$
  assumptions
      $\forall H,H'(H \subseteq H' \rightarrow expl(H) \subseteq expl(H'))$
endadapter
```

Fig. 9. The $TP_{Adapter}$.

```
adapter $D_{Adapter}$
  include circuit, $TP_{Adapter}$
  rename circuit by $TP_{Adapter}$
    symptom → datum,
  variables h : hypothesis, x : datum, H,H´: hypotheses
  axioms
    $\forall x, H\ (x \in expl(H) \leftrightarrow \exists h\ (h \in H \wedge cause(h,x)))$
  proof obligation
    $Input(battery\text{-}broken) \rightarrow \exists h\ Input(h)$
    $\forall x\ \exists h\ causes(h, x) \rightarrow$
       $(\forall H\ (\forall h\ (h \in H) \rightarrow complete(H)))$
    $(H \subseteq H' \rightarrow \{x \mid \exists h\ (h \in H \wedge causes(h,x))\} \subseteq$
    $\{x \mid \exists h\ (h \in H' \wedge causes(h,x))\})$
    $\rightarrow \forall H,H'(H \subseteq H' \rightarrow expl(H) \subseteq expl(H'))$
  assumption
    $\exists x\ (x \in observables)$
endadapter
```

Fig. 10. The $D_{Adapter}$.

3. Notice that we do not assume complete knowledge of symptoms. Knowing only that $bulb_1$ is not lit does not imply that $bulb_2$ is lit.

4. Assumptions arise as open goals during proving that the competence of the method implies the goal. The proof must fail because it is not possible. Open goals, i.e. the places where the proof fails, indicate the assumptions that are necessary to enable a successful proof.

**dkr** Input
**sorts** *object*
**predicates** *Input* : *object*
**variables** *x* : *object*
**axioms**
$\exists x\ input(x)$
**enddkr**

**dkr** Output
**sorts** *object*
**predicates** *Output* : *object*
**enddkr**

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

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

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

**skr** Correct
**sorts**
*object*,
*objects* **set of** *object*
**predicates**
*Correct* : *objects*
**endskr**

```
init;
repeat
    generate;
    select;
    if ∃x New-node(x) then Node(x) := New-node(x);
until ¬∃x New-node(x)
Output(x) := Node(x);
```

Fig. 5 The specification of knowledge roles and control.

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.

We use a simple and familiar device of two bulbs and one battery to illustrate the different elements. Two possible symptoms can be observed (one of the bulbs is not lit) and three different elementary hypotheses are provided (see Fig. 8). The domain knowledge defines their causal relationships. The meta-knowledge ensues two properties

**competence** *set-minimizer*
**sorts** *object, objects* **set of** *object*
**static predicates**
*Input* : *object*
*Correct* : *objects*
**dynamic predicates**
*Output* : *objects*
**variables** *x* : *object, y ,z* : *objects*
**axioms**
(1) Each *output* was an element of *input*
$[set\text{-}minimizer]\ \forall x, y\ (x \in Output(y) \rightarrow Input(x))$
(2)There is an *output*
$[set\text{-}minimizer]\ \exists y\ Output(y)$
(3) The *output* is correct
$[set\text{-}minimizer]\ \forall y(Output(y) \rightarrow Correct(y))$
(4) The *output* is (locally) minimal
$[set\text{-}minimizer]$
$\neg \exists x,y,z\ (x \in Output(y) \wedge z = y \setminus \{x\} \wedge Correct(z))$
**endcompetence**

Fig. 6 The competence of the PSM.

**requirements** *set-minimizer*
**sorts** *object, objects* **set of** *object*
**static predicates**
*input* : *object*
*correct* : *objects*
**variables** *x* : *object, y* : *objects*
**axioms**
**requirement**
(1) non-emptyness of *input*:
$\exists x\ input(x)$
(2) the set of all input elements is a correct set
$\forall x,y((input(x) \rightarrow x \in y) \rightarrow correct(y))$
**endrequirements**

Fig. 7 The requirements of the PSM.

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 the hypotheses only assume components as broken and no knowledge is provided that constrains the fault behaviour of components. Assuming more broken components only 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 broken wires. 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

**domain model** *circuit*
**signature**
**sorts** *hypothesis, hypotheses* **set of** *hypothesis, symptom*
**functions**
$bulb_1$*-broken* : *hypothesis*
$bulb_2$*-broken* : *hypothesis*
*battery-broken* : *hypothesis*
*no-lit*$_1$ : *symptom*
*no-lit2* : *symptom*
**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**
$causes(battery\text{-}broken, no\text{-}lit_1)$
$causes(battery\text{-}broken, no\text{-}lit_2)$
$causes(bulb_1\text{-}broken, no\text{-}lit_1)$
$causes(bulb_2\text{-}broken, no\text{-}lit_2)$
**assumption**
complete fault knowledge
$\forall h\ (h \neq battery\text{-}broken \vee h \neq bulb_1\text{-}broken$
$\rightarrow \neg causes(h, no\text{-}lit_1))$
$\forall h\ (h \neq battery\text{-}broken \vee h \neq bulb_2\text{-}broken$
$\rightarrow \neg causes(h, no\text{-}lit_2))$
**enddm**

Fig. 8 The domain knowledge.

changing the truth values of a predicate according to the truth values of a formula that is used to define the transition. Two different types of such elementary state transitions exist:

$p :\leftrightarrow \varepsilon x.\varphi$ and $p :\leftrightarrow \lambda x.\varphi$

The ε-operator expresses non-deterministic selection of one ground literal. A formula φ can be used to restrict the set of possible ground literals from which one is chosen. All other ground literals of the predicate *p* are set to false. The λ-operator allows updates of all ground literals of a predicate *p* according to the truth values of a formula φ. All ground literals are set to true for which the according variable assignments evaluate the formula to true. All other ground literals of the predicate *p* are set to false.

MLPM provides the usual procedural constructs such as sequence, if-then-else, choice, and while-loop to define complex transition. We will make use of these constructs when we define the control flow of the entire method. As inference actions are regarded to be primitive they are defined by only one elementary transition.

```
inf init
    sorts object,[1] objects set of object
    static predicates Input : object
    dynamic predicates Node : objects
    variables x : object, y ,z : objects
    axioms
        [init] $\exists y\ \forall x, z(x \in Node(y)$
                $\leftrightarrow Input(x)) \wedge (z = y \vee \neg Node(z)))$
    implementation
        $init =_{def} Node :\leftrightarrow \varepsilon y.\ \forall x\ (Input(x) \rightarrow x \in y)$
endinf

inf generate
    sorts object, objects set of object
    static predicates Node : objects
    dynamic predicates Nodes : objects
    variables x : object, y,z : objects
    axioms
        [generate]$\forall y(Nodes(y) \leftrightarrow \exists x\ \exists z(x \notin y \wedge Node(z)$
            $\wedge\ y = z \setminus \{x\}))$
    implementation
        $generate =_{def} Nodes :\leftrightarrow$
            $\lambda y.(\exists x\ \exists z(x \notin y \wedge Node(z) \wedge y = z \setminus \{x\}))$
endinf

inf select
    sorts object, objects set of object
    static predicates Nodes : objects, Correct : objects
    dynamic predicates New-node : nodes
    variables x,y, z : objects
    axioms
        [select] $\exists y\ (New\text{-}node(y) \leftrightarrow$
            $(Nodes(y) \wedge Correct(y) \wedge$
            $\forall z(z = y \vee \neg New\text{-}node(z)))) \vee \neg\forall x\ New\text{-}node(x)$
    implementation
        $select =_{def} New\text{-}node :\leftrightarrow \varepsilon x.(Nodes(x) \wedge Correct(x))$
endinf
```

1. object must be an enumarable sort.

Fig. 4 The specification of the inference actions.

### 4.1.2 Dynamic Knowledge Roles

Dynamic knowledge roles (dkr) are means to represent the state of the reasoning process and are modelled by algebraic specifications (see Figure 5). Axioms can be used to represent state invariants. We define the requirement that the input provided to the method has to be non-empty.

### 4.1.3 Static Knowledge Roles

Static knowledge roles (skr) are means to include domain knowledge into the reasoning process of a PSM. Again, they are modelled by algebraic specifications. Axioms are used to define requirements on the domain knowledge. Our method *set-minimizer* requires knowledge about correct sets. This is modelled by the static knowledge roles as given in Figure 5.

### 4.1.4 Control Flow

The operational description of a PSM is completed by defining the control flow (see Figure 5) that defines the execution order of the inference actions. Again, we use Modal Logic for Predicate Modification (MLPM) (see section 4.1.1). An elementary state transition is achieved by changing the truth values of a predicate according to the truth values of a formula that is used to define the transition. Complex transitions are built up by defining procedural control (i.e., sequence, branch, and loop) on top of these elementary transitions. The inference action *init* selects the set of all hypotheses as current node. The inference action *generate* generates all successor sets that contain one element less. The inference action *select* tries to select a correct set under this successors. The search process stops if it fails. Otherwise the loop of *generate* and *select* is applied to such a complete successor.

## 4.2 Competence Theory

The competence theory describes the functionality of the PSM. Again algebraic specifications enriched by the modality operators of dynamic logic can be used for this purpose (see Figure 6). As in the description of inference actions we distinguish static and dynamic predicates. The competence theory in Figure 6 defines that the *set-minimizer* is able to find an output set that is a correct and locally minimal subset of the input. Local minimality means, that there is no correct subset of the output that has only one element less.

## 4.3 Requirements

The method has two requirements: there must be an input and the set of all input objects is a correct set. Then one can guaranty that the method finds a local-minimal and correct set (cf. Figure 7).

## 5 The Domain Model

A domain model consists of three main parts: the domain 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 the formalization of our approach. The task *abductive diagnosis* receives a set of observations as input and delivers a complete and parsimonious explanation (see e.g. [2]). 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 [2]. Therefore, we have to

**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*.

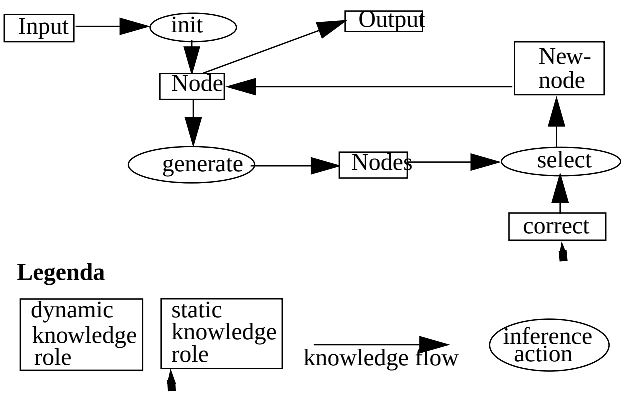

Fig. 3 Knowledge flow diagram of *set-minimizer.*

apply heuristic search strategies. In the following, we characterize such a 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. In the following, we provide the operational specification, the competence, and the requirements of the method.

### 4.1 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 three steps. The inference action *init* construct the set of all input elements as initial set. The inference action *generate* generates all successor sets that contain one element less. The inference action *select* tries to find a valid set under this successors. Figure 3 gives the knowledge flow diagram of the method.

#### 4.1.1 Inference Actions.

We use algebraic specifications enriched by the modality operators of dynamic logic to specify the functionality of inference actions. We distinguish between predicates that have the same truth values in the initial state and in the state after the execution of an inference action (called *static predicates*) from the predicates that change as a result of executing the inference action (called *dynamic predicates*).[2] Figure 4 provides the definition of the three inference actions of the PSM. The functional specification is extended by an operational specification (called implementation) that express the inference in a procedural way. We use a variant of dynamic logic for this purpose. The Modal Logic for Predicate Modification (MLPM) (see [14]) was developed as a generalization of the modelling primitives of the specification languages KARL (see [12]) and $(ML)^2$ (see [33]). In addition to these languages, it provides an axiomatization that enables automated proofs. MLPM represents a state by the truth values of the predicates. An elementary state transition is achieved by

2. Dynamic knowledge roles and static knowledge roles should not get mixed with dynamic and static predicates. The former are determined in the context of the entire PSM and the latter are determined in the context of an individual inference action.

domain knowledge distinguish a PSM from usual software products. Pre-conditions on valid inputs are extended to complex requirements on available domain knowledge.

### 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. [18], [30], [34]). 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. 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 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 the result of a measurement fault, etc. (cf. [10]).

### 2.1.4 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 task definition (cf. [11], [4]). 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. 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.

Usually an adapter introduces new requirements or assumptions because in general, most problems tackled with KBSs are inherently complex and intractable (cf. [2], [20]). 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.

1. In terms of [35], a task definition is an *extensional* specification and a PSM combines *extensional* with *intensional* specification elements. The entire competence of the PSM and their elementary reasoning steps are specified extensionally. The interaction of the elementary reasoning steps in order to achieve the competence is specified intensionally.

## 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. [26]).

(PO-iii)We have 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.
- (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. [32]). The conceptual model applied to describe KBSs is used to brake the general proof obligations into smaller pieces and makes parts of them that should be solved by the KBS; a *problem-solving method* (PSM) that 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 [6]), the reuse of PSMs for different tasks and domains ([22], [6], [3]), and the reuse of domain knowledge for different tasks and PSMs (cf. [18], [30], [34]). 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. It 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 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.). 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. [5], [36]) and have already been applied by [29] and [21] 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.

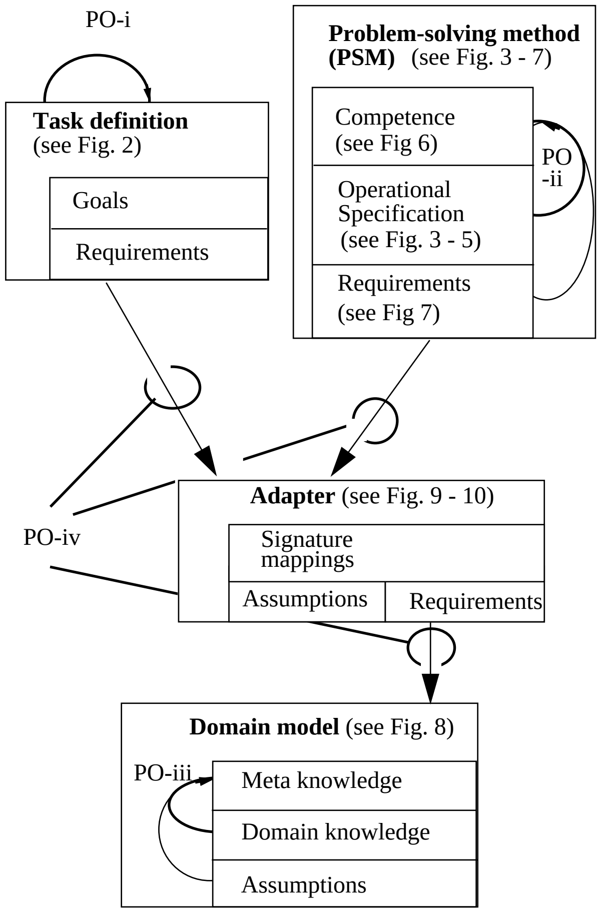

Fig. 1 The four elements of a specification of a KBS.

### 2.1.2 The Problem-Solving Method

The concept PSM is present in a large part of current knowledge-engineering frameworks (e.g. Generic Tasks [7], CommonKADS [6], [27], Method-to-task approach [9], Configurable Role-Limiting Methods [22]). In general, PSMs are used to describe the reasoning process of a KBS. Besides some differences between the approaches, there is strong consensus that a PSM:

- decompose 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, [31] 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 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. 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. [19]) 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 regarded as an implementation issue that is neglected during the specification process of the KBS.[1]

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

# Specifying Knowledge-Based Systems with Reusable Components

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, Dep. of CS, P.O. Box 800, 9700 AV Groningen, NL, rix@cs.rug.nl

**Abstract.** The paper introduces an approach 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 conceptual model we can decompose the overall verification task of the knowledge-based systems into different proof obligations. Each proof obligation deals with a different aspect of the entire system. The use of the conceptual model 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 combing and adapting different components.

## 1 INTRODUCTION

During the last years, several conceptual and formal specification techniques for knowledge-based systems (KBSs) have been developed (see [17], [13] 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 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 [27]) 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 [5]) and dynamic logic (see [19]).

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.

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. In section 3 we introduce an example of a task definition. We discuss a simple abductive problem. In section 4, we provide the definition of a PSM. In section 5, we sketch the definitions of a domain model. Section 6 discuss the role of the adapter and discuss assumptions that are necessary to relate the task description with the competence of the PSM. Section 7 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.

### 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

In *Proceedings of the 9th International Conference on Software Engineering & Knowledge Engineering (SEKE-97)*, Madrid, Spain, June 18-20, 1997.