Using Ontologies For Defining Tasks, Problem-Solving Methods and Their Mappings

D. Fensel[1], E. Motta[2], S. Decker[1], and Z. Zdrahal[2]

[1] Institut AIFB, University of Karlsruhe, D-76128 Karlsruhe,
{dieter.fensel,stefan.decker}@aifb.uni-karlsruhe.de

[2] Knowledge Media Institute, The Open University, Milton Keynes MK7 6AA, United Kingdom,
{e.motta,z.zdrahal}@open.ac.uk

Abstract. In recent years two main technologies for knowledge sharing and reuse have emerged: *ontologies* and *problem solving methods* (PSMs). Ontologies specify reusable conceptualizations which can be shared by multiple reasoning components communicating during a problem solving process. PSMs describe in a domain-independent way the generic reasoning steps and knowledge types needed to perform a task. Typically PSMs are specified in a task-specific fashion, using modelling frameworks which describe their control and inference structures as well as their knowledge requirements and competence. In this paper we discuss a novel approach to PSM specification, which is based on the use of formal ontologies. In particular our specifications abstract from control, data flow and other dynamic aspects of PSMs to focus on the logical theory associated with a PSM (method ontology). This approach concentrates on the competence and knowledge requirements of a PSM, rather than internal control details, thus enabling black-box-style reuse. In the paper we also look at the nature of PSM specifications and we show that these can be characterised in a task-independent style as generic search strategies. The resulting 'modelling gap' between method-independent task specifications and task-independent method ontologies can be bridged by constructing the relevant *adapter* ontology, which reformulates the method ontology in task-specific terms. An important aspect of the ontology-centred approach described here is that, in contrast with other characterisations of task-independent PSMs, it does away with the simple, binary distinction between weak and strong methods. We argue that any method can be defined in either task-independent or task-dependent style and therefore such distinction is of limited utility in PSM reuse. The differences between PSMs which affect reuse concern the ontological commitments which they make with respect to domain knowledge and goal specifications.

1 Introduction

The concept of *generic problem-solving method* (PSM) is present in several knowledge-engineering frameworks (e.g. Generic Tasks [Chandrasekaran et al., 1992]; Configurable Role-Limiting Methods [Puppe, 1993]; CommonKADS [Schreiber et al., 1994]; the Method-To-Task approach [Eriksson et al., 1995]; Components of Expertise [Steels, 1990]; GDM [Terpstra et al., 1993]; MIKE [Angele et al., 1996]). In general a PSM describes in a domain-independent way which reasoning steps and which types of knowledge are needed to perform a task. Libraries of PSMs have been developed, e.g.

To appear in
Proceedings of European Knowledge Acquisition Workshop (EKAW-97), Lecture Notes in Artificial Intelligence (LNAI), Springer-Verlag, 1997.

see [Benjamins, 1995], [Breuker & Van de Velde, 1994], [Chandrasekaran et al., 1992], [Motta & Zdrahal, 1996], and [Puppe, 1993], which support reuse-centred models of KBS development, thus improving the efficiency of the development process and the robustness of the target application system.

In this paper, we look at two fundamental issues associated with PSM specifications: i) the epistemology of the modelling frameworks used to characterise PSMs and ii) the nature of PSM specifications. These issues are discussed below.

Reuse-centred PSM descriptions

Describing PSMs in the style of CommonKADS [Schreiber et al., 1994] requires to specify much of the internal reasoning process of a PSM. In particular, the following descriptions need to be given:

1) the internal reasoning steps of the PSM;
2) the data flows between the reasoning steps;
3) the control that guides the dynamic execution of the internal reasoning steps;
4) the knowledge requirements of a PSM;
5) the goals that can be achieved by a PSM.

However, most of these aspects have to do with understanding how a PSM achieves its goals. To assess the applicability of a PSM one only needs knowledge about its competence and domain requirements - i.e. (4) and (5) above ([van de Velde, 1988], [Akkermans et al., 1993]). In particular, our approach to the specification of the competence of a PSM makes use of *formal ontologies*. This approach provides two main advantages:

- A formal specification adds a precise meaning and enables mechanised support. Specifications in natural language are necessarily imprecise, contain ambiguity and are difficult to verify for completeness and consistency [van Harmelen & Fensel, 1995]. Moreover, establishing the competence of a PSM in relation to some ontological assumptions may require difficult proof processes that are only realistic if some mechanised proof support can be provided [Fensel & Schönegge, 1997].
- An ontology provides “an explicit specification of a conceptualization“ [Gruber, 1993], which can be shared by multiple reasoning components communicating during a problem solving process. Using ontological engineering for describing PSMs provides two important benefits with respect to reuse. The resulting PSM specification i) is grounded on a common, shared terminology and ii) its knowledge requirements are conceptualised as ontological *commitments* [Gruber, 1995].

The nature of PSM descriptions

PSMs are normally described in a task-specific way. There are two main reasons for this. Task-independent PSMs are often regarded as *weak* methods [Mc Dermott, 1988] and much KBS research of the past two decades can be seen as a reaction to the weak - i.e. task-independent - problem solving paradigms used in the sixties and early seventies [Newell & Simon, 1972]. This reaction has taken different forms and has been formulated according to different principles - see for instance the *knowledge as power principle* [Lenat & Feigenbaum, 1987] and the *knowledge interaction hypothesis*

[Bylander & Chandrasekaran, 1988]. But essentially the message here is that efficient, knowledge-based problem solving subscribes to either task- or application-specific paradigms. The other reason which explains the limited 'appeal' of task-independent PSMs is the trade-off between *usability* and *reusability* [Klinker et al., 1991]. The more reusable a PSM, the larger the distance between this PSM and an application specification, which means that more work is required to bridge the representation gap between the PSM and the application.

Nonetheless, in recent years there has been renewed interest in task-independent specifications of PSMs (cf. [Beys et al., 1996][van Heijst and A. Anjewerden, 1996]). van Heijst and Anjewerden point out that the "task specific formulation of PSMs unnecessarily limits the applicability of PSMs" [van Heijst and A. Anjewerden, 1996], and suggest that the applicability conditions of PSMs can be specified in terms of domain-independent meta-characteristics of the target domain model.

However, the trade-off between usability and reusability requires techniques which facilitate the process of configuring a task-independent PSM for a particular task and domain. We see this problem as one of *ontology mapping* and in this paper we characterise PSM configuration as a specialization process, during which ontological commitments are introduced. Specifically, our approach comprises three phases:

- The specification of method-independent ontologies which define task or problem types.
- The task-independent specification of generic search methods.
- The specification of *adapters* [Fensel & Groenboom, 1997] which map the ontology of a generic search method to task-specific terms to produce task-specific PSMs. Because adapters can be stacked on top of each other (cf. [Fensel, 1997b]) we view the task-specific refinement of generic search strategies as a stepwise process overcoming the dualistic view of weak and strong PSMs. Hence, our approach postulates a continuum between both extremes where a step into a more task-specific variant of a PSM is achieved by means of the relevant adapter.

Contents of the paper

In the following we will illustrate the approach in a formal way, by discussing a test case which involves the configuration of a task-independent specification of a PSM for a class of tasks. Specifically, we will discuss the following model components.

- Section 2 describes a specification of a method-independent ontology for *parametric design* tasks [Wielinga et al., 1995][Motta & Zdrahal, 1996].
- Section 3 specifies a *method ontology* for a propose & revise PSM [Marcus & McDermott, 1989] [Zdrahal & Motta, 1995]. This method ontology expresses the competence of propose & revise in terms of assumptions over the properties of two types of state transitions: *propose* transitions and *revise* transitions. The former enrich the completeness of a state, the latter specify a transition from an incorrect to a correct state. No further commitments are made. This method specification is highly task-independent. It can be applied to any task whose problem space can be expressed as a search process on correct, complete and preferred states.
- Section 4 describes the *adapter* which specialises the propose & revise method

ontology for parametric design. Similar adapters can be defined to adapt propose & revise to other tasks - see [Fensel, 1997a] for a description of a configuration of a propose & revise PSM for assignment tasks.

Finally, section 5 discusses the significance of the work, compares our approach to alternative proposals, and outlines directions of future research.

2 A Task Ontology for Parametric Design

In the following we present a task ontology for parametric design. First, we provide an informal description of the ontology. Then we discuss a specification using Sloppy-logic (S-logic). S-logic makes use of rich semantic modelling primitives within the framework defined by Frame-logic [Kifer et al., 1995] - see Fig. 1.

2.1 A Sketch of Parametric Design

Design can be characterised in generic terms as the process of constructing artifacts. Thus, the essential feature of design problem solving is its *constructive* nature: solutions are constructed rather than retrieved from a pre-existing set.

In order to construct an artifact one needs some *building blocks* - i.e. a *technology* [Chandrasekaran, 1990]. In addition, the design process is subjected to a number of *constraints*, which can be related either to the design technology - e.g. technological limitations impose constraints on the minimum size of supporting walls - or to external factors - e.g. most civilised countries require a minimum ceiling height in living rooms.

Design is a goal-driven process, where the goals are specified in terms of a number of functionalities which the target artifact should provide. A good way of informally characterizing the goal-oriented nature of the design process is to see it as driven by *needs* and *desires* [Wielinga et al., 1995]. Thus, the design process[1] can be characterised as a function which takes as input a set of needs, desires, constraints and a possibly incomplete set of building blocks, and produces an artifact as output.

A restricted class of design problems is *configuration design* [Mittal & Frayman, 1989], which can be defined as a design problem where all building blocks are given as input to the design process. [Mittal & Frayman, 1989] show that the complexity of the configuration task decreases significantly by assuming that “the artifacts are configured according to some known functional architectures“. For instance, a computer configuration can be functionally described in terms of processor, printing, memory, data communication, etc. This assumption makes it possible to impose a structure on the space of feasible designs, thus restricting the number of possible configurations. This assumption can be characterised as postulating the existence of one or more *functional solution templates*.

A stronger assumption, which further restricts the space of possible designs is that which postulates the existence of a *parametrised solution template* for the target artifact. In this scenario design problem solving can be described as the process of assigning values to *design parameters* in accordance with the given needs, constraints, and desires. Applications for which this assumption holds are called *parametric design*

1. This is of course an idealised description of the design process.

applications.

The VT elevator design problem [Marcus et al., 1988] [Yost & Rothenfluh, 1996] provides a well-known example of a parametric design task. Here the problem is to configure an elevator in accordance with the given requirement specification and the applicable constraints. The parametrised solution template consists of 199 design parameters which specify the various structural and functional aspects of an elevator - e.g. number of doors, speed, load, etc.

More precisely a parametric design application can be characterised as a mapping from a six-dimensional space <P, VR, C, R, Pr, cf> to a set of solution designs, { D_{sol_1} ,......., D_{sol_n} }, where

- P = Parameters = {p_1,......, p_n};
- Vr = Value ranges = {V_1,......, V_n}, where V_i = {v_{i1},....., v_{in_i} };
- C = Constraints = {c_1,....., c_m};
- R = Requirements = {r_1,......, r_k};
- Pr = Preferences = {pr_1,......., pr_j};
- cf = Cost Function.

These concepts are discussed in the next section, where we illustrate a semi-formal specification - given in S-logic - of the different entities which make up the parametric design task ontology.

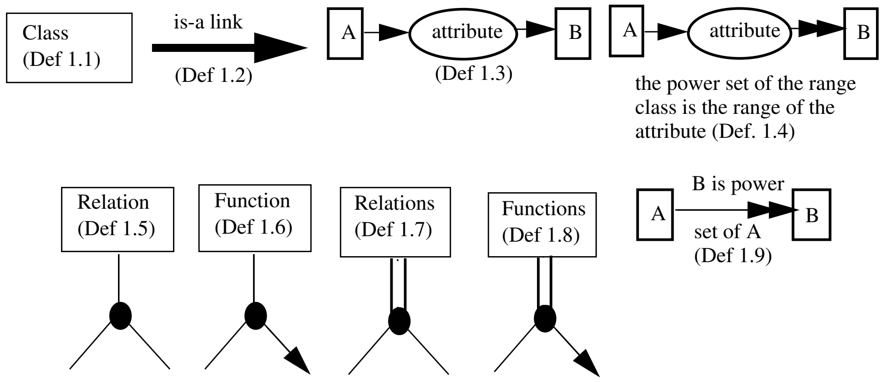
2.2 A Semiformal Specification of Parametric Design with S-Logic

The basic element of parametric design is the concept of *parameter*. We model parameters by means of a class, Parameter (see Def 2.1 in Fig. 2). Each parameter has a value range, which constrains the possible values which the parameter can take in a design model. The union of all value ranges defines the set of all legal parameter values. The latter is modelled by means of class Value range. The attribute allowed values associates a range with a set of values (Def 2.2). The attribute range models the association between a parameter and a range (Def 2.1).

One could also expect that a parameter is directly associated with a value. But a parameter gets a value assigned by a design model and has not a value by itself. That is, there is no functional (i.e., attributional) dependency between parameters and values. It is a design model that introduces a mapping between parameters and values. Such a design model can therefore be modelled as a (partial) function between the parameters and values provided by their ranges (see Def 2.4.a). Incomplete design models, i.e., design models that do not assign a value to each parameter are partial functions. The class Design model models the solution space of our task.

Two further sets are used to characterize subspaces of the solution space. Requirements should be satisfied by a solution and constraints should not be violated by it. We can model a requirement as the set of design models that fulfils it and a constraint as the set of design models that violate it. Therefore, we introduce two classes, Requirement and Constraint, defined as sets of design models (see Def 2.5, Def 2.6 & Def 2.7). These definitions should not be confused with their extensional or intensional representation. The actual set of design models that fulfil a requirement can be described extensionally

Graphical notation



Linear notation

Def 1.1 A class definition
A

Def 1.2 An is-a link
$A \leq B$

Def 1.3 A class definition with an attribute having B as range.
$A(\text{attribute} \rightarrow B)$

Def 1.4 A class definition with an attribute having the power set of B as range
$A(\text{attribute} \rightarrow$ **SET OF** $B)$

Def 1.5 A class describing a relation (i.e., each element of the class is an element of the relation)
$A(...): B_1 \times B_1 \times ...$

Def 1.6 A class describing a function
$A(...): B_1 \times B_1 \times ... \times B_n \rightarrow C$

Def 1.7 A class describing a set of relations (i.e., each element of the class is a relation)
$A(...):: B_1 \times B_1 \times ...$

Def 1.8 A class where each element of the class is a function
$A(...):: B_1 \times B_1 \times ... \times B_n \rightarrow C$

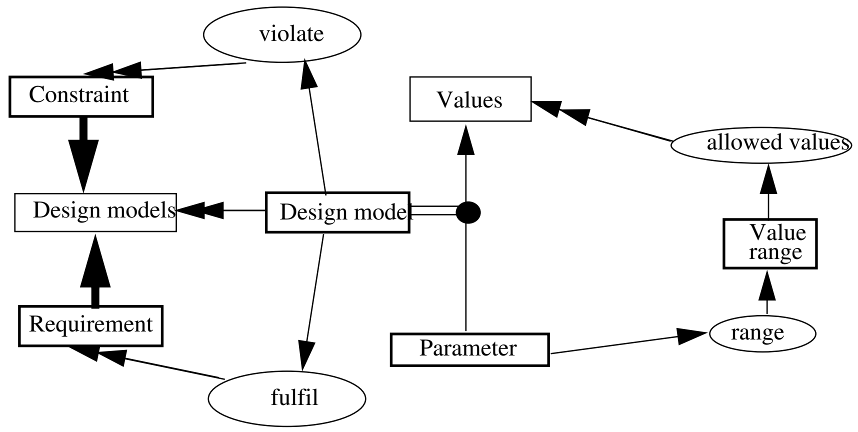
Def 1.9 The class B is the powerset of class A
B : **SET OF** A

Fig. 1 Legend of S-logic.

by enumerating the design models or by providing an intensional characterisation of the sufficient and necessary conditions for fulfilling the requirement.[2]

The attributes fulfil and violate deliver for each design model the set of fulfilled requirements and violated constraints (see Def 2.4.a). The two axioms, Ax 2.1 and Ax 2.2, ensure the appropriate relationships between design models, requirements, and

2. The reader might find surprising that requirements and constraints are modelled as set of design models. The advantage of the approach we have chosen is that it does not require to postulate the existence of an additional domain lexicon to support the specification of constraints and requirements. For instance, the parametric design framework discussed in [Wielinga et al., 1995] introduces a domain vocabulary and the notion of domain theory. Here, we have chosen to minimise the number of concepts required to discuss the ontology and therefore we characterise requirements and constraints as subset of design models. However, the rationale for this choice is pragmatic, rather than ontological.



Def 2.1 Parameter(range → Value range)

Def 2.2 Value range(allowed values → **SET OF** Value)

Def 2.3 Value

Def 2.4.a Design model :: Parameter → Value

Def 2.4.b Design model(fulfil → **SET OF** Requirement, violate → **SET OF** Constraint)
with for all d ∈ Design model holds:
d(p) ∈ allowed values(range(p)) if d is defined for p.

Def 2.5 Design models : **SET OF** Design model

Def 2.6 Requirement ≤ Design models

Def 2.7 Constraint ≤ Design models

Ax 2.1 For all d ∈ Design model and r ∈ Requirement holds: d ∈ r ↔ r ∈ fulfil(d)

Ax 2.2 For all d ∈ Design model and c ∈ Constraint holds: d ∈ c ↔ c ∈ violate(d)

Fig. 2 Entities in the parametric design task ontology.

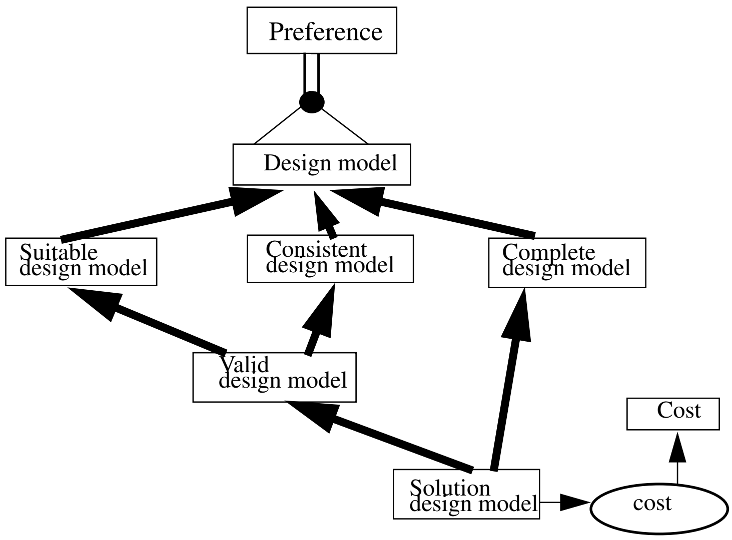
constraints.

Figure 3 shows a taxonomy of types of design models. A *solution* design is defined as a design model which is both *valid* and *complete* (Def 3.6). A valid design model is *suitable* and *consistent* (Def 3.5). A suitable design model fulfils all requirements (Def 3.2 and Ax 3.1). A consistent design model does not violate any constraint (Def 3.3 and Ax 3.2). A complete design model is one in which each parameter has a value (Def 3.4 and Ax 3.3).

Def 3.1 introduces the notion of *preferences*. Preferences describe task knowledge which, given two design models, D_1 and D_2, is used to specify which of the two - if any - is the 'better' one, in accordance with some criterion. We model each preference as a binary relation which specifies a partial order over design models. Each element of the class Preference is such a relation.

While preferences typically define local criteria to choose between alternative design models, a cost function provides a global criterion for assessing the cost of a design model. The class Cost has to be well-founded to introduce an order on costs (Def 3.7).

The ontology associates a cost with a solution design, rather than with a generic design



Def 3.1 Preference:: Design model x Design model

Def 3.2 Suitable design model ≤ Design model

Def 3.3 Consistent design model ≤ Design model

Def 3.4 Complete design model ≤ Design model

Def 3.5 Valid design model ≤ Suitable design model, Consistent design model

Def 3.6 Solution design model(cost → Cost)
≤ Valid design model, Complete design model

Def 3.7 Cost has an associated total order.

Ax 3.1 $d \in$ Suitable design model $\leftrightarrow$
($d \in$ Design model $\wedge$ for all $r \in$ Requirement holds $r \in$ fulfil (d))

Ax 3.2 $d \in$ Consistent design model $\leftrightarrow$ ($d \in$ Design model $\wedge$ violate(d) = $\emptyset$)

Ax 3.3 $d \in$ Complete design model $\leftrightarrow$
($d \in$ Design model $\wedge$ $p \in$ Parameter $\rightarrow \exists$ v . ($v \in$ range(p) $\wedge$ $d(p) = v$))

Ax 3.4 goal $\in$ Solution design model

Ax 3.5 $\neg \exists$ d . ($d \in$ Solution design model $\wedge$ cost(d) < cost(goal) $\wedge$ $d \neq$ goal)

Fig. 3. Types of design models.

model, in order to minimise *ontological commitments* [Gruber, 1993]. However, it is feasible to envisage situations in which cost is defined also for non solution models. For instance, a design system which uses a case-based approach to design could use cost as one of a number of criteria for selecting a design model from a case library. In this scenario the selected model is not necessarily a solution model.

Finally, Ax 3.4 and Ax 3.5 define the goal of the task. It is to find the optimal solution design - i.e. the cheapest of all possible solution design models.

3 A PSM-Ontology for Propose & Revise

A well-known problem solving method which can be used for parametric design problem solving is *propose & revise* [Marcus & McDermott, 1989] [Zdrahal & Motta, 1995]. Actually, as argued in [Zdrahal & Motta, 1995] the term 'propose & revise' is better used to describe a class, rather than a specific PSM; different control regimes and revision strategies can be used within the basic propose & revise framework. The basic idea underlying propose & revise is the use of *knowledge-based backtracking* to focus search. That is, instead of either backtracking to the last chronological choice point or using dependency-directed backtracking, a propose & revise problem solver reacts to inconsistency by means of application-specific *fix knowledge*. This approach removes the need for 'blind' backtracking, thus improving the performance of the problem solver.

In the following, we define method ontologies for both 'weak' and 'strong' versions of propose & revise. The first specification, which is shown in section 3.1, describes a problem solver which is assumed to be capable of finding complete, correct and optimal states. To achieve this competence strong assumptions on available (heuristic) knowledge have to be made. In particular the method assumes that the available propose and fix knowledge is sufficient to reach complete, correct and optimal states.

This specification is obviously quite 'optimistic' and therefore in section 3.2 we will discuss a more 'realistic version of propose & revise, which replaces the assumption of global optimality with one which only assumes locally optimal transitions, thus exhibiting weaker competence.

3.1 Propose and Revise with optimal Competence

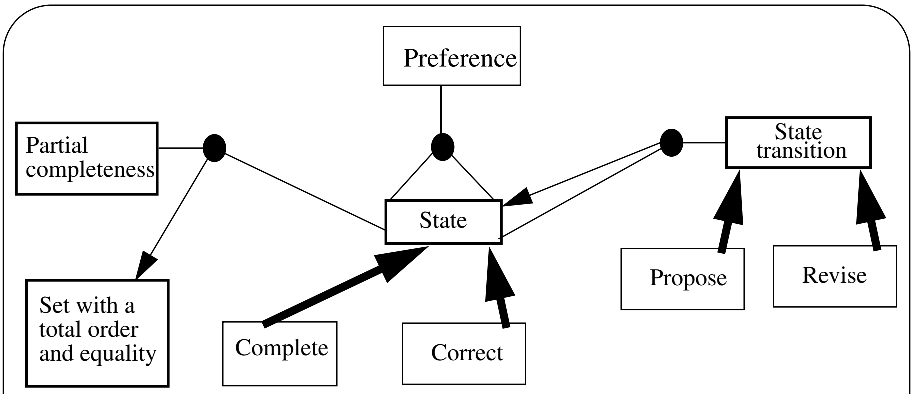
We take a state-based view to describe task-independent PSMs. That is, PSMs are described in terms of state (cf. Def 4.1) and (elementary and complex) state transition (Def 4.6). Such an approach is in accordance with [Motta & Zdrahal, 1996] who characterise KBS problem solving as a search-based process. Thus, in the following, we provide a task-independent specification of propose & revise as a search-based process.

Propose & revise distinguishes two types of state transitions and, as a result, two types of states these transitions are applied to. A Propose transition (cf. Def 4.7) extends the *completeness* of a state while the Revise transition (cf. Def 4.8) transforms an illegal state into a *correct* one. Thus, the ontological commitment here is not formulated in task-specific terms but in terms of generic notions such as completeness and correctness. In other words we assume that the problem solver is able to identify complete and correct states (cf. Def 4.3 and Def 4.5).

Def 4.2 introduces a preference on states that is used to model the assumption that propose and revise steps in our problem solver always result into optimal states (Ax 4.2, Ax 4.4, Ax 4.7). Of course this assumption formulates very strong requirements on both propose and revise knowledge. Specifically, it assumes that the search space defined by propose and revise steps is complete with respect to optimality. This assumption is much too strong, for instance it is not satisfied by the fix knowledge in the VT domain, and therefore in the next section we will substitute it with a weaker notion of optimality, which is contextualised with respect to the space of possible moves.[3]

The competence of the method is characterized by Ax 4.1 and Ax 4.2. The output is a complete, correct and optimal state. To achieve this competence the following

3. Not even completeness is guaranteed in the VT case.



Terminology

Def 4.1 State
Def 4.2 Preference: State x State
Def 4.3 Complete $\leq$ State
Def 4.4 Partial completeness : State $\rightarrow$ set with a associated total order and equality
Def 4.5 Correct $\leq$ State
Def 4.6 State transition : State $\rightarrow$ State
Def 4.7 Propose $\leq$ State transition
Def 4.8 Revise$\leq$ State transition

Competence

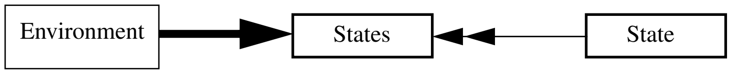
Ax 4.1 output $\in$ State
Ax 4.2 The output is a complete, correct and optimal state
Complete(output) $\wedge$ Correct(output) $\wedge$
$\neg\exists$ s . (s $\in$ State $\wedge$ Complete(s) $\wedge$ Correct(s) $\wedge$ (output,s) $\in$ Preference))

Knowledge Requirements

Ax 4.3 The propose knowledge never fails and monotonically extends the state
$\neg$ Complete(s) $\rightarrow$ Partial completeness(s) < Partial completeness(propose(s))
Ax 4.4 The application of a propose leads to an optimal state
$\neg$ Complete(s) $\rightarrow$
$\neg\exists$ s´ . (s´ $\in$ State $\wedge$ Correct(s´) $\wedge$ propose(s) < s´ $\wedge$
Partial completeness(s´) = Partial completeness(propose(s))
Ax 4.5 The revise knowledge never fails
$\neg$ Correct(s) $\rightarrow$ Correct(revise(s))
Ax 4.6 The application of revise does not change the completeness of a state
Partial completeness(revise(s)) = Partial completeness(s)
Ax 4.7 The application of revise leads to an optimal state
$\neg$ Correct(s) $\rightarrow$
$\neg$ $\exists$ s´ . (s´ $\in$ State $\wedge$ Correct(s´) $\wedge$
Partial completeness(revise(s)) = Partial completeness(s´) $\wedge$ revise(s) < s´)

Fig. 4. A method ontology for a globally optimal propose & revise.

requirements on propose and revise knowledge are necessary: the propose knowledge never fails and monotonically extends the state (Ax 4.3); the application of a propose



Terminology

Def 5.1 States : **SET OF** State
Def 5.2 Environment $\leq$ States

Competence

Ax 4.2´ The output is a complete, correct and local optimal state

$\text{Complete(output)} \wedge \text{Correct(output)} \wedge \text{local-optimal(output)}$

$\text{local-optimal}(x) \leftrightarrow$
$\exists s\, \forall s'\, .\, (s \in \text{State} \wedge \neg\, (\text{Complete}(s) \wedge \text{Correct}(s)) \wedge x \in \text{Environment}(s) \wedge$
$s' \in \text{Environment}(s) \wedge \text{Complete}(s') \wedge \text{Correct}(s')$
$\rightarrow \neg(x,s') \in \text{Preference})$

Knowledge Requirements

Ax 4.4´ The application of a propose leads to a local optimal state

$\neg\, \text{Complete}(s) \rightarrow$
$\neg\exists s'\, .\, (s' \in \text{State} \wedge \text{Correct}(s') \wedge \text{propose}(s) < s' \wedge$
$\text{Partial completeness}(s) < \text{Partial completeness}(s')$
$= \text{Partial completeness}(\text{propose}(s) \wedge s' \in \text{Environment}(s))$

Ax 4.7´ The application of revise leads to a a local optimal state

$\neg\, \mathit{Correct}(s) \rightarrow$
$\neg\, \exists\, s'\, .\, (s' \in \mathit{State} \wedge \mathit{Correct}(s') \wedge$
$\mathit{Partial\ completeness}(\mathit{revise}(s)) = \mathit{Partial\ completeness}(s') \wedge \mathit{revise}(s) < s' \wedge$
$s' \in \text{Environment}(s))$

Fig. 5. An extended method ontology for a locally optimal propose & revise.

transition leads to an optimal state (Ax 4.4); the revise knowledge never fails (Ax 4.5); the application of a revise transition does not change the completeness of a state (Ax 4.6); and the application of a revise transition leads to a (not necessarily complete) optimal design model (Ax 4.7). We need the notion of partial completeness of states for formulating the requirement that propose does monotonically extend the completeness of a state and revise does not change this partial completeness (Def 4.4). This completes our description of a method ontology for an optimal and complete propose & revise problem solver.

3.2 Weakening of Propose and Revise

Instead of requiring global optimality, it may only be realistic to require local optimality. That is, propose and revise transitions are only optimal with respect to same local environment of the current state. We can use this idea to generalize the definition of the Ax 4.2, Ax 4.4, and Ax 4.7.

These generalized definitions are provided in Fig. 5. Ax 4.4´ and Ax 4.7´ ensure local optimality of propose and revise transitions. According to the properties of the domain knowledge used to define such environments these axioms define relatively weaker or stronger requirements. The definitions correspond to the original axioms of section 3.1 in the case where each environment consists of the entire problem space.

A drawback of this generalized specification is the weakened notion of the competence of the method. We can guarantee optimality only in the sense that a successor state is a local optimum of the environment of a predecessor state (see Ax 4.2´).

4 Integrating Method and Task Ontologies to Produce Task-specific Versions of a PSM

In the previous sections we have specified a task ontology for parametric design and weaker and stronger versions of a task-independent propose & revise problem solver. It remains to connect both, i.e. to configure propose & revise for parametric design.

This step consists of producing the relevant *adapter* - see Fig. 6.[4] The main ontological decision is to interpret a state in terms of a design model. Specifically, the definitions in Fig. 6 identify correct states with valid design models, complete states with complete design models, and partial completeness with the set of parameters which are assigned a value by a design model. Order and equality of partial completeness are defined in terms of set inclusion and set equality of parameter sets.

The crucial axiom is Ax 6.3, which ensures that the output of the method corresponds to the goal of the task. The proof of this axiom is rather trivial. The main effort in verifying our ontological specification is to ensure the optimality of the output of propose and revise as assumed by Ax 4.2. Methodologies and tools for verifying such axioms are described in [Fensel & Schönegge, 1997].

5 Conclusions, Related and Future Work

In this paper we have defined i) an ontology for the parametric design task, which is independent of any domain and PSM; ii) an ontology for propose & revise, which is independent of any domain and task model; and finally iii) an adapter which integrates task and method specifications. The resulting model constitutes a task-specific variant of propose & revise: propose & revise for parametric design. Thus, we have provided descriptions which enable reuse of task specifications as well as reuse of task-

Def 6.1 $\text{State}_{P\&R}$: Design model_{PD}

Def 6.2 $\text{Complete}_{P\&R}$: Complete design model_{PD}

Def 6.3 Set with a total order and $\text{equality}_{P\&R}$: **SET OF** parameter_{PD}

Def 6.4 $<_{P\&R} : \subset_{PD}$, $=_{P\&R} : =_{PD}$

Def 6.5 $\text{Correct}_{P\&R}$ = Valid design model_{PD}

Ax 6.1 $\text{Preference}_{P\&R}(s_{P\&R}, s'_{P\&R}) \leftrightarrow \text{cost}(d_{PD}) < \text{cost}(d'_{PD}) \wedge s_{P\&R} = d_{PD} \wedge s'_{P\&R} = d'_{PD}$

Ax 6.2 Partial $\text{completeness}_{P\&R}(d_{P\&R}) = \{p_{PD} \mid (p_{PD}, v_{PD}) \in d_{PD}\}$

Ax 6.3 $\text{output}_{P\&R} = \text{goal}_{PD}$

Fig. 6. The PSM ontology of propose & revise for parametric design.

4. The infix-notation in is used to distinguish terms from parametric design (infix PD) and terms from propose & revise (infix P&R).

independent and task-specific PSMs.

A number of authors have discussed task-independent PSM specifications. In particular [van Heijst and A. Anjewerden, 1996] formulate task-independent PSMs in terms of *acceptance criteria*. For instance, their task-independent formalization of cover & differentiate states that the 'cover' part of the method is applicable to domains where there is a relation X, which is transitive, anti-symmetric and anti-reflexive. However, this formulation does not say anything about the feasibility of applying the method to a domain. In other words it does not provide a replacement problem solving model (or paradigm) for the task-specific model provided by cover & differentiate. As a matter of fact most of the time it won't make much sense to apply the cover sub-method to a relation with the given properties. In contrast with this approach our task-independent PSMs are described as variations of search methods - i.e. they substitute a task-specific problem solving paradigm with a generic one.

The approach taken by [van Heijst and A. Anjewerden, 1996] also postulates a clear separation between task-specific and task-independent descriptions of PSMs. A similar, dualistic view is also taken by McDermott [Mc Dermott, 1988], who compares role-limiting methods to weak methods and indicates that “a weak method is more open with respect to control than a role-limiting method can be; a weak method does not put any limits on the nature or complexity of the task-specific control knowledge it uses“. We believe this characterization of strong vs weak methods is too coarse-grained. For instance a weak method such as A* makes precise assumptions about the existence of heuristic knowledge, which make it possible to converge to an optimal solution. Moreover, as this paper shows, it is possible to reformulate a strong method such as propose & revise as a specialised, but task-independent search mechanism. So, is such a task-independent propose & revise a weak or a strong method? Basically it is a search algorithm - just like hill-climbing and A* - which makes strong assumptions about the availability of domain-specific knowledge to avoid backtracking when an inconsistent state is found. In a nutshell there are no strong and weak methods, there is a continuum in which one can define search strategies which make stronger and stronger assumptions about the availability of task-specific control knowledge. But there is no sharp dividing line.

In this paper we have characterised this method specialization process as one which consists of carrying out the appropriate ontology mappings. This approach clearly separates the specification of a task-independent reasoning strategy from the issue of adapting the strategy to alternative classes of tasks. Another example of this approach is given in [Fensel et al., 1996] who investigated the board game method. This method is a refinement of chronological backtracking for one-player board games. Therefore, the board game method is already a task-specific refinement of a generic search strategy. However, it can be further specialized to become applicable to assignment or parametric design tasks (cf. [Eriksson et al., 1995]).

The use of ontologies for characterising tasks and PSMs is not completely new. For example, [ten Teije, 1997] defines a task ontology for diagnostic problems and [Gennari et al., 1994] have proposed to use ontologies as a way to specify the knowledge requirements of a problem solving method. However, the latter define mainly names for knowledge types and neither approach provides a black-box-style specification of the competence of PSMs ([ten Teije, 1997] abstracts from all heuristic search knowledge that is essential in specifying PSMs). In contrast with these approaches we use formal ontologies to provide declarative, axiomatic specifications of both generic problem classes (task ontologies) and PSMs (method ontologies). Moreover, we demonstrated the use of adapters as a formal technique to integrate tasks and method ontologies.

An ontology-driven bottom-up development process is described in [Reynaud & Tort, 1997], which shows how to derive a reasoning method from an expert ontology. Therefore, their aim is orthogonal to our approach, which is concerned with the reuse of existing reasoning methods. However, it would be worthwhile to investigate whether their approach can be used to select and refine PSMs based on the ontological characterization proposed here.

[Mizoguchi et al., 1995] use ontologies to characterize tasks. A task ontology is used to decompose a task into subtasks, to identify the required knowledge types and to construct a problem solver that simulates the problem-solving behaviour of a domain expert. Again, the main distinction to our approach is that we aim for a declarative (black-box) characterization of problems and PSMs which does not refer to internal details of the problem-solving process.

In conclusion, in this paper we have shown an approach which centres on formal, axiomatic, reuse-oriented specifications of generic KBS components. Like all research on software reuse, it is the ultimate effectiveness of our approach will have to be validated empirically, by trying it out on a number of different application domains.

Acknowledgement. We thank Annette ten Teije, Frank van Harmelen, Mark Willems, and two anonymous reviewers for helpful comments on drafts of the paper.

6 References

[Akkermans et al., 1993] 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.

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

[Benjamins, 1995] 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.

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

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

[Bylander & Chandrasekaran, 1988] T. Bylander, and B. Chandrasekaran: Generic Tasks in Knowledge-Based Reasoning: The Right Level of Abstraction for Knowledge Acquisition. In B. Gaines et al. (eds.), *Knowledge Acquisition for Knowledge-Based Systems*, vol 1, pp. 65—77. Academic Press, London, 1988.

[Chandrasekaran, 1990] B. Chandrasekaran: Design Problem Solving: A Task Analysis. *AI Magazine*, 11(4):59—71, Winter Issue, 1990.

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

[Eriksson et al., 1995] 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.

[Farquhar et al., 1997] A. Farquhar, R. Fickas, and J. Rice: The Ontolingua Server: a Tool for Collaborative Ontology Construction, *International Journal of Human-Computer Studies (IJHCS)*, 46(6):707—728, 1997.

[Fensel, 1997a] D. Fensel: An Ontology-based Broker: Making Problem-Solving Method Reuse Work. In *Proceeedings of the Workshop on Problem-Solving Methods for Knowledge-based Systems (W26) during IJCAI-97,* Japan, August 23, 1997.

[Fensel, 1997b] D. Fensel: The Tower-of-Adapters Method for Developing and Reusing Problem-Solving Methods. To appear in *Proceedings of European Knowledge Acquisition Workshop (EKAW-97)*, LNAI, Springer-Verlag, 1997.

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

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

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

[Gennari et al., 1994] Gennari, J. H., Tu, S. W., Rothenfluh, T. E., Musen, M. A. Mapping Domains to Methods in Support of Reuse. In *Proceedings of the 8th Banff Knowledge Acquisition Workshop (KAW-94)*, Banff, Canada, 1994.

[Gruber, 1993] T. R. Gruber: A Translation Approach to Portable Ontology Specifications, *Knowledge Acquisition*, 5(2), 1993.

[Gruber, 1995] T. R. Gruber: Toward Principles for the Design of Ontologies Used for Knowledge Sharing, *International Journal of Human-Computer Studies (IJHCS)*, 43(5/6):907—928, 1995.

[Kifer et al., 1995] M. Kifer, G. Lausen, and J. Wu: Logical Foundations of Object-Oriented and Frame-Based Languages, *Journal of the ACM*, 42, 1995.

[Klinker et al., 1991] G. Klinker, C. Bhola, G. Dallemagne, D. Marques, and J. McDermott: Usable and Reusable Programmin Constructs, *Knowledge Acquisition,* 3:117—136, 1991.

[Lenat & Feigenbaum, 1987] D. B. Lenat and E. A. Feigenbaum: On the Thresholds of Knowledge. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI-87)*, 1987.

[Marcus et al., 1988] S. Marcus, J. Stout, and J. McDermott VT: An Expert Elevator Designer That Uses Knowledge-based Backtracking, *AI Magazine*, 9(1):95—111, 1988.

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

[Mc Dermott, 1988] J. Mc Dermott: Preliminary Steps Toward a Taxonomy of Problem-Solving Methods. In S. Marcus (ed.). *Automating Knowledge Acquisition for Experts Systems*, Kluwer Academic Publisher, Boston, 1988.

[Mittal & Frayman, 1989] S. Mittal and F. Frayman: Towards a Generic Model of Configuration Tasks. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - IJCAI '89*, San Mateo, CA, Morgan-Kaufman, 1989.

[Mizoguchi et al., 1995] R. Mizoguchi, J. Vanwelkenhuysen, and M. Ikeda: Task Ontologies for reuse of Problem Solving Knowledge. In N. J. I. Mars (ed.), *Towards Very Large Knowledge Bases,* IOS Press, 1995.

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

[Newell & Simon, 1972] A. Newell and H. A. Simon: *Human Problem Solving*, Prentice Hall, 1972.

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

[Reynaud & Tort, 1997] C. Reynaud and F. Tort: Using Explicit Ontologies to Create Problem Solving Methods, *International Journal of Human-Computer Studies (IJHCS)*, 46:339—364, 1997.

[Schreiber et al., 1994] 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.

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

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

[Terpstra et al., 1993] 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.

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

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

[van de Velde, 1988] 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.

[van Harmelen & Fensel, 1995] F. van Harmelen and D. Fensel: Formal Methods in Knowledge Engineering, *The Knowledge Engineering Review*, 10(4), 1995.

[van Heijst et al., 1997] 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)*, to appear 1997.

[Wielinga et al., 1995] B. J. Wielinga, J. M. Akkermans, and A. Th. Schreiber: A Formal Analysis of Parametric Design Problem Solving. In *Proceedings of the 9th Banff Knowledge Acquisition Workshop (KAW-95)*, Banff, Canada, January 26 - Feruary 3, 1995.

[Yost & Rothenfluh, 1996] G. R. Yost and T.R. Rothenfluh: Configuring elevator systems, *International Journal of Human-Computer Studies (IJHCS)*, 44(3/4):521—568, 1996.

[Zdrahal & Motta, 1995] Z. Zdrahal and E. Motta: An In-Depth Analysis of Propose & Revise Problem Solving Methods. In *Proceedings of the 9th Banff Knowledge Acquisition Workshop (KAW-95)*, Banff, Canada, January 26 - Feruary 3, 1995.