In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´98)*, Banff, Kanada, April 1998.

Structured Development of Problem Solving Methods

Dieter Fensel[1] and Enrico Motta[2]

[1] Institute AIFB,
University of Karlsruhe,
D-76128 Karlsruhe, Germany
dieter.fensel@aifb.uni-karlsruhe.de
http://www.aifb.uni-karlsruhe.de/~dfe

[2] Knowledge Media Institute
The Open University
Walton Hall, Milton Keynes, UK
E.Motta@open.ac.uk
http://kmi.open.ac.uk/~enrico

Abstract. *Problem solving methods* (PSMs) are domain-independent reasoning components, which specify patterns of behavior which can be reused across applications. While the availability of extensive PSM libraries and the emerging consensus on PSM specification languages indicate the maturity of the field, a number of important research issues are still open. In particular, very little progress has been achieved on foundational and methodological issues. Existing libraries of PSMs lack a clear theoretical basis and only provide weak support for the method development process, usually in the form of informal guidelines. In this paper we will address these issues by illustrating a framework which characterizes PSMs in terms of *problem commitments*, *problem-solving paradigms* and *domain assumptions*. This framework provides i) a theoretical foundation for situating PSM research and individual PSMs, as well as ii) an organization which allows us to characterize method development and selection as a process of navigating through a three-dimensional space (defined by the three components of our framework). Individual moves through this space are specified by means of *adapters*. In the paper we will illustrate these ideas in detail, with examples taken from parametric design problem solving.

1. Introduction

Problem solving methods (PSMs) describe domain-independent reasoning components, which specify patterns of behavior which can be reused across applications. For instance, *Propose&Revise* (Marcus et al., 1988; Zdrahal and Motta, 1995) provides a generic reasoning pattern, characterized by iterative sequences of model 'extension' and 'revision', which can be reused quite easily to solve scheduling (Stout et al., 1988) and design (Marcus et al., 1988) problems. PSMs provide an important technology for supporting structured development approaches in knowledge engineering: they can be used i) to provide strong, model-based frameworks in which to carry out knowledge acquisition (Marcus, 1988; van Heijst et al., 1992) and ii) to support the rapid development of robust and maintainable applications through component reuse (Runkel et al., 1996; Motta, 1997; Motta and Zdrahal, 1997). More in general, the study of PSMs can be seen as a way to move beyond the notion of knowledge engineering as an 'art' (Feigenbaum, 1977), to formulate a task-oriented systematization of the field, which will make it possible to produce rigorous handbooks similar to those available for other engineering fields. From a philosophical perspective, such a systematization could be used as the source for experimenting with "functional theories of intelligence" (Chandrasekaran, 1987).

Thus, the study of PSMs is important for both practical and theoretical reasons. So far, most of the research effort has focused on identifying and specifying PSMs. As a result, several PSM libraries are now available (Marcus, 1988; Breuker et al., 1987; Benjamins, 1993; Puppe, 1993; O'Hara, 1995; Breuker and van de Velde, 1994; Motta, 1997) and a number of PSM specification languages have been proposed, ranging from informal notations (Benjamins, 1993; Schreiber at al., 1994) to formal modeling languages (Fensel and van Harmelen, 1994). The latter area of research is now well established, to such an extent that two projects, one in Europe, IBROW^3 (Benjamins et al., 1998), and one in the US, High Performance Knowledge Bases (HPKB, 1997), have been set up with the aim (among other ones) of producing standard formalisms for PSM specification.

While the availability of extensive PSM libraries and the emerging consensus on PSM specification languages indicate the maturity and the 'healthy state' of the field, a number of important research issues are still open. In particular, very little progress has been achieved on foundational and methodological issues. Existing libraries of PSMs lack a clear theoretical basis (typically, they are just associations of problem solving components to tasks) and only provide weak support for the method development process, usually in the form of informal guidelines (Benjamins, 1993; O'Hara, 1995). As a result, practitioners have encountered problems when trying to reuse these libraries. An interesting case study is reported by Orsvarn (1996), who discusses the problems he experienced when attempting to reuse Benjamins' library. For example, he found that in some cases not all assumptions associated with a method were made explicit in the method specification. He also found "tacit dependencies" between different parts of the library (more precisely, different branches of the *task-method structure* - see section 2.1). As a consequence of these problems, he had to modify the structure of the library quite extensively, despite the fact that his target application was relatively straightforward.

In our view these difficulties stem from three aspects of published libraries of problem solving methods: they lack a clear theoretical basis, the components are only informally specified and the method refinement operators are not explicitly represented. As a result, i) it is difficult to characterize the coverage of a particular library (i.e. what is the space of problem solving behaviors covered by a library); ii) it is difficult to compare and contrast PSMs associated with different tasks; iii) it is difficult to understand how a PSM was developed (and what alternative specifications are feasible); iv) it is difficult to support automatic method selection and configuration, as envisaged in the IBROW^3 project; and v) it is difficult to verify the properties of a library formally. For instance, it is impossible to check whether or not a library satisfies the requirements of *method correctness* and *method generality* postulated by Orsvarn.

In this paper we will address these issues by illustrating a framework which characterizes PSMs in terms of *problem commitments*, *problem-solving paradigms* and *domain assumptions*. This framework provides i) a theoretical foundation for situating PSM research and individual PSMs, as well as ii) an organization which allows us to characterize method development and selection as a process of navigating through a three-dimensional space (defined by the three components of our framework). Individual moves through this space are formally specified by means of *adapters* (Fensel and Groenboom, 1997; Fensel, 1997). In the rest of the paper we will illustrate these ideas in detail, with examples taken from *parametric design problem solving* (Wielinga et al., 1995; Motta and Zdrahal, 1996).

2. Foundations of Problem solving methods

2.1 Problems with the task-centred perspective on problem solving methods

PSMs are normally described informally as "ways to solve a task" (Benjamins, 1993). This informal 'method-solves-task' association is used as a structuring principle for organizing libraries of PSMs. The resulting organization, which is shown in figure 1, has been used in several approaches (Steels, 1990; Chandrasekaran et al., 1992; Benjamins, 1993; Puerta et al., 1992) and is normally referred to as a *task-method structure*.

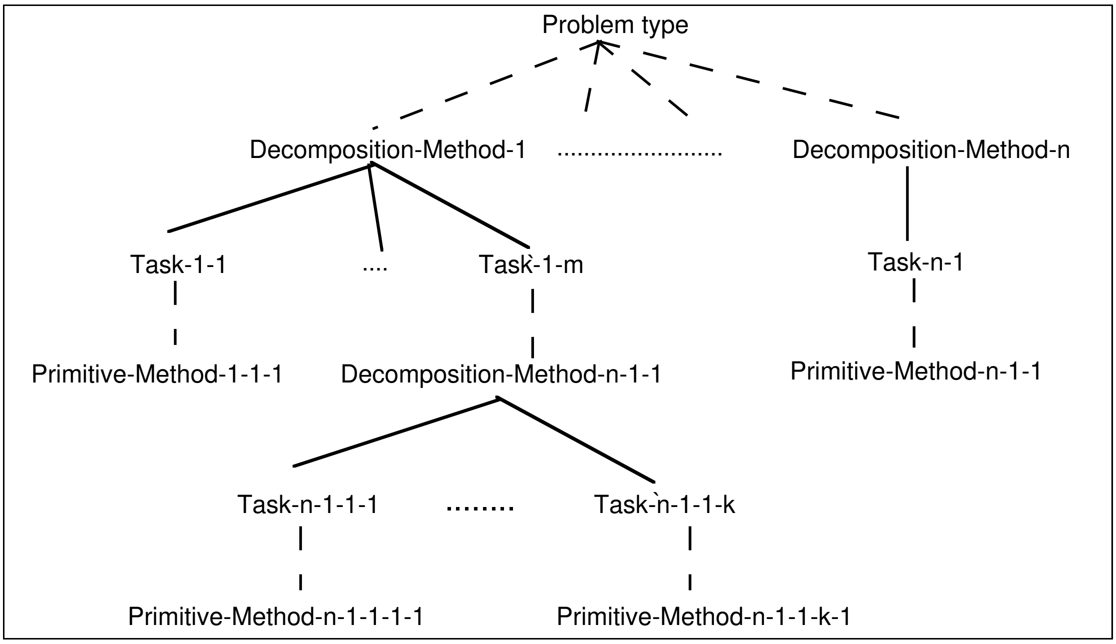

Figure 1. Generic structure of task-method hierarchies.

The root of a task-method structure is given by a high-level task, such as diagnosis or design. These high-level tasks specify *problem types*. Tasks are either solved directly (by means of *primitive methods*), or are decomposed into subtasks (by means of *decomposition methods*). Thus, the resulting libraries are organized in a strong, task-oriented way. This approach permeates most recent research not just on PSMs but on knowledge systems in general and its origins can be found in the reaction to the so-called *weak methods* (Newell and Simon, 1972), which characterized early artificial intelligence research. In a nutshell, researchers in knowledge systems believe that intelligent (i.e. efficient) problem solving is not so much a function of clever algorithms, as a function of the availability of task-specific and domain-specific problem solving knowledge. In the context of PSMs, this approach has been realized by developing *strong methods* (McDermott, 1988), which embed strong commitments to the available domain knowledge.

While it is hard to imagine anybody in the knowledge engineering community objecting to the knowledge-intensive stance characterizing research in this area, a number of researchers have highlighted problems and limitations associated with task-oriented PSM specifications. For instance Beys et al. (1996) have pointed out that task-specific formulations unnecessarily restrict the reuse of PSMs across tasks and have proposed that PSMs be specified in a task-independent style. In (Fensel et al., 1997) we showed that this could be achieved with no loss of functionality and no loss of abstraction (i.e. without resorting to a lower level of description). In particular, we provided a task-independent specification of a Propose&Revise problem solver, which replaces task-related commitments (e.g. valid design model) with task-independent notions, such as correctness and consistency.

Another problem with the use of task-specific conceptualizations is that, in some cases, these can make it difficult to understand what a PSM actually does. For instance, Motta and Zdrahal (1996) discuss the competence of a number of Propose&Revise problem solvers and show that differences in their competence can be accounted for by formulating them as search algorithms and comparing their state-selection strategies. Interestingly, this problem seems to be specular to the one reported by Clancey (1985), which complained about the "blinding effect" of the implementation terminology used in rule-based systems, which made understanding the problem solving competence of these systems much more difficult.

In summary, while task-orientation has traditionally been the defining feature of PSM research, there is growing awareness in the field that this feature is not just unnecessary but can even be counter-productive.

2.2 Problem solving methods as task-specific formulations of search algorithms

The work by Motta and Zdrahal (Motta and Zdrahal, 1996; 1997; Motta, 1997; Zdrahal and Motta, 1995; 1996) attempts to provide a task-independent foundation to PSMs. Their approach, which is based on the *Task/Method/Domain/Application* (TMDA) framework - see figure 2, considers a PSM as a specialization of a task-specific, but PSM-independent, *problem solving model,* which is in turn constructed by instantiating a generic *problem solving paradigm* (e.g. search) in terms of a *task ontology*. Motta and Zdrahal instantiated their approach in the area of parametric design and showed that several PSMs existing in the literature - e.g. *Propose&Backtrack* (Runkel et al., 1996), *Propose&Exchange* (Poeck and Puppe, 1992) and Propose and Revise - could be characterized as specializations of a common problem solving model, obtained by instantiated an ontology for parametric design tasks in terms of a search model of problem solving. In particular, all PSMs in the parametric design library described in (Motta, 1997) subscribe to a common, generic control regime. As discussed in (Motta, 1997; Motta and Zdrahal, 1997), this approach i) makes it possible to plug and play functionally equivalent components with no change to the control structure and ii) facilitates the comparative evaluation of alternative PSMs.

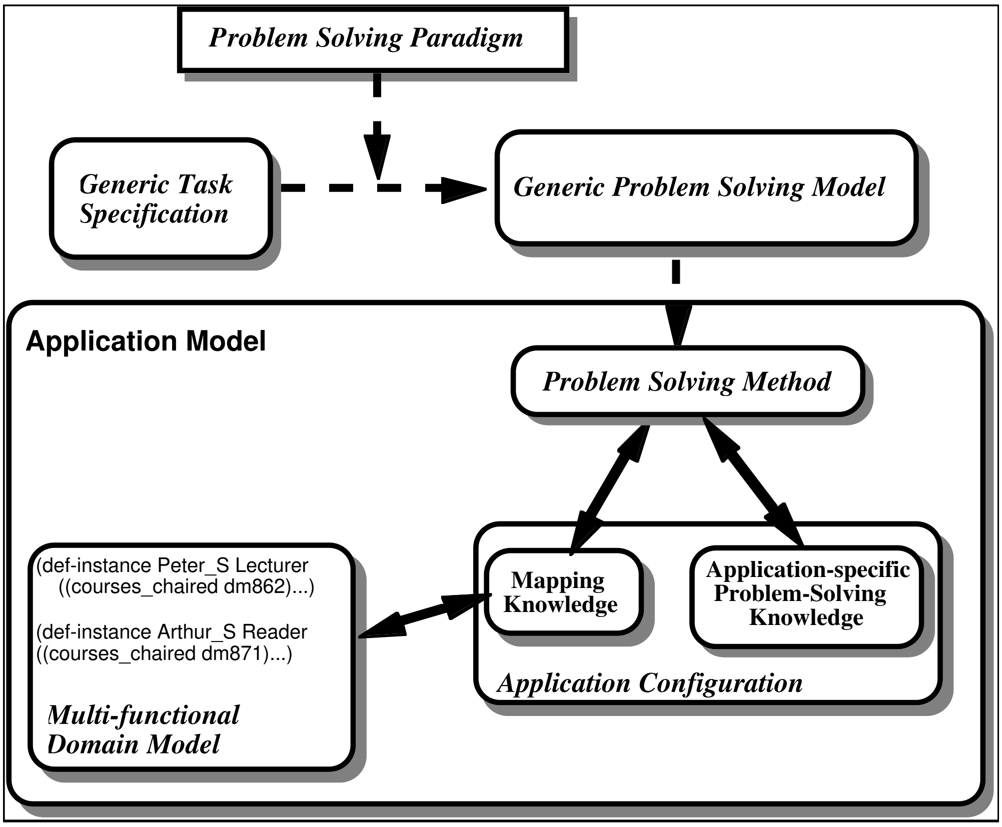

Figure 2. The TMDA modelling framework.

This work is important because it shows that i) it is possible to provide task-independent foundations to PSMs, even within a task-oriented approach, and that ii) the reliance on a common problem solving model facilitates the construction and the analysis of a library of problem solving methods. However the TMDA framework is limited insofar as it only allows the analysis of PSMs tackling the same class of tasks - i.e. while the framework generalizes from a particular class of tasks, it is still task-centred. Another limitation of the approach is that it only characterizes method development informally, as a relatively unstructured specialization process. The framework we propose in this paper, which is called *PPA (Problems, Paradigms, Assumptions),* addresses these problems by generalizing from the TMDA approach and by proposing the notion of *adapter* as a construct for formalizing PSM refinement steps.

Adapters make it possible to reuse and integrate different types of components—e.g. a task-independent problem solving method and a task specification—which have been defined independently of each other. In a simple case, an adapter maps the different terminologies associated with a task definition, problem solving method or domain model. In a more complex case an adapter might introduce additional requirements and assumptions, which are needed—for instance—to relate the competence of a problem-solving method to the functionality required by a task definition.

The explicit characterization of moves through the PSM space is important both to formalize the method development process and also to ensure that not only problem solving components but also the process of creating them becomes part of a library. The importance of recording the knowledge engineering experience in a library has been recognized for a number of years (Stutt and Motta, 1995; van de Velde, 1994). However, approaches to capturing design expertise in knowledge engineering have mostly centered on recording informal guidelines. In contrast with these approaches, adapters provide a formal way of capturing the method development process, thus providing the basis for automatic method configuration.

The PPA framework is discussed in the next section.

3. A three-dimensional characterization of the space of problem solving methods

A problem-solving method in the PPA framework consists of three ingredients (see figure 3).

- **Problem-solving paradigm**. This is a high-level description which specifies a type of problem solving rather than an actual algorithm. A problem-solving paradigm fixes some basic data structures, provides an initial task-subtask decomposition and (optionally) a generic control regime. As in the TMDA framework, this generic control regime is meant to be common to all PSMs which subscribe to the same problem solving paradigm. Examples of problem solving paradigms are: generate & test, local search and problem reduction (Smith & Lowry, 1990). An important hypothesis in AI is that all intelligent problem solving can be characterized as a search process (Newell and Simon, 1976). If this hypothesis is adhered to, then PSMs can be characterized as specializations of the search paradigm.
- **Problem commitments**. These specify ontological commitments to the type of problem that is solved by the problem-solving method. These commitments are expressed by subscribing to a particular *task ontology*. A task ontology specifies a task in terms of initial and goal states in the universe of discourse and introduces the terminology necessary to express task-specific commitments. The ontological commitments introduced by a task (in particular, by a problem type) can be used to refine the competence of a PSM, the structure of its *computational state* and the nature of the *state transitions* it can execute. For instance, a generic search method can thus be transformed into a more specific method for model-based diagnosis or parametric design. A task-specific refinement of a method is still reusable because it is formulated independently of any domain. That is, the method may be applicable to technical or medical diagnostic problems. However, it is limited to a specific class of tasks. The advantage of refining PSMs in a task-specific way is that the resulting model provides much stronger support for knowledge acquisition and application development than a task-independent one.
- **Domain Assumptions**. These are assumptions on the domain knowledge that is required to instantiate a PSM in a particular application. These assumptions specify the types and the properties of the knowledge structures which need to be provided by a domain model, in addition to those required to instantiate a task ontology. For instance, when solving a design problem by means of Propose&Revise, a domain needs to provide the knowledge required to express *procedures* and *fixes*, in addition to the knowledge needed to formulate the specific design problem - e.g. parts and constraints. Domain assumptions are necessary to enable efficient problem solving for complex and intractable problems. The reliance on such domain-specific knowledge

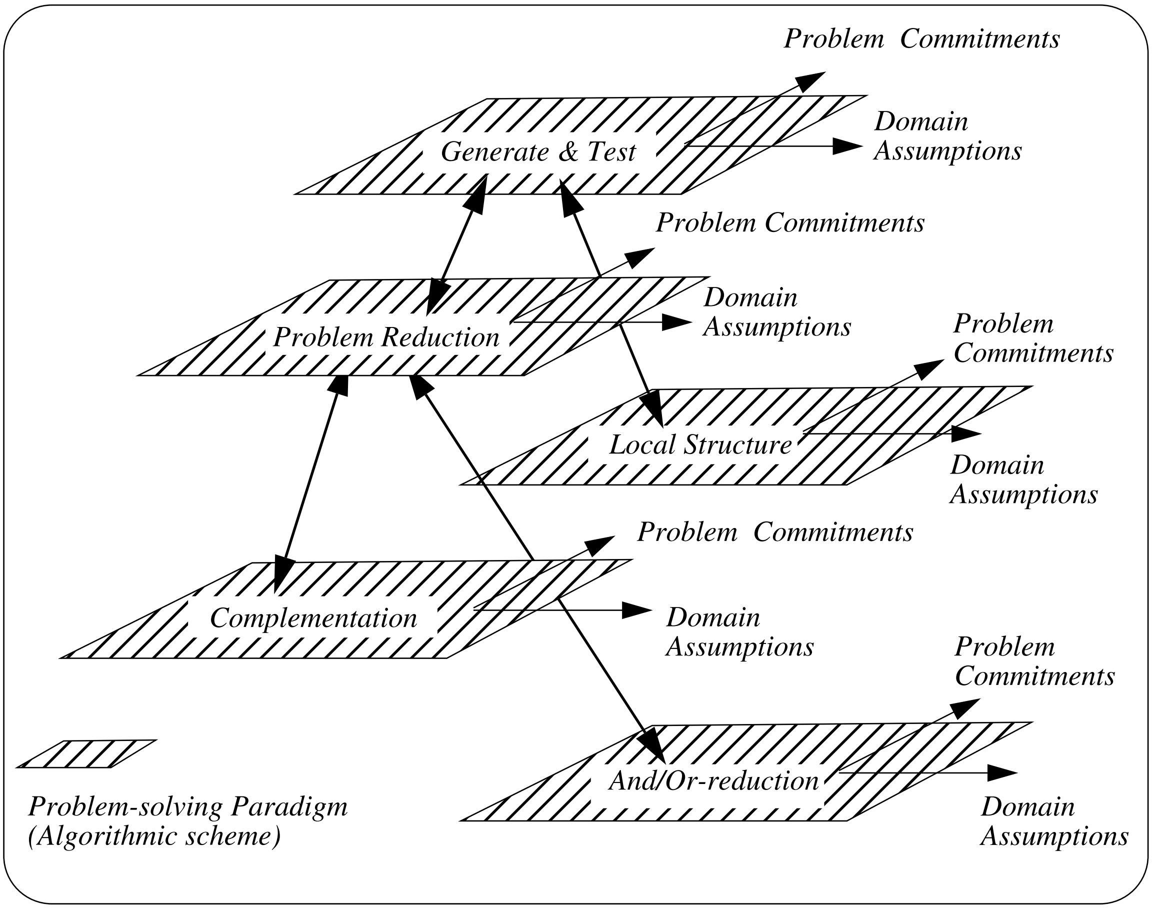

Figure 3. The navigation space for developing problem-solving methods.

is the defining feature of knowledge-intensive approaches to problem solving.

While these three components cannot be said to specify truly orthogonal dimensions (the choice of a problem solving paradigm may impose constraints on the domain knowledge), they effectively provide alternative degrees of freedom for method specification and refinement (although some points in the three-dimensional space may not be reachable). For instance, PSMs can be specified in a task-independent way - i.e. with no problem commitments; they can be specified with no domain assumptions - i.e. by instantiating a problem solving paradigm with respect to a task ontology, without further commitments; and they can be specified without introducing any control regime or task-subtask decomposition. Such a specification won't necessarily be operational, but it would be good enough to support a PSM *broker*, such as the one envisaged in the IBROW[3] project (Benjamins et al., 1998). The feasibility of this kind of PSM specifications was shown in an earlier paper of ours (Fensel et al., 1997), where we illustrated a declarative specification of a Propose&Revise problem solver, which abstracts from all procedural and control aspects associated with a PSM. Incidentally, this specification was also task-independent, thus showing that it is possible to characterize PSMs purely in terms of domain assumptions, with no commitments to a task and only generic commitments to a problem solving paradigm.

Clearly identifying and separating the problem-solving paradigm, problem commitments, and domain assumptions enables a principled way to developing a problem-solving method and structuring libraries of problem-solving methods. Current approaches to problem-solving methods usually merge these different aspects, thus limiting the possibilities for reuse, obscuring the nature of the methods, and making it difficult to reconstruct the process which led to a particular PSM specification. In the PPA framework, the development and adaptation of problem-solving methods is characterized as a navigation process in this three-dimensional space. The nature of the navigation process itself differs with respect to the dimensions of the development process. Moves through the dimension of problem solving paradigms are not necessarily structure-preserving. In fact, changing the problem-solving paradigm is a revolutionary act that creates new structures in the problem-solving process. In contrast with paradigm-shifting activities, moves which refine the problem type or the domain assumptions are structure-preserving.

Moves through the three-dimensional space are represented by means of *adapters*. In some cases these are needed to map the different terminologies associated with task, method and domain specifications. For instance, figure 4 shows an adapter which specializes a task-independent specification of a Propose&Revise problem solver for a parametric design task.

In more complex cases, adapters can introduce further commitments needed to relate the competence of a problem-solving method with the goal associated with a task specification. For example, let's consider the case in which we specialize a task-independent Propose&Revise for an optimal parametric design task (see figure 5, This example is described in detail in (Fensel et al., 1997)). In general, optimality can only be guaranteed by introducing strong assumptions on the available propose and fix knowledge. Thus, the resulting adapter not only maps method-specific to task-specific terminology, but also introduces additional commitments required by the optimality criterion. For instance, it introduces the requirement that propose and revise steps must be optimal.

Adapters can also be used to refine task or method specifications. These cases are illustrated in figures 6-8. Figure 6 shows the definition of a generic design task - see section 4.1 for more details on this problem type - and figure 7 shows an adapter specializing the definition of the generic design task for parametric design. Finally, figure 8 shows an adapter which refines a generic hill climbing method to produce a set minimizer.

The adapters described here are similar to the *adapter patterns* discussed by (Gamma et al., 1995), where adapters are given as patterns which make it possible to reuse object classes.

adapter *propose & revise for parametric design tasks*
    import *propose&revise; paramteric design*
    export *propose&revise for parametric design tasks*
    rename
        state → *DesignModel,*
    axioms
        $goal(output)$
        $d < d \leftrightarrow cost(d) < cost(d')$
        $Partial\ completeness(d) := \{p \mid p \in assigned(d)\}$
endadapter

Figure 4. Specializing Propose&Revise for parametric design tasks.

adapter *propose & revise for optimal parametric design tasks*
import *propose&revise for paramteric design*
export *propose & revise for optimal parametric design tasks*
axioms
/* The output is a complete, correct and optimal design model. */
$complete(output) \wedge correct(output) \wedge$
$\neg\exists\, d\,.\,(d \in ParametricDesignModel \wedge complete(d) \wedge correct(d) \wedge output \prec d)$
/* The propose knowledge never fails and monotonically extends the design model. */
$\neg\, complete(d) \rightarrow Partial\ completeness(d) < Partial\ completeness(propose(d))$
/* The application of a propose leads to an optimal design model. */
$\neg\, complete(d) \rightarrow$
$\neg\exists\, d'\,(d' \in ParametricDesignModel \wedge correct(d') \wedge propose(d) \prec d' \wedge$
$Partial\ completeness(d') = Partial\ completeness(propose(d))$
/* The revise knowledge never fails. */
$\neg\, correct(d) \rightarrow correct(revise(d))$
/* The application of revise does not change the completeness of a design model. */
$Partial\ completeness(revise(d)) = Partial\ completeness(d)$
/* The application of revise leads to an optimal design model. */
$\neg\, correct(d) \rightarrow$
$\neg\, \exists\, d'\,(d' \in ParametricDesignModel \wedge correct(d') \wedge$
$Partial\ completeness(revise(d)) = Partial\ completeness(d') \wedge$
$revise(d) \prec d'\,)$
endadapter

Figure 5. Specializing Propose&Revise for optimal parametric design tasks.

While this functionality is provided also by our notion of adapter, our characterization also makes it possible to specify explicitly the commitments and assumptions which are necessary to 'bridge the gap' between a problem definition (task) and the competence of a problem-solving method.

Natural candidates for the formal definition of adapters are algebraic specifications. These have been developed in software engineering to define the functionality of a software artifact (Bidoit et al., 1991; Wirsing, 1990). Briefly, algebraic specifications comprise a *signature* (consisting of types, constants, functions, and predicates) and a set of axioms that define the properties of these syntactical elements.

In the next section we will illustrate these ideas by showing how the PPA approach can be used to carry out a rationale reconstruction of part of the library for parametric design described in (Motta, 1997), henceforth, the 'Motta library'. The resulting structure teases out the various types of task-specific and domain-specific commitments associated with different problem solving components and makes explicit the model development process by means of the relevant adapters.

task *Design*
sorts
DesignModel, *Constraint*, *Requirement*, *Cost*,
Constraints : **set of** Constraint, *Requirements* : **set of** *Requirement*;
functions
violated : *DesignModel* → *Constraints*;
fulfilled : *DesignModel* → *Requirements*;
cost : *DesignModel* → *Cost*;
predicates
consistent, *optimal*, *suitable*, *valid* , *solution*, *goal* : *DesignModel*;
axioms
$goal(x) \leftrightarrow solution(x)$
$solution(x) \leftrightarrow valid(x) \leftrightarrow consistent(x) \wedge suitable(x)$
$consistent(x) \leftrightarrow violated(x) = \emptyset$
$suitable(x) \leftrightarrow \forall\, y\; (y \in fulfilled(x))$
$optimal(x) \leftrightarrow \neg\, \exists\, y\; (cost(y) < cost(x))$
endtask

Figure 6. Specification of problem type design.

adapter *Problem Type Parametric Design*
import
Problem Type Design;
export
Problem Type Parametric Design;
terminology mapping
DesignModel → *ParametricDesignModel*;
sorts
$Parameter = \{p_1,...,p_n\}$,
Parameters : **set of** *Parameter*,
$ValueRange_1, ..., ValueRange_n$;
functions
$ParametricDesignModel : Parameter \rightarrow ValueRange_1 \cup ... \cup ValueRange_n$
where $ParametricDesignModel(p_i) \in ValueRange_i$ for all i=1,..,n;
assigned : *ParametricDesignModel* → *Parameters*;
predicates
complete : *ParametricDesignModel*;
axioms
$solution_{Export}(x) \leftrightarrow solution_{Import}(x) \wedge complete(x)$;
$complete(x) \leftrightarrow \forall\, y\; (y \in assigned(x))$;
endadapter

Figure 7. Parametric design as a specialization of problem type design.

```
adapter set mimimizer
    import hill climbing
    export set mimimizer
    axioms
    /* The input set must be correct. */
        correct(input)
    /* select1 must select the input set. */
        select1(x) = {x}
    /* Successors are subsets that contain one less element .*/
        successor(x,y) ↔ ∃z . (z ∈ x ∧ y = x \ {z})
    /* We prefer smaller sets if they are still correct. */
        x < y ↔ correct(y) ∧ y ⊂ x
endadapter
```

Figure 8. Set minimization by adapting hill-climbing.

4. A case study: problem solving components for parametric design

As indicated earlier, the library of problem solving components for parametric design developed by Motta (1997) characterizes each problem solving method as a specialization of a generic problem solving model obtained by instantiating a generic search model of problem solving by means of a parametric design task ontology. Here we show how the various steps in the process can be made explicit by means of the relevant adapters. The overall picture of refining the problem specificity of a generic search method is provided in Figure 9. On the left side we refine the problem definition and on the right side we refine the problem-solving paradigm that guides the problem-solving process. The *virtual elements* do not require a specification because these specifications follow from a combination of an existing specification and an adapter. However, for convenience, a library may also directly provide these derived specifications.

4.1 Defining the problem type

Design can be characterized in generic terms as the process of constructing artifacts. Usually, the target artifact should fulfil a number of requirements, should not violate the relevant constraints and should follow the principle of economy - i.e. it should have minimal cost (Motta and Zdrahal, 1996). Figure 6 shows a possible characterization of the class of design problems. This definition can be specialized for parametric design problems by introducing parameters and value ranges, as shown in figure 7. Parametric design problems reduce the complexity of the design task by assuming the existence of a *parametrized solution template* for the target artifact. Hence, as shown in figure 7, a design model can now be defined as a partial function from parameters to value ranges and a solution can be defined as a *valid* and *complete* design model. This is a model in which all parameters are bound, all requirements are satisfied and no constraint is violated.

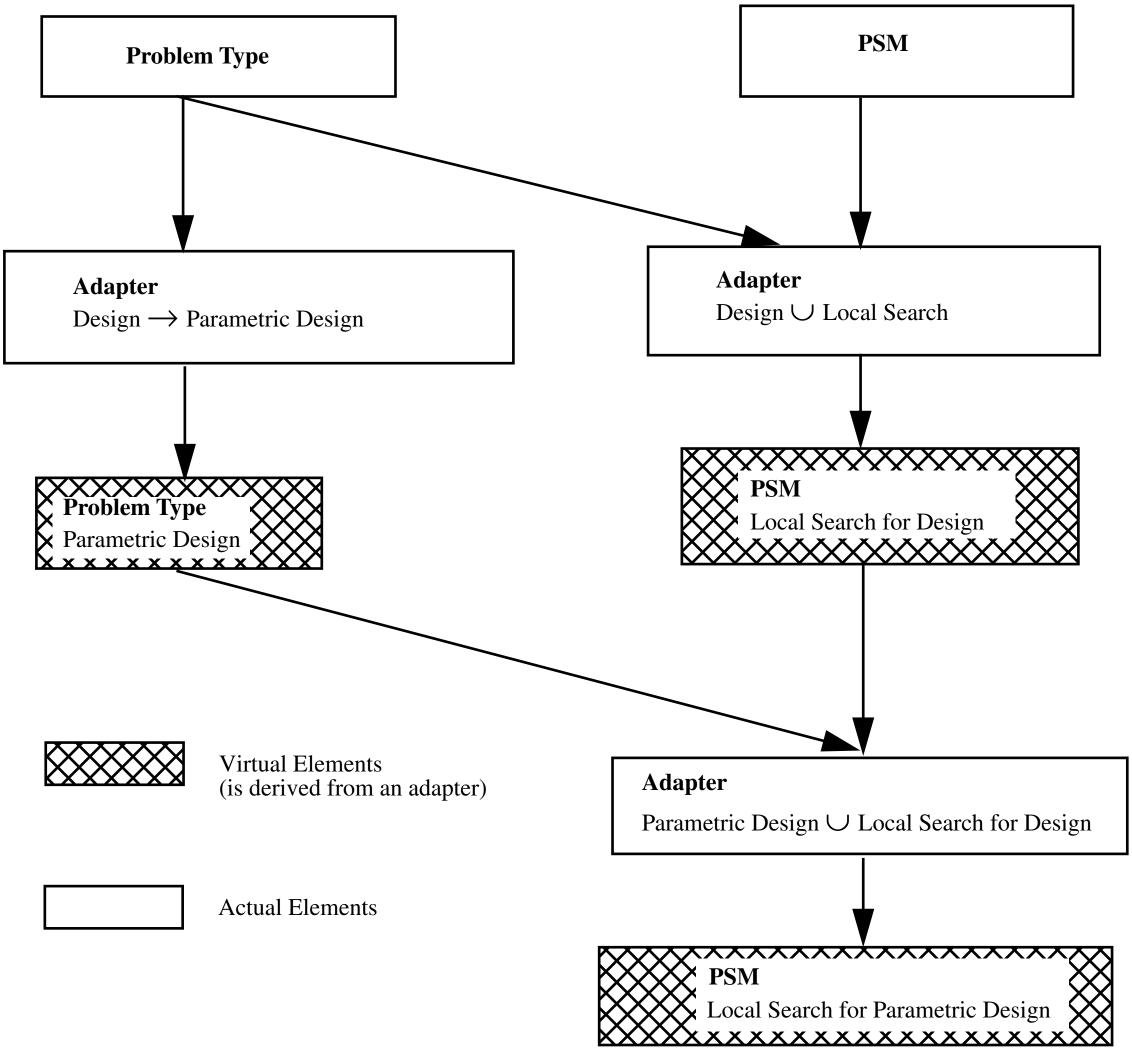

Figure 9. The Development Graph of a PSM for Parametric Design.

4.2 Initial PSM specification

We start with the algorithmic scheme *local search* that defines the common pattern of all problem-solving methods described in (Motta, 1997). A generalized, task-independent version of this control regime is shown in figure 10. Basically, *LocalSearch* initializes the search process and *RecursiveSearch* searches until a solution has been found. At each cycle a node is selected and its successors derived. *UpdateNodes* takes as input the current search space and the newly generated nodes and produce a new search space. Different search strategies can be modeled by defining alternative methods for this task. For instance, *best-first search* can be modeled by performing a simple union of *Nodes* and *SuccessorNodes*, while *hill-climbing* can be modeled by ensuring that *UpdateNodes* returns only *SuccessorNodes*.

The local search model can be specialized for design tasks by means of the adapter shown in figure 11. Essentially, this adapter introduces the term *design model* into the method and maps the notion of task solution to that of method solution. Refining the resulting model of

PSM Local Search

/* There are some simplifications to report: We ignore the case when *SelectNode* does not deliver an output and when *DeriveSuccessorNodes* does not deliver an output. In consequence we only terminate when we have found a solution and we do not terminate for intermediate empty inputs. */

control flow

LocalSearch()
    /* Initializes and starts the search. */
    Nodes:= *Initialize*();
    Output := *RecursiveSearch*(*Nodes*)

RecursiveSearch(*Nodes*)
    /* Searches. */
    Node := *SelectNode*(Nodes);
    SuccessorNodes := *DeriveSuccessorNodes*(*Node*);
    Nodes := *UpdateNodes*(*Nodes*, *SuccessorNodes*);
    IF *OptimalSolution*(*x*) for a $x \in$ *Nodes*
        THEN RETURN these *x*´s
        ELSE *RecursiveSearch*(*Nodes*)
    ENDIF

sorts
    Object, *Objects* : **set of** *Object*;
functions
    Node : *Object*;
    Nodes, *Output*, *SuccessorNodes* : *Objects*;
prediactes
    OptimalSolution : *Object*;

elementary inferences
    DeriveSuccessorNodes,
    Initialize,
    SelectNode,
    UpdateNodes

Figure 10. The *local search* problem solving method.

design problem solving for parametric design is simply a matter of importing the problem type *Parametric Design* and the PSM *Local Search for Design*, to produce the PSM *Local Search for Parametric Design*.

The resulting PSM is still unspecified, as it does not say how to select design models, how to derive successor designs, and how to update the set of designs. These aspects will be discussed in the next section.

4.3 Refining the initial PSM

The PSM defined in the previous section is parametrized in terms of the following four

adapter *Problem-Solving Local Seach forDesign;*
    import
        Problem Type Design, Problem Solving Local Seach;
    export
        Problem-Solving Local Seach forDesign;
    rename
        Object → *DesignModel*;
    axioms
        goal(*Output*)
endadapter

Figure 11. Specializing local search for design.

subtasks.

- *Initialize*. This defines the initial search space.
- *SelectNode*. This selects the node which is going to be expanded.
- *DeriveSuccessorNodes*. This task derives the successors of the current node.
- *UpdateNodes*. This task updates the current search space to take into account the newly generated nodes.

In this section we will discuss these subtasks, in particular showing how different PSMs can be defined by introducing task- or domain-specific commitments, or simply by selecting alternative control regimes. As the starting point for our discussion we will use the model of parametric design problem solving obtained by instantiating the local search method for parametric design tasks.

4.3.1. Task Initialize

Initializing the design space requires not just generating a number of design models, but also evaluating them, so that task *SelectNode* can be carried out. For instance, a design model in the Motta library is evaluated in terms of four task-specific criteria: feasibility, completeness, consistency and cost. These criteria are defined in a task-oriented style, although, as discussed in (Fensel et al., 1997), it is also possible to characterize these notions in a task-independent fashion.

4.3.2. Task DeriveSuccessorNodes

DeriveSuccessorNodes in (Motta, 1997) uses a rather complex three step procedure instead of directly using a successor relationship. First, a *design context* is abstracted from the node, then a *design focus* is derived within the context, and finally a *transformation operator* is derived from the design focus. The application of the transformation operator provides the successor relationship between nodes. The procedure that refines *DeriveSuccessorNodes* is shown in figure 12. Notice that we define this refinement not declaratively via an adapter but via a decomposition in more elementary steps and a procedural order on these steps. That is, we apply the common pattern of refining a task by a problem-solving method. An adapter would refine *DeriveSuccessorNodes* by introducing additional axioms that specialize the

PSM *DeriveSuccessorNodes*

control flow

DeriveSuccessorNodes(*X*)
Context := *DecideOverContext*(*X*);
Focus := *DecideOverFocus*(*X*,*Context*);
Transformation := *DecideOverTransformation*(*X*,*Context*,*Focus*);
Bookmark *Context*, *Focus*, and *Transformation* in *History* of *X*;
X := *ApplyTransformation*(*X*,*Transformation*);
RETURN *X*

sorts

Context-sort, Focus-sort, Transformation-sort;
History-sort : **set of** (*Context* x *Focus* x *Transformation*);

functions

Context : *Context-sort*; *Focus* : *Focus-sort*;
Transformation : *Transformation-sort*; *History* : *History-sort*;

elementary inferences

DecideOverContext,
DecideOverFocus,
DecideOverTransformation,
Bookmark,
ApplyTransformation

Figure 12. The PSM *DeriveSuccessorNodes*.

relations that define the competence of *DeriveSuccessorNodes*.

The use of the three-step selection process is motivated by the structure of the problem type:

- The quality information of a design model distinguishes four different criteria: violation of constraints, fulfillment of requirements, completeness of value assignments and costs. Four different contexts can be immediately identified according to these four criteria: trying to repair constraint violations, trying to improve fulfillment, trying to improve completeness and trying to reduce costs. As the reader may have realized, a more pedantic organization would have already introduced three of these context decisions for design problems, given that these are not specific to parametric design. For reasons of simplicity we have skipped this intermediate step.
- Within a context, we can decide about the focus of the design activity by using the object information, in this case the structure of the parametric design model. A parametric design model is a set of parameters and a transformation can only refer to the value of a parameter. Selecting a focus is therefore about deciding what parameter or set of parameters should be updated next.
- Finally, we select a transformation operator. In general, several transformations may be applicable for a given context and focus.

The method shown in figure 12 provides the default way of carrying out task *DeriveSuccessorNodes* in the Motta library. The reason for this choice is that this method employs minimal commitments. In section 4.4 we will briefly illustrate how this method can be refined to specific PSMs, such as Propose&Revise, which add commitments to the basic control regime employed by *DeriveSuccessorNodes*.

4.3.3. Task SelectNode

At any stage of a problem solving process, a reasoning system (be it human or artificial) knows about a number of design states which are relevant to the problem in hand. Typically, these are the states which have been explored during the current design process, i.e. the states included in the portion of the design space searched so far. However, human designers are of course capable of reusing past designs and the same applies to problem solvers which make use of case-based reasoning techniques when solving design applications (Zdrahal & Motta, 1996). Thus, the current design space includes in general all the states known to the problem solver, either because they have been explored during the current design process, or because the problem solver has access to other relevant design knowledge (e.g. a case-based library of design states).

Task *SelectNode* combines both control and problem solving aspects. It selects a node according to a number of criteria, which include an estimation of its feasibility (i.e., whether the design model may lead to a solution). Incidentally, methods like simulated annealing do not always select the best local node, in order to escape local optima in the search space.

The default state selection method provided by the Motta library chooses a design model which does not violate any constraint, has maximal extension and minimizes the cost. This selection criterion is the one typically used by problem solvers which are not concerned with cost issues (as for instance most constraint satisfaction engines) and which deal with inconsistencies by backtracking to an earlier state. These include both methods which make use of simple control regimes, such as depth-first search with chronological backtracking (Runkel et al., 1996), as well as more sophisticated regimes based on techniques such as *backjumping* (Gaschnig, 1978; Dechter, 1988). The differences between these methods - e.g. backjumping vs. depth-first search with chronological backtracking - are therefore explained in terms of different notions of feasibility, rather than in terms of different state selection policies. Clever backtracking techniques, such as backjumping, can propagate infeasibility 'backwards', to nodes which precede an inconsistent state. Hence, even though different methods may use the same state selection policy, they can still exhibit different search behaviors. *That is, different search strategies are described declaratively by specifying the search guidance as a parameter of a generic search pattern rather than by providing different procedural control structures for each search variant.*

4.3.4. Task UpdateNodes

UpdateNodes significantly influences the type of search that is performed. If the output is obtained by merging the *SuccessorNodes* with the original input *Nodes*, a variant of best-first search is performed, where all expanded nodes are available for selecting the next node. If only *SuccessorNodes* are returned, then the search is restricted to the environment of the selected node. In this case a variant of *hill-climbing* is performed.

4.4 Capturing a family of PSM specifications

A summary of parametric design problem solving in terms of its states and state transitions is provided in Figure 13. This style of specification results from the approach adopted here, which consists of choosing an algorithmic scheme and applying several adapters that refine its state descriptions and state transitions. Such a specification is more abstract than what is usually called a PSM in the literature. For instance, let's consider Propose&Revise. Propose&Revise for parametric design instantiates the framework by distinguishing two *contexts* when deriving successor design models:

- *Propose* context. Consistent design models are extended until they are complete or inconsistent.
- *Revise* context. Inconsistent design models are repaired until they are consistent.

That is, this PSM can be derived from our framework by simply refining some of its parameters. An extensive discussion of how various PSMs for parametric design can be derived as specializations of a generic search model is provided in (Motta, 1997).

Figure 13 also includes a slot for specifying the competence of a PSM. This describes its functionality (and utility), abstracting from how it is achieved (Akkermans et al., 1993; Fensel and Groenboom, 1997; ten Teije, 1997). Such competence descriptions enable black-box reuse. The general problem of establishing competence descriptions of problem-solving methods stems from the fact that they can only be established in terms of assumptions about the required knowledge. The termination, completeness and correctness of generic search

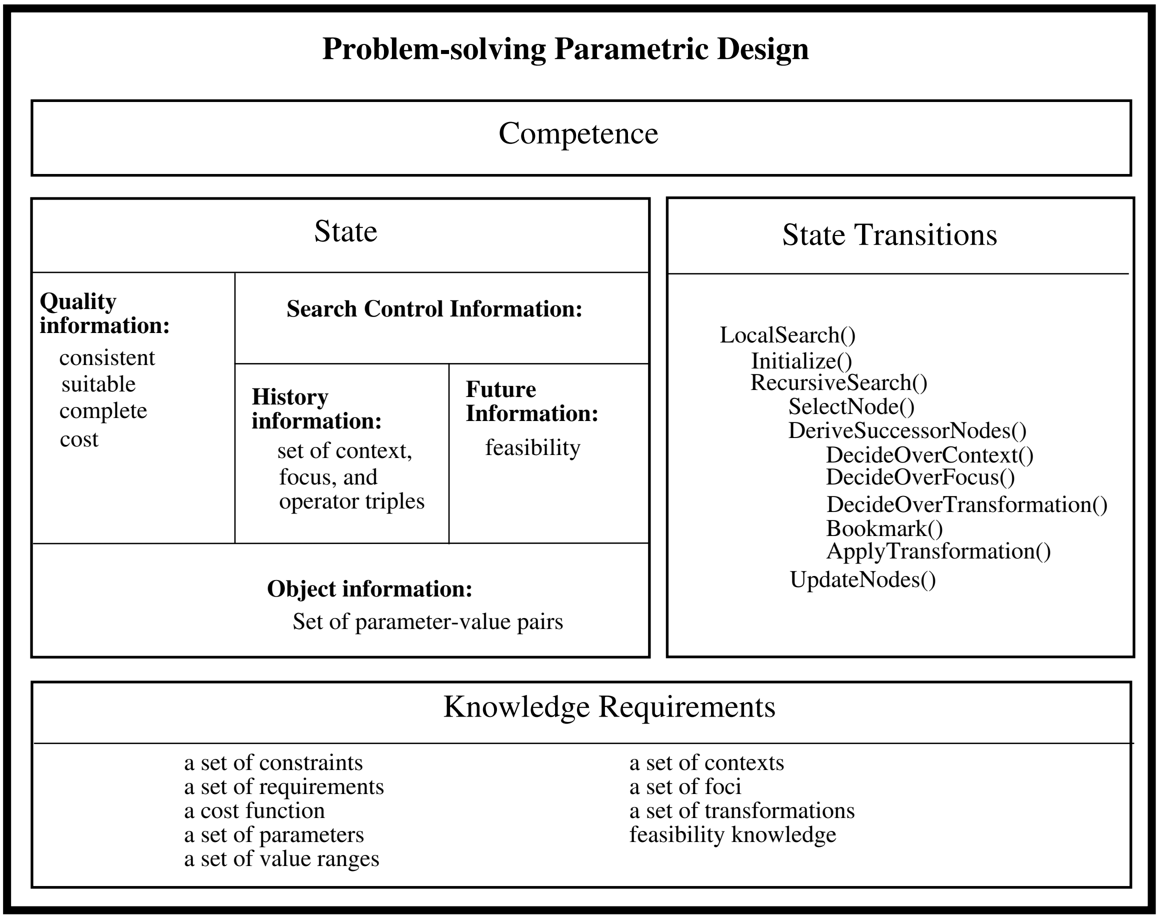

Figure 13. An abstract specification of parametric design problem solving.

methods like A* (or specialized ones, such as Propose&Revise) can only be guaranteed by placing strong requirements on the knowledge that is used to guide the search process. As a consequence, the association of a competence specification to a PSM is not a bijective relation. On the contrary, it depends on the assumptions associated with the PSM (Benjamins et al., 1996).

5. Conclusions

In this paper we have presented a systematic approach enabling the structured development, adaptation, and reuse of PSMs. A large number of PSMs can be described by means of i) a structured set of generic task and problem-solving patterns and ii) adapters formalizing the relevant refinement steps. These are explicitly modeled, thus allowing their reuse for new problem types and different problem-solving schemes.

Adapters introduce a new modeling element into existing modeling approaches, which makes the method development process explicit and external to the model, i.e. separated from problem solving components. In most other approaches, adaptation is treated as a side issue and adaptation via several levels is not considered at all. As discussed in section 1, this is especially a problem for libraries of PSMs. In our view, by providing a clear foundation for both PSM specification and specialization we can construct sound and reusable libraries. While this is only an unproved claim as far as reuse is concerned, the experience reported by Motta and Zdrahal (Motta, 1997; Motta and Zdrahal, 1997) shows that the reliance on clear theoretical foundations allows a better understanding of PSMs and facilitates their evaluation and comparison.

Still, our current adapter concept may be too general and we are currently working on a typology of adapters. So far we have identified two basic dimensions for organizing adapters: in terms of their purpose (*teleological dimension*) and in terms of the syntactical entity refined by an adapter (*epistemological dimension*). The first dimension indicates whether an adapter is used to refine a problem type, a domain dependency via assumptions, or a problem-solving paradigm. The second dimension distinguishes whether an adapter refines, for instance, a state definition, a state transition, or a declarative competence specification. Working this out in more detail will help transforming the development process of PSMs into a more structured engineering activity.

References

Akkermans, J. M., Wielinga, B. J. and Schreiber, A. T. (1993). Steps in constructing problem-solving methods. In N. Aussenac, G. Boy, M. Linster, J-G Ganascia and Y. Kodratoff (Editors). *Knowledge Acquisition for Knowledge-Based Systems - EKAW '93*. Lecture Notes in Artificial Intelligence, LNCS 723, Springer-Verlag.

Balkany A., Birmingham W.P. and Runkel J. (1994). Solving Sisyphus by Design. International Journal of Human-Computer Studies, 40(2). pp. 221-241.

Benjamins, R. (1993). *Problem Solving Methods for Diagnosis*. PhD Thesis, Department of Social Science Informatics, University of Amsterdam.

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

Benjamins, R., Plaza, E., Motta, E., Fensel, D., Studer, R., Wielinga, B.. Schreiber, G., Zdrahal, Z. (1998): An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide-Web. In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´98),* Banff, Canada, April 18-23.

Beys, P., Benjamins, R. and van Heijst, G. (1996). Remedying the Reusability-Usability Tradeoff for Problem-solving Methods. In B. Gaines and M. Musen (Editors), *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*. Banff, Alberta, Canada.

Bidoit, M., Kreowski, H.-J. , Lescane, P., Orejas, F. and Sannella, D. (eds.) (1991). *Algebraic System Specification and Development*, Lecture Notes in Computer Science (LNCS) 501, Springer-Verlag.

Breuker, J. A. and Van de Velde, W. (1994). *CommonKADS Library for Expertise Modelling*. IOS Press, Amsterdam, The Netherlands.

Breuker J., Wielinga B.J., van Someren M., de Hoog R., Schreiber G., de Greef P., Bredeweg B., Wielemaker J., Billault J.-P., Davoodi M. and Hayward S. (1987). *Model Driven Knowledge Acquisition: Interpretation Models*. Deliverable D1, Esprit Project P1098, KADS. University of Amsterdam.

Chandrasekaran, B. (1987). Towards a functional architecture for intelligence based on generic information processing tasks. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI '87)*, pp. 1183-1192. Morgan Kaufmann.

Chandrasekaran, B., Johnson, T.R. and Smith, J.W. (1992). Task-Structure Analysis for Knowledge Modelling. *Communications of the ACM 35(9)*, pp. 124-137.

Clancey W. J. (1985). Heuristic Classification. *Artificial Intelligence, 27*, pp. 289-350.

Dechter, R. (1988). Constraint Processing Incorporating Backjumping, Learning and Cutset-Decomposition. Proceedings of the 4th Conference on Artificial Intelligence Applications - CAIA '88, pp. 312-319.

Feigenbaum, E. A. (1977). The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA.

Fensel, D. (1997). The Tower-of-Adapter Method for Developing and Reusing Problem-Solving Methods. In R. Benjamins and E. Plaza (Editors). *Knowledge Acquisition, Modeling, and Management. Proceedings of the 10th European Workshop, EKAW ‘97.* Lecture Notes in Artificial Intelligence 1319, Springer-Verlag.

Fensel, D. and Groenboom, R. (1997). 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.

Fensel, D., Motta, E., Decker, S., and Zdrahal, Z. (1997). The use of Ontologies for Specifying Tasks and Problem Solving Methods: A Case Study. In R. Benjamins and E. Plaza (Editors). *Knowledge Acquisition, Modeling, and Management. Proceedings of the 10th European Workshop, EKAW ‘97.* Lecture Notes in Artificial Intelligence 1319, Springer-Verlag.

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

Gamma, E., Helm, R. Johnson, R., and Vlissides, J. (1995). *Design Patterns*, Addison-Wesley Pub.

Gaschnig, J. (1978). Experimental case studies of Backtrack vs. Waltz-type vs. new algorithms for satisficing assignment problems. Proceedings of the 2nd Biennal Conference of the Canadian Society for Computational Studies of Intelligence. Toronto, July 1978.

HPKB (1997). High Performance Knowledge Bases. Darpa Project. Details available from http://www.teknowledge.com:80/HPKB/.

Marcus S. (Editor) (1988). *Automatic Knowledge Acquisition for Expert Systems.* Kluwer Academic, Boston, MA.

Marcus, S, Stout, J., and McDermott, J. (1988). VT: An Expert Elevator Designer that uses Knowledge-Based Backtracking. *AI Magazine*, 9(1), pp. 95-112.

McDermott, J. (1988). Preliminary steps toward a taxonomy of problem-solving methods. In S. Marcus

(Editor), *Automating Knowledge Acquisition for Expert Systems*, Kluwer Academic.

Motta E. (1997) Reusable Components for Knowledge Models. *PhD Thesis*. Knowledge Media Institute. The Open University. UK

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

Motta, E. and Zdrahal, Z. (1997). An approach to the organization of a library of problem solving methods which integrates the search paradigm with task and method ontologies. To appear in the *International Journal of Human-Computer Studies*. Available from http://kmi.open.ac.uk/~enrico/papers/ijhcs_psm.ps.gz.

Newell, A. and Simon, H. A. (1972). *Human Problem Solving*. Prentice Hall, Englewood, NJ.

Newell, A. and Simon, H. A. (1976). Computer Science as Empirical Enquiry: Symbols and Search. *Communications of the ACM*, *19(3)*, pp. 113-126, March.

O'Hara, K. (1995). The GDM Grammar, v.4.6. *VITAL Project Report NOTT/T252.3.3*. Available from the author at AI Group, Department of Psychology, University of Nottingham, UK.

Orsvärn, K. (1996). Principles for Libraries of Task Decomposition Methods - Conclusions from a Case-study. In N. Shadbolt, K. O'Hara, and G. Schreiber (Editors). *Advances in Knowledge Acquisition - EKAW '96*. Lecture Notes in Artificial Intelligence, 1076. Springer-Verlag, pp. 48-65.

Poeck, K. and Puppe, F. (1992). COKE: Efficient Solving of Complex Assignment Problems with the Propose-and-Exchange Method. *5th International Conference on Tools with Artificial Intelligence*. Arlington, Virginia.

Puerta, A. R., Egar, J. W., Tu, S. W., and Musen, M. A. (1992). A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools. *Knowledge Acquisition, 4(2)*, pp. 171-196.

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

Runkel, J. T., Birmingham, W. B. and Balkany, A. (1996). Solving VT by Reuse. *International Journal of Human-Computer Studies 44 (3/4)*, pp. 403-433.

Schreiber, A.Th., Wielinga B. J., Akkermans, H., van de Velde, W., and Anjewierden, A. (1994). CML: The CommonKADS Conceptual Modelling Language. In L. Steels, A. T. Schreiber, and W. van de Velde (Editors), *A Future for Knowledge Acquisition, Proceedings of the 8th European Knowledge Acquisition Workshop*. Springer Verlag, LNAI 867, pp. 283-300.

Smith, D. R. and Lowry, M. R. (1990). Algorithm Theories and Design Tactics, *Science of Computer Programming*, 14, pp. 305—321.

Steels, L. (1990). Components of Expertise. *AI Magazine, 11(2)*, pp. 29-49.

Stout, J., Caplain, G., Marcus, S. and McDermott, J. (1988). Toward Automating Recognition of Differing Problem-Solving Demands. *International Journal of Man-Machine Studies, 29(5),* pp. 599-611.

Stutt, A. and Motta, E. (1995). Recording the design decisions of a knowledge engineering community to facilitate re-use of design models. In B. Gaines and M. Musen (Editors), *Proceedings of the 9th Banff Knowledge Acquisition Workshop*, Banff, Canada, January 26 -February 3.

ten Teije, A. (1997). Automated Configuration of Problem Solving Methods in Diagnosis, *Ph.D. thesis*, University of Amsterdam, the Netherlands.

Van de Velde, W. (1994). A Constructivist View on Knowledge Engineering. In A. Cohn (Editor), *Proceedings of the 11th European Conference on Artificial Intelligence*, pp. 727-731. Wiley and Sons, London.

Van Heijst G., Terpstra P., Wielinga B. and Shadbolt N. (1992). Using Generalized Directive Models in Knowledge Acquisition. In Th. Wetter, K.-D. Althoff, J. Boose, B. R. Gaines, M. Linster and F. Schmalhofer (Editors), *Current Developments in Knowledge Acquisition - EKAW '92*. Lecture Notes in Artificial Intelligence (LNAI) 599, Springer-Verlag, pp.112-32.

Wielinga B. J., Akkermans J. M.. and Schreiber A. T.. (1995). A Formal Analysis of Parametric Design

Problem Solving. In B. Gaines and M. A. Musen (Editors), *Proceedings of the 9th Banff Knowledge Acquisition Workshop*, pp. 37-1 - 37- 15.

Wirsing, M. (1990). *Algebraic Specification*. In J. van Leeuwen (editor), Handbook of Theoretical Computer Science, Elsevier Science Publ.

Zdrahal Z. and Motta E. (1995). An In-Depth Analysis of Propose & Revise Problem Solving Methods. In B. R. Gaines and M. Musen (Editors), *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pp. 38-1 - 38- 20.

Zdrahal Z. and Motta E. (1996). Improving Competence by Integrating Case-Based Reasoning and Heuristic Search. In B. Gaines and M. Musen (Editors), *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*. Banff, Alberta, Canada.