In: *Proceedings of the 10th European Workshop on Knowledge Acquisition, Modeling, and Management (EKAW'97)*, Lecture Notes in Artificial Intelligence (LNAI), Springer-Verlag, 1997.

Reuse of Problem-Solving Methods and Family Resemblances

Rainer Perkuhn

Institute AIFB
University of Karlsruhe (TH)
D-76128 Karlsruhe, Germany
e-mail: perkuhn@aifb.uni-karlsruhe.de

Abstract

In the last years a common notion of a Problem-Solving Method (PSM) emerged from different knowledge engineering frameworks. As a generic description of the dynamic behaviour of knowledge based systems PSMs are favored subjects of reuse. Up to now, most investigations on the reuse of PSMs focus on static features and methods as objects of reuse. By this, they ignore a lot of information of how the PSM was developed that is, in principle, entailed in the different parts of a conceptual model of a PSM.

In this paper the information of the different parts of PSMs is reconsidered from a reuse process point of view. A framework for generalized problem-solving methods is presented that describes the structure of a category of methods based on family resemblances. These generalized methods can be used to structure libraries of PSMs and - in the process of reuse - as a means to derive an incarnation, i.e. a member of its family of PSMs.

For illustrating the ideas, the approach is applied to the task rsp. problem type of parametric design.

Keywords: Problem Solving Methods, Reuse, Similarities, Categories of PSMs

1 Introduction

Most current knowledge engineering frameworks consider a notion of a Problem-Solving Method (PSM) that converged from a number of different approaches (Generic Tasks [Chandrasekaran, Johnson, and Smith, 1992], CommonKADS [Schreiber et al., 1994], Method-to-Task Approach [Eriksson et al., 1995], Components of Expertise [Steels, 1990], GDM [Terpstra et al., 1993], MIKE [Angele, Fensel, and Studer, 1996]). A PSM is a knowledge level description of a problem-oriented, but domain-independent reasoning strategy. Augmented with domain-specific knowledge a PSM can be reused across different applications. By reuse of PSMs, the development time of knowledge based systems is shortened, the quality of the system is improved, the maintenance is simplified, and the overall costs are reduced - computed against the investment cost for systematic reuse. In this context it seems to be worth while investigating systematic approaches to planned reuse.

Loosely speaking, reuse means finding an adequate component to a given task description out of a given set of reuse candidates. This leads to the following issues:

- determining what is a reuse candidate,
- organizing the set of reuse candidates (library, repository),

- describing the characteristics of tasks and components (indexing),
- supporting the lookup mechanism (retrieval),
- defining a metric for assessing the adaquacy of the found component(s).

Most current approaches take implicitly a notion of a PSM as a candidate for reuse as granted. Based upon these prerequisites they attempt to characterize the functionality of a PSM from a competence point of view. A suitable description of the competence might be used for indexing and, by matching against the goals of a given task, also for retrieval. But, instead of drawing conclusions from the competence (and further) information for the structure of the library, the approaches treat the components as totally isolated from each other.

In general, minor attention is paid to the relation between tasks, between PSMs and between tasks and PSMs. Some taxonomies for tasks rsp. problem types have been suggested (e.g. [Breuker and van de Velde, 1994]). With the assumption that a task rsp. problem type in the taxonomy can be related to a limited number of PSMs, the retrieval can be split up into two steps: First, determining the problem type, and second, selecting a PSM from the (small) set of PSMs attached to the problem type. But none of the suggested taxonomies can cope with the fact that there is no such thing as a pure analytical or generative task. Every diagnosing task involves the aspect of generating a report to present the result of the analytical step. A configuration task cannot ignore analytical knowledge to distinguish acceptable and not acceptable configurations - besides the trivial task of unconstrained configuration. A comprehensive reuse framework should be able to offer e.g. the PSM propose&revise for both tasks. Even, it might be more straightforward to relate propose&revise to a generative task since the effort spent on reusing might mirror the likeliness of the relation.

An approach that tries to capture the intertwining of tasks is that of a suite ([Breuker, 1994]). A suite relates the problem types of the above mentioned taxonomy to each other. It prescribes a standard ordering by defining a successor relation between the problem types that is assumed to be held for most applications. But, to be really useful in a reuse framework a suite contains insufficient information. It is just one meta-level structure for all tasks. The gain of information by using a suite is very small. The suite fits every task and may be instantiated accordingly. But there is no support of how this information narrows the range of applicable PSMs.

Instead of one meta-level structure it would be reasonable to suggest several structures for different task clusters or families. Each structure should handle a specific task type and contain more information than a suite.

The next section discusses the role categories can play in the reuse process and introduces the notion of family resemblances.

The following section shows how the relation between tasks can be exploited to derive a structure for a category of PSMs based on family resemblances. To assess the relevance of the features of a PSM for a family structure, this section reconsiders the motivation of the different parts of the common notion of a PSM.

Section 4 illustrates the ideas with a case study for the task rsp. problem type of parametric design; section 5 concludes and discusses related and future work.

2 Categories and Family Resemblances

Normally, only a few PSMs are really suitable to accomplish a given task. Even with an exhaustive characterization of a PSM's competence the space of all reusable PSMs has to be searched one way or the other. The retrieval process starts at one point and repeats to look for the next candidate if the previous attempt failed - until success or failure for the last candidate. Obviously, it is not reasonable to treat the space of all PSMs as an unordered flat collection because then retrieval is linear search. In an ideal reuse scenario the retrieval would start at one or possibly several entry points according to the competence of the PSMs. A mismatch would then trigger continuing search in the neighbourhood of the entry point. But for this, the notion of neighbourhoodness of PSMs has to be established on some kind of similarity measure between PSMs. The metric of similarity is based on some set of features and defined as the ratio of the weighted sum of the common features by the weighted sum of the distinctive features ([Tversky, 1977]) - given suitable sets of features and weights.

A first idea to support browsing through the space of PSMs would be to categorize the methods. Repeating this step results in a taxonomy of PSMs that can be used as a hierarchical structure of a library. Taxonomies enable hierachical instead of linear search but they are - as mentioned above - not suitable for categorizing PSMs with respect to tasks the PSMs should accomplish. But there is a way to combine most of the advantages of categories and to avoid the restrictions of strict taxonomies.

Classical theory of categorization defines categories based on necessary and sufficient conditions on a certain set of features: if (and only if) an object has all the features of a category, then it is a member, i.e. if one feature is missing, an object is definitely not member of the category. As a consequence there is a clear border and distinction between members and non-members. So, in a taxonomy it is not possible to have two different categories, e.g. one for PSMs for parametric design tasks and one for PSMs for diagnosis tasks, and an object, e.g. the PSM propose&revise, belonging to both of these categories (as shown in figure 1.a). A cognitive theory of categorization weakens the membership definition via necessary and sufficient conditions and focusses on the structure of a category according to the degree of similarity between objects ([Wittgenstein, 1953], [Rosch, 1975], [Lakoff, 1991]). In this framework a category is represented by one or several prototypes. A prototype is a possibly virtual - in the sense of not necessarily existent - object that unites all the features that are considered most prominent for this category.

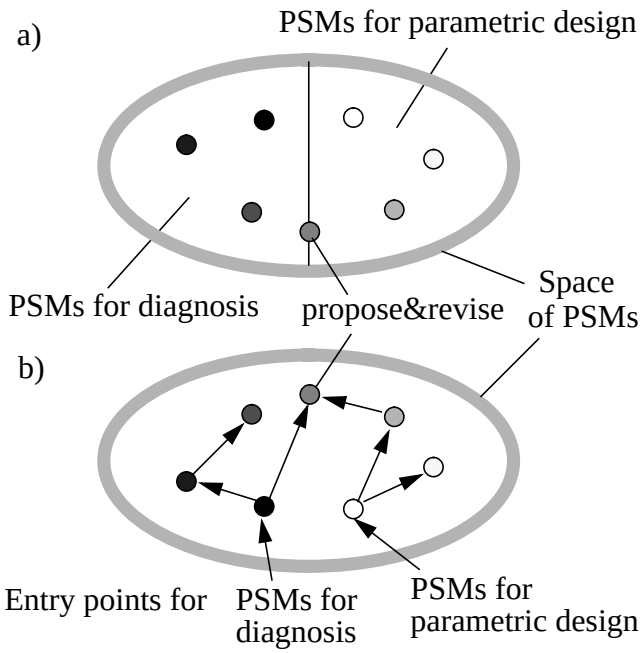

Figure 1. Categories of PSMs based
a) on necessary and sufficient conditions and
b) on family resemblances

E.g. the prototype of the category “birds“ is a two-feet flying, egg-laying animal of certain size and shape. But of course, pinguins would be called birds although they do not fly. And there is no reason to assume that the prototype has a real counterpart of exactly the same size and shape. The prototype is just a projection of the condensed experiences with objects that are considered members of the category and a constructed means to express its structure. The membership to a category is a gradual property depending on the closeness to the prototype(s). If a category is so diverse that it cannot be expressed by one prototype solely (e.g. the category “games“) the category may be also represented by several prototypes. The underlying notion of the relation between several prototypes and the closeness between a prototype and the elements of a category is termed family resemblances. In the following, some features of PSMs are discussed to derive prototypical structures. These generalized prototypical PSMs can then be used to describe a family of PSMs and as a means in the reuse process to come to an incarnation of a member of this family.

3 Families of PSMs

The notion of a PSM consists of a functional and operational specification. The functional specification describes the competence of a PSM and its interface to the environment. The operational specification comprises four main parts (s. figure 2):

- the decomposition of the task into several subtasks,

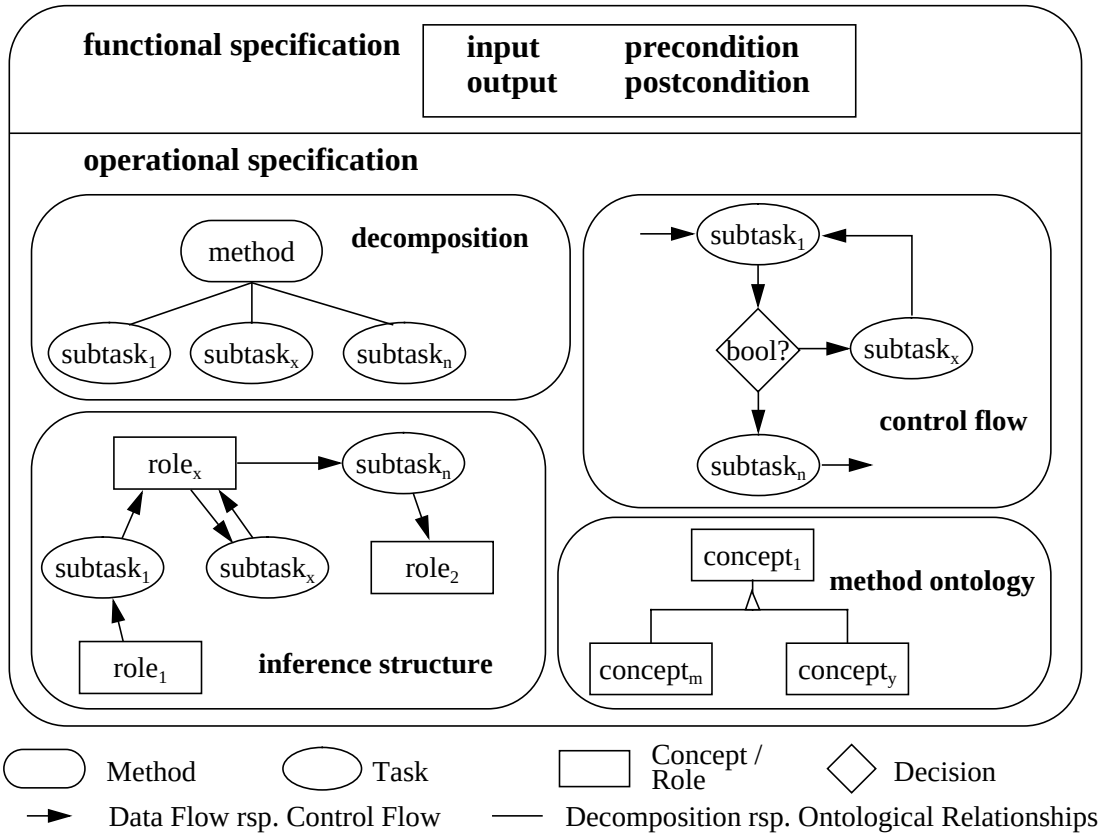

Figure 2. (Parts of the) Description of a PSM (cf. [Angele et al., 1996])

- an inference structure for this level of decomposition, i.e. the data flow between the subtasks,
- a control mechanism that constrains the order of accomplishing the subtasks in order to meet the PSM‘s competence with respect to the goals achievable by the subtasks, and
- a method ontology that describes the concepts used by the method (and their interrelationships).

Provided an adequate model of an application domain (domain ontology) and a suitable mapping of the domain-specific concepts to the method-specific concepts, a PSM can be reused across several domains and applications.

To be able to assess the relevance of the different features of a PSM for a notion of similarity, first, the motivation that led to the different parts respectively is reconsidered. Thereupon it is possible to elaborate their specific contribution to the conceptual model of a PSM.

3.1 Reconsidering Task Structure Analysis

Understanding a PSM as a way to break down one complex task into several subtasks is the knowledge engineering variant of the divide-and-conquer principle. In addition, the special aim of a task structure analysis is to figure out what aspect of the problem makes it a hard problem. Refining the goal of a task by several subgoals of its subtasks reveals the simple and crucial parts of the problem. The crucial parts show that, where, and what kinds of mostly heuristic, specific knowledge is necessary to make the whole process computational tractable.

Obviously, the decomposition process can be repeated for every new subtask until finally the tasks can be accomplished by elementary steps that can be written down straightforward as inference actions. These are the leaves in the resulting tree consisting of alternating levels of, on the one hand, matching alternative methods (or elementary inference actions) to tasks and, on the other hand, decomposing each method into several subtasks (s. figure 3).

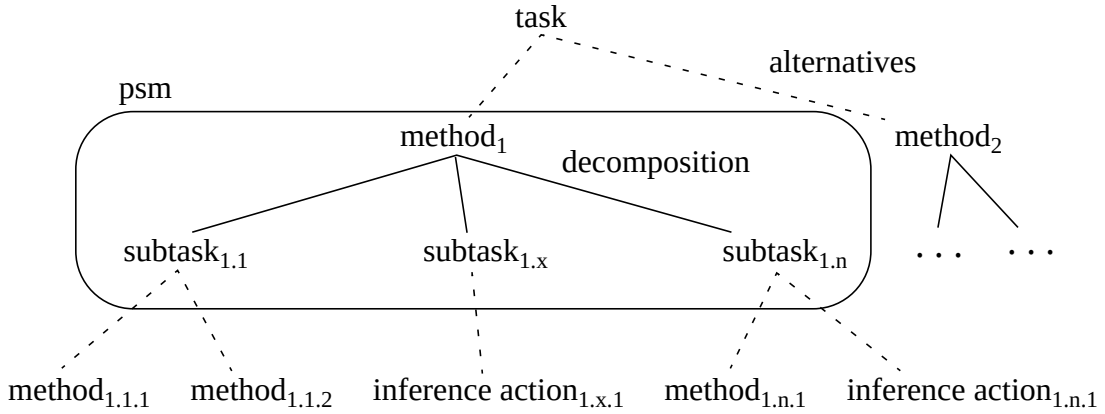

Figure 3. Embedding the notion of PSM into the method-to-task paradigm (cf. [Angele et al., 1996])

But these figures are a little bit misleading. It must be stressed that a PSM is only one

step of decomposition; a PSM determines new subtasks and, by this, sets up new goals. The decomposition does not constrain the way of how to achieve the new goals. So, the part of the tree beneath a PSM should be treated separately from the PSM itself. Of course, the complete specification of the dynamic part of a knowledge based system is one instantiation of the tree after deciding for one of each alternative. Nevertheless, a description of a conceptual model of a PSM as shown in figure 2 is set up on top of the notion of subtasks rsp. their goals and does not look into the task internal details. Also, from the reuse process point of view a PSM is primarily a means for one step of the top down analysis.

Corrolary 1 - Task decomposition

A PSM is a way to decompose one task into several subtasks. The goal of the overall subtask is broken down by setting up new subgoals for the subtasks. A PSM does not prescribe how these (sub-)goals have to be achieved.

The notion of a PSM does not restrict the way how the subgoals are related to each other and to the goal of the overall task. Of course, if all subtasks are successfully accomplished, the goal of the major task should have been met. But the subgoals may depend on each other and be coherent. The subgoals need not to be mutually exclusive. One goal may be subsumed by another - provided that the specific knowledge required to achieve this goal is available. So, in the following the goals of a method‘s subtasks should not be considered disjoint. At least for a subset of all possible inputs they may overlap.

The following example is intended to illustrate this idea. If one task is to decide for the next candidate (e.g an employee to be assigned to an office room) the method “select“ sets up two subtasks (in other cases, more and different subtasks would be possible).

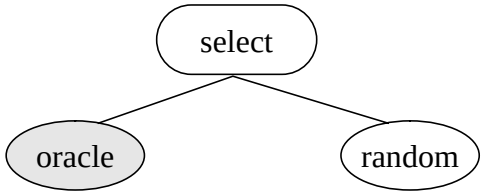

Figure 4. Decomposition with a coloured (optional) component

“Oracle“ has the goal to select always the best next candidate, “random“ only to present one (arbitrarily selected) candidate (e.g. due to the alphabetical order of their names). Obviously, the goal of “oracle“ is subsumed by the goal of “random“ but the achievability of this goal depends on whether the knowledge of the best next candidate is available. Actually, three different cases have to be distinguished: Methods for “oracle“ know the answer for a) every question, b) some questions, or c) no question. I.e. they yield the best next candidate for a) every input, b) some input, or c) no input. The cases a) and c) could be handled straightforward: In a) oracle is the alternative that could be chosen to perform select, in c) random has to be taken. But case b) requires some way to express that random has to cope with the input oracle cannot handle. Both are subtasks of the same method and could not be considered alternatives. Since an omniscent oracle will hardly ever been found in reality, case a) will be ignored in the following. The other two cases are represented in figure 4: The difference between them is described by a colour, but the similarity is captured by integrating them into one common underlying structure. In the extreme case c) that no specific oracle knowledge is available it would be reasonable to be able to wipe out these

nodes from the figures of the task decomposition tree. Therefore, they should be marked as optional components. In the reuse process the optional components can be checked with respect to the necessary knowledge and be treated accordingly.

Corrolary 2 - Relations between subtasks, optional components

The goals of the subtasks of a PSM are not necessarily mutually exclusive. Some subgoals may overlap, some subgoals may be subsumed by other subgoals. In case of subsumption the subsumed goal should be marked as optional.

The relation between the optional subtasks and the specific knowledge they require is documented best as a kind of inference structure.

3.2 Reconsidering Inference Structures

An inference structure describes the data flow between the subtasks that are the result of the decomposition. The ways of the possible data flow is represented by directed links between the subtasks; the structure of the data is specified as roles on the links (s. figure 2). Roles contain the (derived) static knowledge of the inference structure. A role may be a data interface between two subtasks or between a subtask and the domain specific knowledge. In the latter case the role the domain knowledge plays in the inference process is expressed in form of a mapping.

Inference structures can be seen in the reuse process twofold. On the one hand, as part of a complete specification they are objects for reuse, on the other hand, as interpretation models they are means for reuse. In the context of a reuse process-oriented framework the latter aspect is more attractive. As an interpretation model an inference structure is used to check whether the necessary knowledge for each subtask is available from the application domain rsp. whether any domain knowledge can be elicitated and interpreted in a way that it can fill the role in the inference structure. Elicitating and interpreting are highly creative acts but spending time and effort on them is reasonable only to a certain extent. Finally, the interpretation model is judged binary either suitable or not. But taken the motivation of the task decomposition seriously there is no reason to assume that the applicability of one subtask depends on the availability of some specific knowledge other subtasks require. A really useful interpretation structure should separate these affairs. Then, the evaluation of the interpretation structure does not yield a binary, but a gradually decision - depending on what different kinds of knowledge are available.

Of course, some kinds of knowledge are really necessary and are, as such, the crucial features for a binary decision. But, in the case that the goal of one subtask is subsumed by or overlaps with goals of other subtasks, it is sufficient that their common goal can be achieved one way or the other. If the goal of the coherent tasks can be achieved by some subset of them alone, the rest is optional. In the following the relation between an optional role and the subtasks that use the knowledge of this role is defined and illustrated as a colour. The colouring of an inference

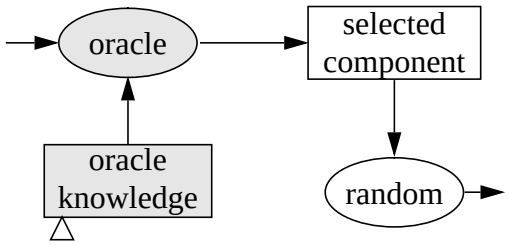

Figure 5. Inference struture with coloured (optional) components

structure shows what kinds of knowledge are not binary decisive for the suitability of the structure and what dependants of the role also can be wiped out if the knowledge is not available. This way, this structure can be used to derive different incarnations that are all members of the same family of PSMs.

Corrolary 3 - Coloured inference structures

> An inference structure describes the data flow between the subtasks (that are the result of the decomposition) via roles. The coherence between optional tasks and the specific knowledge they require is defined by a colour. All nodes of the same colour can be deleted from the structure if the specific knowledge role cannot be filled.

The information that was collected up to this point to characterize a family of PSMs can be condensed to a notion of a coloured inference structure including a specification of the goals of the subtasks. The information of the decomposition is an intrinsic feature of the inference structure. Most part of this information can already be specified by a new version of the language KARL ([Angele et al., 1996]). The revised version of KARL allows to describe PSMs on a formal and conceptual level, especially goals and competences can be expressed by pre- and postconditions. But, up to now there is no way to treat the different parts separately and, by this, to really support the reuse process. For illustration purposes an alternative is preferred: Another way to express the characteristics of a PSM is a graph grammar rule (s. figure 5, cf. GDM [Terpstra et al., 1993]): A method is rewritten by its inference structure. Missing optional knowledge leads to deleting all same coloured parts of the inference structure. But it should be noted that it is a misinterpretation to treat a task-decomposition tree only as the result of recursive applications of grammar rules, i.e. as a word of the language accepted by the grammar. A method can be seen as a rule, but the selection of a PSM to a given task involves some aspects that cannot be expressed if hidden in rewriting. The matching of a PSM‘s competence against the goal of a task may reveal (hidden) assumptions of the PSM, e.g. the single-fault-assumption or the assumption of acyclic revise rules ([Benjamins, Fensel, and Straatman, 1996]). Assumptions trigger different processes of checking properties of the domain knowledge or reformulating the task ([Fensel and Schoenegge, 1997]). Afterwards, if the method-to-task adaptation did not run into a dead end, plugging in the method enables a new rewriting step. A PSM grammar cannot separate the different aspects, it ignores the essence of a PSM. But it is a useful instrument to reconstruct the development process.

Most of the issues discussed up to this point can be easily transferred to the notion of a method ontology. The concepts and relations may be also optional and coloured according to the colours of the roles (and vice versa). Further, it would be interesting to compare the relation between method and subtask ontologies (cf. [Studer et al., 1996]) to the expressiveness of the graph grammar rules for the methods. Since this paper attempts to present a coherent view of the especially dynamic part of a PSM, the aspect of method ontologies is not elaborated any further. Instead a brief sketch of how control knowledge can be characterized is presented.

3.3 Reconsidering Control Knowledge

The last aspect of a PSM that has not been considered yet is the ordering of the subta-

sks. The control knowledge specifies when each subtask may be accomplished - dependent on the goals of other subtasks. Thinking in the paradigm of e.g. Structured Analysis control flow may expressed in terms of sequence, alternative, and iteration.

If there is a reason to stick to this paradigm and to the assumption of a single processor system the control can be illustrated by flow diagrams (s. figure 2). Then, the control knowledge of a generalized PSM with optional components can be described

- by enumerating all control flows for all possible incarnations or
- by using coloured flow diagrams similar to coloured inference structures.

Leaving the paradigm and relaxing the single processor assumption opens new ways to describe control knowledge.

Once again, the motivation of the task decomposition helps to introduce a wide notion of control knowledge. The initial state before task decomposition consists of a problem to be solved. Unfortunately, the solution of the problem is not obvious. The only way to solve the problem in one step would be to ask an oracle that always yields the correct solution. This is the first trivial model a knowledge engineer might have about the problem solving process. The aim of decomposition is to set up subgoals that could be achieved with less miraculous knowledge. The knowledge engineer attempts to break down the goal to be able to cope with - at least some of - the subgoals. The degree of how well understood a subtask is might affect organizational models especially how the subtasks could/should be distributed between a human expert and the system. A very critical but incomplete understood task e.g. deciding what to do next in an emergency case of a nuclear plant should rest with a human expert as long as possible.

Generalizing this szenario of two agents to a multi-agent szenario enables constellations in which each subtask has its own agent. Of course, the agents need a way to communicate with each other e.g. via a blackboard. But on top of the constellation control knowledge can be easily specified without the restrictions of structured analysis. Every agent just has to know what kinds of knowledge it expects from which (data delivering) predecessor and what conditions must hold on this input. Then, an agent can start accomplishing its task immediately when the expected input is provided and satisfies the constraints.

Corrolary 4 - Control knowledge

Control knowledge has to specify what constraints have to be satisfied by the input a subtasks expects from its (data delivering) predecessors. Subtasks can be accomplished as soon as the complete expected input is provided that satisfies the constraints.

By this, this framework also captures parallel - possibly competing - subtasks that will become even more important in the near future. Nevertheless, coming back from internet visions downto earth, if agent-oriented specifying of control is not wanted, the framework can be used at least as a means to derive a control flow specification in the mentioned terms: Starting with the “last“ subtask that delivers the output of the method the control flow specification may be constructed backwards - by iteratively comparing what a subtask expects with what its predecessors can provide it with.

4 One Family of PSMs for the Task Type Parametric Design

In the following the ideas of the last two sections are illustrated with an example. A simplified version of the task type parametric design (cf. [Motta and Zdrahal, 1996]) sets up a goal for the framework. Then different subgoals and the relation to each other are discussed and result in a description of a simple family structure.

The similarity of names for the subtasks to subtask names of well-known PSMs is purely arbitrary. They are not intended to suggest a 1:1-mapping of the family structure to existing PSMs. But they may indicate a way of how a similar incarnation of the family structure can be derived.

4.1 Parametric Design

The task type of parametric design works with a system model that describes the structure of the system to be designed with a number of parameters. Designing means that a value has to be assigned to every parameter from its specific range.

SYSTEM

$PARAMETER_1$: { $RANGE_1$}

. . .

$PARAMETER_n$: { $RANGE_n$}

If there is no restriction of what a correct (or good) design is, then the values can be arbitrarily taken from the respective ranges. But normally designs are least distinguished between correct and incorrect assignments. Initially well-chosen values enable a straightforward search for a correct assignment. Sometimes, a subset of the parameters can be made responsible for the failure and, sometimes, the reason why a design is incorrect can be explained and named. The reason becomes particularly useful if it can be connected to a way how to improve the assignment at least locally. But this local change may effect that the assignment is still incorrect due to other reasons.

4.2 Subtasks and the Family Structure

The overall goal of a parametric design task is to find an assignment that maps each parameter to a value out of its range. If only a subset of possible assignments are valid assignments the validity must be checked, e.g. with a boolean function that yields TRUE if the assignment is valid and FALSE otherwise. So, constrained parametric design has to find a valid assignment with respect to this test. Of course, normally, a design is assessed according to a quality metric and the goal is to find the best possible design. But for simplification reasons this aspect is not considered any further.

Based on this framework, the task can at least be split up into two subtasks: One that has the goal to find a possibly partial assignment and the other one has the goal to decide whether this assignment is valid or not. The subtasks do not restrict the degree of freedom whether the first task has to generate a complete assignment or whether it starts with a subset of the parameters that is extended after a successful test. This decision can be seen as part of the control knowledge that is not discussed in detail. To briefly sketch the idea, the successing task has to specify whether it expects an assignment for one, several or all parameters.

This first vague approximation of a “method“ is shown in figure 6 as uncoloured objects.

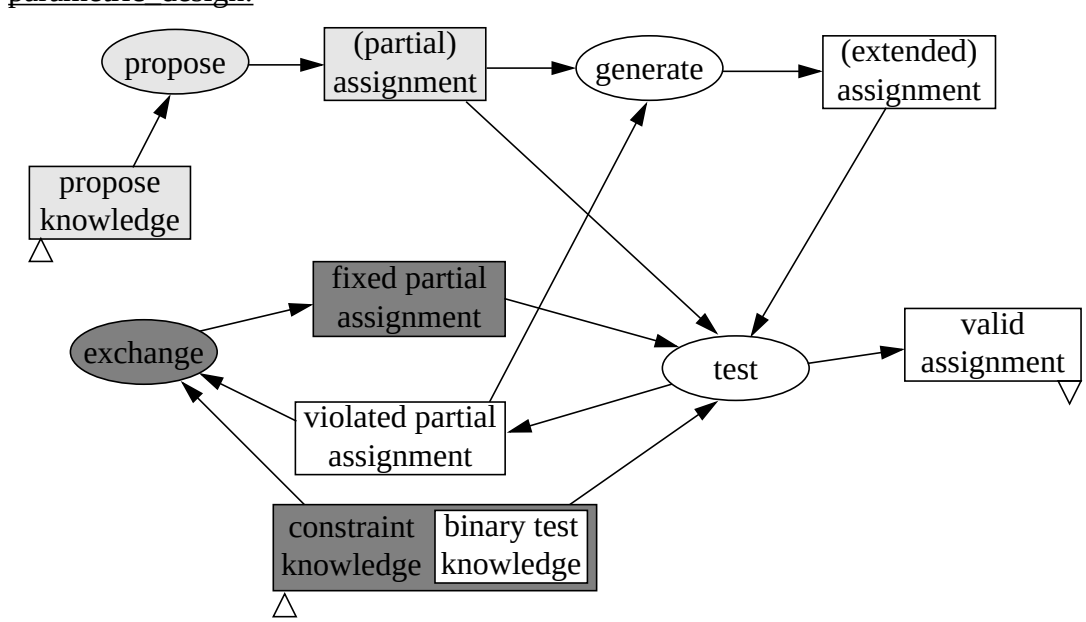

Figure 6. Family structure of the category of PSMs for the task type parametric design

Instead of generating arbitrary assignments and searching the space of all possible assignments unsystematically, knowledge about good (initial) values for at least some of the parameters would improve the search. Of course, only an oracle could always offer valid assignments but starting with presumably good values lowers the average search time. The goal of this task of proposing “good“ values is somehow subsumed by the goal of “generate“. Even less efficient “generate“ will deliver the same values - but after some failed iterations of generating and testing. So, it is not really necessary to have propose knowledge for every parameter because “generate“ can help in that situation as a backup or default subtask. Presumably in most cases there is propose knowledge only for a subset of the parameters so that a partial assignment has to be extended by arbitrary generated values for the rest of the parameters.

The relation of this goal to propose knowledge and the optionality of this coherent area is shown by light grey colouring of the nodes in the left upper corner of figure 6.

More sophisticated test knowledge may also yield a subset of the parameters that is responsible for the violation. Then, the violation can be repaired locally, i.e. only values of some parameters of this subset have to be exchanged. This specific knowledge and the dependent nodes of the structure are marked with dark grey in figure 6.

If there is no special knowledge of how this could be done, again arbitrary partial assignments for this subset of parameters can be generated and tested until the partial assignment does not cause the violation any more (if possible). Although this local repair helps to avoid testing a lot of assignments it does not guarantee that the resulting assignment is valid. It may be invalid due to other conditions so the test has to follow once again. The same holds even if some special revise knowledge is available that tells how to fix a violated constraint. Since this goal is subsumed by the rest of this structure

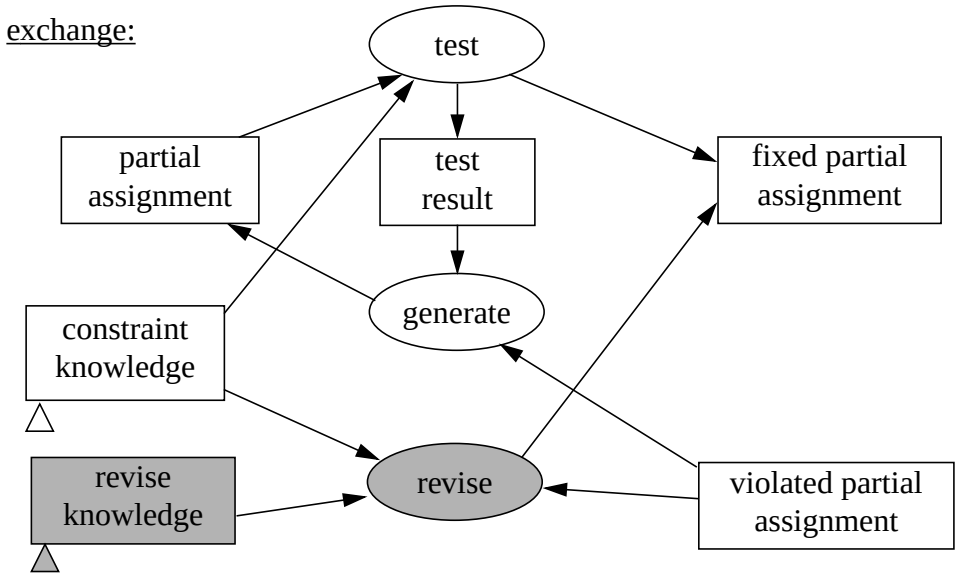

Figure 7. Family structure of the category of PSMs for the (sub-)task type exchange

the subtask and the required knowledge are marked as optional and coloured middle grey (cf. figure 7).

In this family the subtask “test“ mostly determines the control flow. If it expects a complete model, then possibly the two preceding subtasks “propose“ and “generate“ have to be repeated until all parameters have assigned values, otherwise a subset of assigned parameters from one of them once accomplished may be sufficient (Complete vs. Extend Model Then Revise, [Motta and Zdrahal, 1996]). The “generate“ subtask only has to be performed if the subtask “test“ demands an assignment for some parameter the “propose“ task does not have the knowledge about good values for.

4.3 Using the Family Structure to Derive PSMs

The resulting structure represents one family of PSMs for parametric design tasks. As mentioned above there is no need that a counterpart PSM exists in reality. But it can be used to partially structure the library and as a means to derive different incarnations.

Retrieving a component from the library is separated into two steps. First, one or more family structures are selected from the library out of a limited number of “reference structures“ according to the competence. And second, each selected structure is used to browse through the category it describes. The first step is not the subject of this paper but could be briefly sketched as matching the competence against the task goal. For the described parametric design family it is necessary that the central concept of the task is a system model that can be described by parameters. If the central activity is assigning “correct“ values to the parameters, then this family is a possible candidate for the next step. Then the colouring of the family structure can be used as the underlying principle for a questionnaire. The uncoloured roles are necessary features - if they cannot be filled, the structure is excluded from further consideration. The coloured roles mirror the family structure of the category, they are useful but not necessary features. If these roles cannot be filled, all nodes of the same colour can be wiped out. Otherwise the process continues with refining the structures. But in any case, the structure is a means to navigate through the space of PSMs.

Depending on the availability of the above described roles it seems to be straightforward to derive incarnations of the family that are very close rsp. can be refined to well-known PSMs:

- if none of the optional roles is available, the structure is reduced to almost generate&test (cf. figure 8),

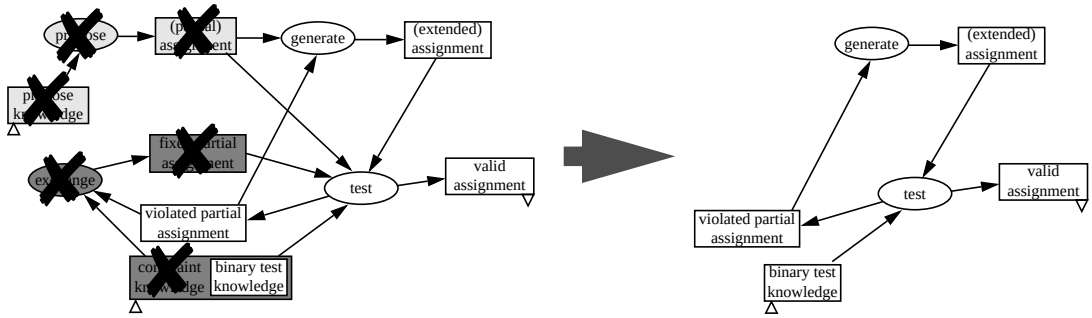

Figure 8. Instantiating the family structure to the PSM generate&test

- if all optional roles are available, the structure strongly resembles propose&revise.

The insight that the two methods are very closely related to each other (according to family resemblances) is strongly supported by the mincer metaphor that showed how one could be transformed into the other (cf. [Fensel and Straatman, 1996]).

Table 1 summarizes the scope of the family described by figure 6 and figure 7. Every (optional) role triggers a question, the dependencies between roles trigger follow-up questions. Further roles independent from the two shown in the table would spawn up further dimensions.

Is knowledge of this special kind available?	binary decision? No ↓ Yes →	constraint knowledge? No ↓ Yes →	revise knowledge? No ↓	revise knowledge? Yes ↓
Yes → propose knowledge? No →	unconstrained propose (&generate)	propose (&generate) &test	propose (&generate) &exchange	propose (&generate) &revise
	unconstrained generate	generate &test	generate &exchange	generate &revise

Table 1. Underlying structure for questionnaire ("→" means follow-up question rsp. decision for row or column, shading means already documented PSM)

By representing the structure of a family of PSMs explicitly a questionnaire can easily be derived. Furthermore, by making explicitly the dependencies between the optional roles e.g. in a table similarity between two members can be expressed: Neighbours in the table are very closely related to each other, greater distances express a lower similarity measure.

5 Conclusion, Related and Future Work

The presented framework is one of the few approaches to tackle the problem of the relation between the subtasks of specific task rsp. problem types. Based on a non-classical notion of categorization a way to structure a family of PSMs is suggested. The description of this family structure is grounded on the well-established, but slightly reconsidered notion of a PSM. The paper aimed to shift the focus from a product-oriented to a process-oriented view on the reuse of PSMs. For this, generalized PSMs incorporate more information than only the binary distinction whether a PSM is suitable or not. By deriving an incarnation of the family this artefact is configured according to the specific circumstances.

Other approaches that investigate the relation between tasks discuss taxonomies, task-decomposition methods, or suites.

Taxonomies (e.g. [Breuker and van de Velde, 1994]) would be ideal for hierarchical search but they cannot cope with the fact that there are no clear borders between categories of PSMs for different task types.

Task-decomposition methods ([Benjamins, 1993]) try to cover a space of PSMs for one task but they suffer from the same disadvantages as taxonomies. They are not able to express the specific difference between the decomposition of the method and the sophisticated selection and adaptation process before plugging in a method.

The deficiencies of the notion of a suite ([Breuker, 1994]) are already discussed above. Although this idea is applied to specific task types ([Benjamins, 1995]) the benefits of a suite remain unclear. Neither the way how to use a suite, nor the way how to exploit the result is documented.

But there is some other work that already use an implicit notion of a family structure.

In the context of the CommonKADS approach a collection of questions rsp. questionnaires about task features is discussed ([Aamodt et al., 1993]) to support the selection of a method for an identified task type. GDM proposes a set of rewrite rules. The configuration of a PSM depends on the selection and application of some of the GDM rules ([Terpstra et al., 1993]). Although, at first glance, these are different issues there is a strong relation between the selection of rules in GDM and the ordering of questions about the task features: Both check some properties of the task rsp. problem type and yield a similar result, namely a PSM. The rule selection and the question ordering look also very similar to wiping out coloured nodes in this approach but GDM and CommonKADS use the common underlying structure of similar family members only implicitly. An explicit representation of the family structure enables a systematic approach to construct questionnaires and to guide the application of grammar rules.

Some new results on automated configuration of PSMs also take some kind of family structure for granted. In this framework, PSMs are derived from more common but task specific parametrized structures ([ten Teije, 1997]). One structure is examined in detail for diagnosis. With specific values for the parameters this structure is instantiated to a PSM. Some parameters may obtain the empty set as value with the consequence that they can be ignored. This resembles the optional components in the approach discussed in this paper, but the relation to the other subtasks and the embedding in an incremental reuse process is not considered.

In the future the relation of family structures to parameters (in the configuration-of-

PMSs-as-parametric-design-paradigm [ten Teije et al., 1996]) and to assumptions (cf. [Benjamins, Fensel, and Straatman, 1996]) will be investigated. Furthermore the specification of control knowledge has to be elaborated. For this, more family structures of different task types have to be developped and specified formally - currently the task types diagnosis and assessment are investigated. Most insights are then expected from using the family structures in real life reuse applications.

Acknowledgements

I would like to thank Rudi Studer, Dieter Fensel, Robert Engels, Stefan Decker, and Michael Erdmann for many interesting discussions, and the anonymous referees for valuable and stimulating comments.

References

[Aamodt et al., 1993] A. Aamodt, B. Benus, C. Duursma, Chr. Tomlinson, R. Schrooten, and W. van der Velde: *Task Features and their Use in CommonKADS.* Deliverable 1.5. Version 1.0, Consortium, University of Amsterdam, 1993.

[Angele et al., 1996] J. Angele, S. Decker, R. Perkuhn, and R. Studer: Modeling Problem Solving Methods in New KARL. In: *Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'96),* Banff, Canada, November 1996, 1-1 - 1-18.

[Angele, Fensel, and Studer, 1996] J. Angele, D. Fensel, and R. Studer: Domain and Task Modeling in MIKE. In: A. Sutcliffe, D. Benyon, F. van Assche (Eds.): *Domain Knowledge for Interactive System Design*, Chapman & Hall, 1996, 149-163.

[Benjamins, 1993] R. Benjamins: *Problem Solving Methods for Diagnosis.* Ph.D. Thesis, University of Amsterdam, Amsterdam, 1993.

[Benjamins, 1995] R. Benjamins: Suite in Dm: A suite of problem types in diagnostic methods. In: *Proceedings of the Knowledge Engineering Forum '95 (KEF'95),* St. Augustin, Germany, March 1995, 7-18.

[Benjamins, Fensel, and Straatman, 1996] R. Benjamins, D. Fensel, and R. Straatman: 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 1996, 408-412.

[Breuker, 1994] J.A. Breuker : A suite of problem types. In: [Breuker and van de Velde, 1994], 57-87.

[Breuker and van de Velde, 1994] J.A. Breuker and W. van de Velde (Eds.): *The CommonKADS Library for Expertise Modeling*. IOS Press, Amsterdam, 1994.

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

[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, 1995, 293-326.

[Fensel and Schoenegge, 1997] D. Fensel and A. Schoenegge: Hunting for Assumptions as Developing Method for Problem-Solving Methods. To appear in: *Workshop Proceedings Problem-solving Methods for Knowledge-based Systems in Connection with the Fifteenth International Joint Conference on Artificial*

Intelligence (IJCAI'97), Nagoya, Japan, August 1997.

[Fensel and Straatman, 1996] D. Fensel and R. Straatman: Problem-Solving Methods: Making Assumptions for Efficiency Reasons. In: *Proceedings of the 9th European Knowledge Acquisition Workshop (EKAW'96)*, Nottingham, England, May 1996, Lecture Notes in Artificial Intelligence (LNAI), vol. 1076, Springer-Verlag, Berlin, 1996, 17-32.

[Gennari et al., 1994] J.H. Gennari, S. Tu, Th.E. Rothenfluh, and M.A. Musen: Mapping Domains to Methods in Support of Reuse. *International Journal of Human-Computer Studies (IJHCS)*, 41, 1994, 399-424.

[Lakoff, 1991] G. Lakoff: *Women, fire, and dangerous things: what categories reveal about the mind.* University of Chicago Press, Chicago, 1991.

[Motta and Zdrahal, 1996] E. Motta and Z. Zdrahal: Parametric Design Problem Solving. In: *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge Based Systems Workshop (KAW'96)*, Banff, Canada, November 1996, 9-1 - 9-20.

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

[Puerta et al., 1992] A. R. Puerta, J. W. Egar, S. W. Tu, and M. A. Musen: A Multiple-Method Knowledge Acquisition Shell for the Automatic Generation of Knowledge Acquisition Tools. *Knowledge Acquisition*, 4, 1992, 171-196.

[Rosch, 1975] E. Rosch and C.B. Mervis: Family Resemblances: Studies in the Internal Structure of Categories. *Cognitive Psychology*, 7, 1975, 573 - 605.

[Schreiber et al., 1994] A.Th. Schreiber, B.J. Wielinga, R. de Hoog, H. Akkermans, and W. van de Velde: CommonKADS: A Comprehensive Methodology for KBS Development. *IEEE Expert*, December 1994, 28-37.

[Schreiber, Wielinga, and Breuker, 1993] G. Schreiber, B. Wielinga, and J. Breuker (Eds.): *KADS. A Principled Approach to Knowledge-Based System Development*. Knowledge-Based Systems, vol. 11, Academic Press, London, 1993.

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

[Studer et al., 1996] R. Studer, H. Eriksson, J.H. Gennari, S. Tu, D. Fensel, and M.A. Musen: Ontologies and the Configuration of Problem Solving Methods. In: *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge Based Systems Workshop (KAW'96)*, Banff, Canada, November 1996, 11-1 - 11-20.

[ten Teije, 1997] A. ten Teije: *Automated Configuration of Problem Solving Methods in Diagnosis*. Ph.D. Thesis, University of Amsterdam, Amsterdam, 1997.

[ten Teije et al., 1996] A. ten Teije, F. van Harmelen, Guus Schreiber, and Bob Wielinga: Construction of Problem Solving Methods as Parametric Design. In: *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge Based Systems Workshop (KAW'96)*, Banff, Canada, November 1996, 12-1 - 12-20.

[Terpstra et al., 1993] P. Terpstra, G. van Heijst, B. Wielinga, and N. Shadbolt: Knowledge Acquisition Support Through Generalized Directive Models. In: J.-M. David, J.-P. Krivine, and R. Simmons (eds.): *Second Generation Expert Systems*, Springer, Berlin, 1993, 428-455.

[Tversky, 1977] S. Tversky: Features of similarity. *Psychological Review*, 84, 1977, 327 - 352.

[Wittgenstein, 1953] L. Wittgenstein: *Philosophical Investigations*. Macmillan, New York, 1953.