Problem Solving for Redesign*

Anita Pos[1] and Hans Akkermans[1] and Remco Straatman[2]

[1] University of Twente (UT)
Department of Computer Science
P.O. Box 217
NL-7500 AE Enschede
The Netherlands
E-mail: {pos,akkerman}@cs.utwente.nl
[2] University of Amsterdam (UvA)
Department of Social Science Informatics (SWI)
Roetersstraat 15
1081 WB Amsterdam
The Netherlands
E-mail: remco@swi.psy.uva.nl

Abstract. A knowledge-level analysis of complex tasks like diagnosis and design can give us a better understanding of these tasks in terms of the goals they aim to achieve and the different ways to achieve these goals. In this paper we present a knowledge-level analysis of *redesign*. Redesign is viewed as a family of methods based on some common principles, and a number of dimensions along which redesign problem solving methods can vary are distinguished. By examining the problem-solving behavior of a number of existing redesign systems and approaches, we came up with a collection of problem-solving methods for redesign and developed a task-method structure for redesign.
In constructing a system for redesign a large number of knowledge-related choices and decisions are made. In order to describe all relevant choices in redesign problem solving, we have to extend the current notion of possible relations between tasks and methods in a PSM architecture. The *realization* of a task by a problem-solving method, and the *decomposition* of a problem-solving method into subtasks are the most common relations in a PSM architecture. However, we suggest to extend these relations with the notions of *task refinement* and *method refinement*. These notions represent intermediate decisions in a task-method structure, in which the competence of a task or method is refined without immediately paying attention to its operationalization in terms of subtasks. Explicit representation of this kind of intermediate decisions helps to make and represent decisions in a more piecemeal fashion.

1 Introduction

The concept of reusable *problem-solving methods (PSMs)* is present in many current knowledge engineering frameworks, e.g. Generic Tasks (Chandrasekaran, 1988), Com-

* This work has been funded by NWO/SION within project 612-322-316, "Evolutionary design in knowledge-based systems" (the REVISE-project). Participants in the REVISE-project are: the TWIST group at the University of Amsterdam, the AI department oAmsterdam and the STEVIN group at the University

ponents of Expertise (Steels, 1990), Method-to-Task (Gennari *et al.*, 1994), role-limiting methods (McDermott, 1988), GTMD (O'Hara & Shadbolt, 1993) and COMMONKADS (Wielinga *et al.*, 1993). The interest in PSMs originates from the need to describe and explicate generic aspects of the problem solving behavior of knowledge based systems. One possible use of PSMs lies in comparing the problem solving behavior of different knowledge based approaches and systems for the same or similar tasks.

In this paper we focus on *comparing* PSMs for *redesign*, and on identifying and representing relevant choices in constructing and selecting PSMs for redesign tasks. As the notion of redesign incorporates many different methods it is best characterized as a family of problem-solving methods. We have made a knowledge-level analysis of redesign, and come up with a collection of problem-solving methods for this task. This collection was obtained in a bottom-up manner by examining the problem solving behavior of existing redesign systems and approaches, most notably those developed in the REVISE-project. Within this project the redesign of technical systems (Eldonk *et al.*, 1996), simulation models (Pos *et al.*, 1997), compositional architectures (Brazier *et al.*, 1996b) and control knowledge in knowledge based systems (Straatman, 1995) is studied.

The number of knowledge based decisions made during the design of a redesign system is large and diverse. In order to describe all relevant choices and decisions in a task-method structure, we need to extend the current notion of possible relations between tasks and methods. The general notion of a prblem-solving method as a direct link between a task goal and the decomposition of this task into subtasks, is not in itself sufficient to describe all the relevant choices and decisions. The notions of task refinement and method refinement are introduced to represent intermediate decisions in constructing and selecting PSMs. In these intermediate decisions, the competence of a task or method is refined without immediate attention being paid to the operationalization of this task or method. This corresponds to changing the exact nature of the problem with the aim of making it easier to solve. Explicit representation of these types of intermediate decisions help to represent decisions in a more piecemeal fashion.

The structure of the paper is as follows: In section 2, we present an extended architecture of a task-method structure, based on the additional notions of task refinement and method refinement. In section 3 we present our view on redesign, and distinguish a number of dimensions along which redesign approaches can differ. We also present some excerpts from a collection of problem-solving methods for redesign, and use these as examples to illustrate the additional refinement relations proposed and their role in describing and comparing tasks and problem-solving methods in a task-method structure. Section 4 concludes the paper, and points out some implications for knowledge engineering.

2 Modeling framework

A general view on tasks, methods and their mutual relations is the following: A *task* is characterized by a goal it can achieve. A task can potentially be realized by a number of problem-solving methods (PSMs). A *problem-solving method* describes a way to solve a task: it decomposes a task into subtasks, each associated with a subgoal, and/or into primitive inferences, that directly achieve goals. Recently, focus has shifted from the description of reasoning strategies per se to the description of assumptions underlying these reasoning strategies (Akkermans *et al.*, 1994; Benjamins *et al.*, 1996). The idea is that PSMs provide solutions to tasks by making *assumptions* about the precise definition of their functionality, and about the available domain knowledge. With this idea in mind,

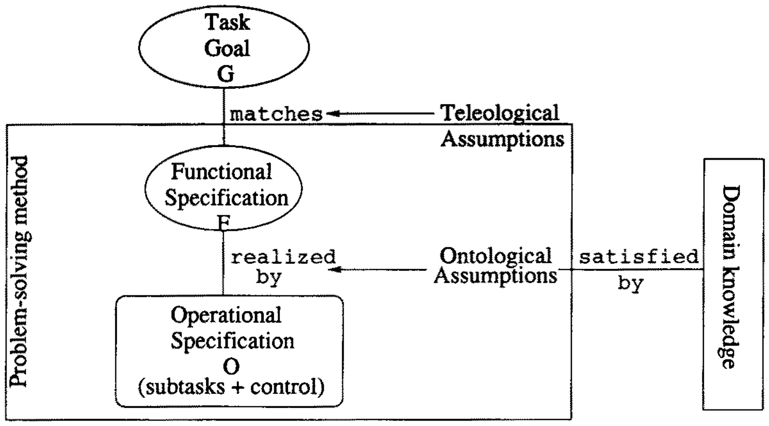

Fig. 1. The architecture of a PSM.

(Benjamins *et al.*, 1996) states that a PSM consists of three subparts (presented in figure 1):

- its *functional specification*. This is a declarative description of the input/output behavior of the PSM.
- its *operational specification*. This is an account of how to realize that behavior. The operational specification of a PSM decomposes a task into subtasks and/or primitive inferences, and defines an ordering over these operators.
- its *assumptions*. Problem-solving methods make assumptions on the precise definition of their functionality (*teleological assumptions*) and on the availability and properties of domain knowledge (*ontological assumptions*). Teleological assumptions are introduced in matching a task goal to the functional specification of a PSM if the functional specification of a PSM is more restricted than the task goal it is meant to achieve. Ontological assumptions are introduced when realizing the functional specification of a PSM by its operational specification, since operationalization often depends on the availability and properties of domain knowledge.

We tried to use this framework to establish a task-method structure for problem solving behavior of different redesign systems and approaches, similar to the task-method structure for diagnosis presented in (Benjamins, 1993). However, although the PSM architecture in (Benjamins *et al.*, 1996) is rich, it does not adequately cover all (intermediate) types of design decisions we encountered. The current framework provides a limited number of relations between tasks and methods: a task goals can be *matched* to the functional specification of a problem-solving method, and a problem-solving method can be *realized* by an operational specification which decomposes the functional specification in a number of subtasks. While trying to construct a task-method structure for redesign, we encountered two types of *intermediate* decisions that can not easily be represented in this framework; we call them *task refinement* and *method refinement*, respectively.

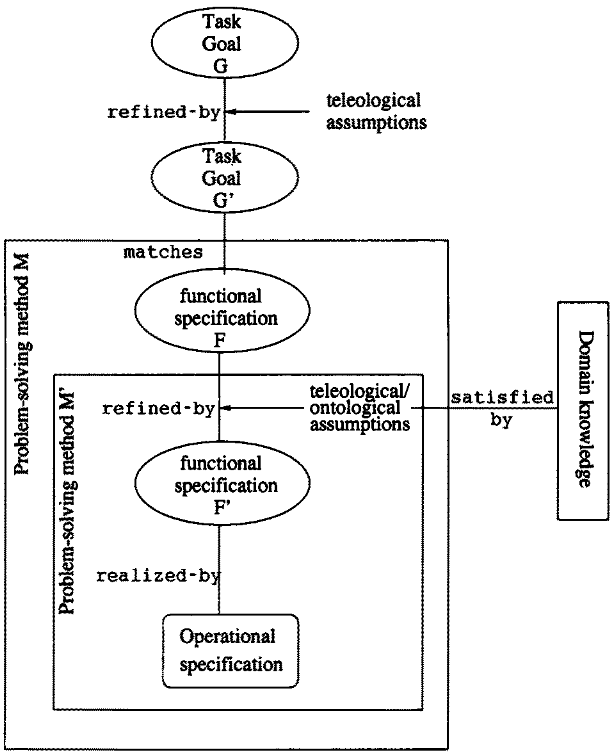
Task Goal G
refined-by
teleological assumptions
Task Goal G'
Problem-solving method M
matches
functional specification F
Problem-solving method M'
refined-by
teleological/ ontological assumptions
satisfied by
Domain knowledge
functional specification F'
realized-by
Operational specification

Fig. 2. An extended architecture for PSMs.

- *Task refinement* refines a task goal G into a more specific task goal G' but without directly making explicit *how* the task should be solved. Refinement of a task specializes a task goal into a weaker task goal by making additional teleological assumptions about the precise definition of the tasks functionality.
- *Method refinement* refines the functional specification F of a PSM into a more refined functional specification F' without making explicit how exactly this functional representation should be operationalized in terms of (control over) subtasks and/or primitive inferences. Competence refinement of a problem-solving method may introduce both additional teleological assumptions on the precise definition of the functional specification of a PSM as well as additional ontological assumptions on additional knowledge structures that should be available in the domain knowledge.

The *competence description* (Akkermans *et al.*, 1994) of a method or goal describes its problem space and requirements on the solution it produces. Under this definition, both decisions described above can be viewed as competence refinements, since they refine the functional specification of a task or method without directly making explicit how this competence could be achieved. This allows us to describe *families* of strongly related problem-solving methods, which can now be described purely in terms of their *competence*. Both forms of refinement are only to be used as *intermediate* steps in a task-method structure: the ultimate goal of constructing a task-method structure is still to relate a task to be solved to an operational description of how this task should be solved. However, the recognition of these two additional relations allow to better distinguish and separate different reasoning steps in the construction of a task-method structure. Figure 2 shows the additional refinement relations, and their place in the PSM architecture. Of course, the process is recursive, i.e. goals of subtasks of a problem-solving method can again either be refined to weaker goals or directly operationalized by an operational specification etc. In the next section we will illustrate the role of our additional refinement relations in describing and comparing tasks and problem-solving methods in a task-method structure by presenting some excerpts from a collection of PSMs for the task of redesign.

3 Redesign

Redesign is an inherent part of most design processes, but can also be seen as a family of design methods in itself. In contrast to design-from-scratch, redesign starts out with an existing design description and modifies this until it fits the current needs as good as possible. In order to perform redesign it is essential that some form of knowledge is available that allows the adaptation of existing designs. This knowledge is based on the following two principles: 1) minimally change the design, and 2) maximally exploit existing properties of the domain. An underlying assumption of the task of redesign is that the existing design description is "close enough" to fulfill the needs by only limited adaptations. However, what is considered close enough in a specific case depends on the nature of the adaptation knowledge, and on the way different requirements interact.

Redesign can play two different roles in the complete design process: First, redesign can be seen as a subphase of the design process. Here, design is viewed as an iterative process that uses intermediate results as a means of getting a final design description which fulfills the requirements. The task of redesign in this context produces a new temporary design description which is (hopefully) closer to the specification than the former design description. This view is the basis for the Propose-Critique-Modify family of design methods discussed in (Chandrasekaran, 1990). Secondly, redesign can be considered in the context of reuse. Here, redesign starts with a previously constructed design description, and a new set of requirements. The previously constructed design description must now be modified to fulfill the new set of requirements. This view is often considered as part of approaches such as case-based design (see e.g. (Kolodner, 1993; Maher *et al.*, 1995)), when the already retrieved case is adapted to suit the new requirements. Although there are very subtle differences between these two views, in both cases the important issue is to bridge the gap between a set of requirements and a design description. Therefore, both Iterative Redesign and Redesign for Reuse can be captured by a single spectrum of problem-solving methods for redesign.

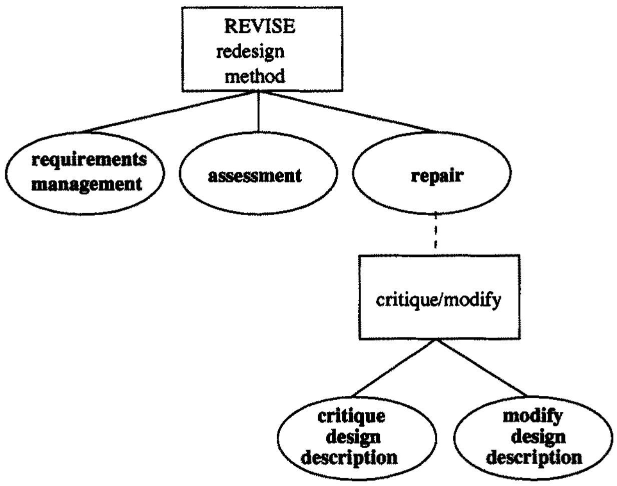

Fig. 3. A partial task-method structure for the redesign task. Rectangles represent methods and ellipses represent tasks. Dashed lines indicate that methods are alternatives for achieving the task goal. Solid lines decompose a method into its subtasks.

In the work described here, redesign is conceived as consisting of two subtasks: requirement management and management of the design description. The latter is further decomposed into assessment and repair. This decomposition of the redesign task (which we will in the remainder of this paper refer to as the "REVISE method") is motivated by work in the REVISE-project. Figure 3 presents a partial task-method structure for the REVISE method.

- *Requirement management.* This subtask is responsible for specification, management, refinement and adaptation of requirements. Some examples of PSMs for this task are discussed in section 3.1.
- *Assessment.* This subtask is responsible for determining the differences between the (properties of the) current design description and the requirements. These differences drive the repair subtask.
- *Repair.* Within this subtask, the design description is adapted such that it will better fit the requirements. Determining which part of the design description will be adapted (critique), and how it will be adapted (modify) are often tightly coupled subtasks in this task. Section 3.2 discusses some examples of PSMs for the subtask of design modification.

The main difference between the REVISE model and other models for (re)design, like Propose-Critique-Modify (Chandrasekaran, 1990), lies in the inclusion of the task of requirement management in the redesign process. In our point of view, explicit management of requirements is essential in non-routine redesign: addition, retraction and modification of the original requirements often forms a major part of any non-routine redesign task. Examples of such non-routine redesign tasks can be found in systems for redesign of compositional architectures (Brazier *et al.*, 1996b), simulation models (Pos & Akkermans, 1996; Pos *et al.*, 1997) and software specifications (Funk & Robertson, 1994).

Many systems that solve redesign-like problems have been described in literature (e.g. (Fischer *et al.*, 1987; Marcus *et al.*, 1987; Daube & Hayes-Roth, 1989; Goel, 1991; Smyth & Keane, 1996; Brazier *et al.*, 1996a)), but when one takes a closer look at the different variants of the redesign task, subtle differences exist that have an impact on how the task can be performed and what kinds of knowledge are involved.

A first source of variation in redesign is the design description. There are several aspects of the design description which are important in the context of redesign. The first of these is the *fixedness* of the structure of the design description; at one end of the spectrum, the structure of the design description can be completely fixed during redesign, and only the values assigned to parameters can be altered. This leads to parametric redesign. On the other end of the spectrum we have situations where changes to the structure of the design description are not limited in any way. Inbetween these extremes, there are cases where a skeleton structure is considered to be fixed but where the specific structure still can be filled in. Another dimension concerning the design description is the *nature of the information* presented in the design description. At one end of this spectrum the design description can purely describe the current status of the design, while at the other end the design description includes a complete plan of design steps resulting in the current design. The latter results in a form of redesign called derivational analogy (Mostow, 1989), while the former is the subject of more standard redesign approaches which directly modify the current design description (e.g. KRITIK (Goel, 1991) and 007 (Pos *et al.*, 1997)).

The requirements put on the design description provide a second source of variation in redesign. Again, there are several dimensions along which the requirements can be classified. The first of these is the *operationality* of requirements. Requirements are operational if their truth can be automatically derived from the design description by some inference method. The question to be considered is whether it is sufficient in an application domain to express needs and desires with operational requirements only, or whether there is a need to express non-operational requirements as well? The latter situation requires more extensive support for requirements management. Software design is a typical example in which the ability to express non-operational requirements is important in supporting the user in requirement specification. Another dimension with respect to the requirements posed on a design description is their *(local or global) nature*. Local requirements are applicable to a single component or parameter, while global requirements specify properties of the complete design. An example of a global requirement is the maximum weight of a device; this weight can not be attributed to a single component but is a function of the combined properties of all the components in the device.

Each redesign process requires some form of knowledge on which adaptations are possible/suitable/useful etc. The nature of the adaptation knowledge is the third source of variation in redesign. Again, there are several dimensions along which this adaptation knowledge can be characterized. The first of these is the *knowledge intensity* of the adaptation knowledge, ranging from purely search based to purely knowledge based approaches. A second dimension is the *generality* of the adaptation knowledge: how widely applicable is the adaptation knowledge. Applicatio-specific fixes are at one end of this spectrum, while very general strategies like 'divide-and-conquer' are located at the opposite end.

requirements	design description	adaptation knowledge
operational/ non-operational	structure fixed/free	search based/ plan based
local/ global	derivation/ design	specific/ generic

Table 1. Dimensions of redesign problems

Table 1 summarizes the dimensions along which redesign problems can differ. A space of redesign problems can be constructed by taking the Cartesian product of the values on each of these dimensions to form a multidimensional problem space for redesign problems. Most of the dimensions mentioned here have been described in the context of design problems other than redesign (Wielinga & Schreiber, 1997; Bernaras, 1994). This is a result of the earlier mentioned position of redesign in the spectrum of methods for design: redesign is both a part of many other design methods, like case-based design, and an umbrella for many different techniques, like parametric (re)design and configuration (re)design.

3.1 Requirement management

The subtask of requirement management is responsible for specification, management, refinement and adaptation of requirements. This task has as its input the current model, and as its output it produces an adequate set of requirements, suitable for assessment of the design description. Requirement management as a separate task is based on the observation that in general a design problem is often initiated by a statement of *needs and desires* (Bernaras, 1994). These, sometimes quite vague, needs and desires are to be interpreted and operationalized into a set of requirements suitable for automated assessment. This corresponds to the task of requirements engineering in software design (Wieringa, 1996). Figure 4 presents a partial task-method structure for the requirement management task. We identified two methods for this task: ask-user-operational-requirements or semi-automated-requirement-management. In many redesign systems, the task of requirements specification is put completely in the hands of the user, and no automated support is provided. This corresponds to the ask-user-operational-requirements PSM. On the other hand, a small number of redesign systems (e.g. (Brazier *et al.*, 1996b; Pos *et al.*, 1997; Reubenstein & Waters, 1991)) explicitly incorporate some form of semi-automated requirement management to ensure that the resulting set of requirements is adequate for further processing in the assessment task.

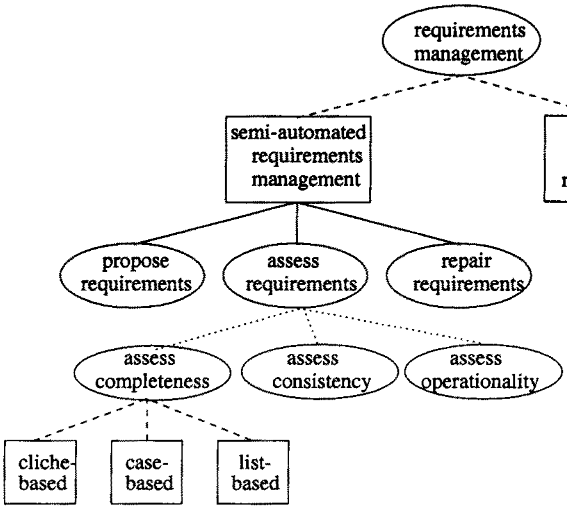

Fig. 4. A partial task-method structure for the requirement management task. Rectangles represent methods and ellipses represent tasks. Dashed lines indicate that methods are alternatives for achieving the task goal. Solid lines decompose a method into its subtasks. Dotted lines indicate competence refinement of tasks or methods. Not all methods are decomposed into subtasks and primitive inferences.

A necessary ontological assumption for including any form of semi-automated requirement specification in a (re)design system is that knowledge on how requirements relate to each other is available in the application domain. The extent and nature of this knowledge determines which form(s) of requirement management are feasible. Requirement management is primarily useful when the set of possible requirements to be posed to the (re)design system is potentially large, when the number of requirements to be simultaneously satisfied may become large or when the need for expressing non-operational requirements arises in a (re)design system due to the complexity of the domain involved. In cases where the set of possible requirements is small and requirements are not apparently interacting, requirement management is usually not necessary.

The PSM 'semi-automated-requirements-management' decomposes the task of requirement management in three subtasks: propose-requirements, assess-requirements and repair-requirements. This decomposition corresponds to the family of methods Chandrasekaran (Chandrasekaran, 1990) calls Propose-Critique-Modify. For the current example, we will focus on the assess-requirements subtask in requirement management.

Requirement assessment The assess-requirements task takes as input a set of requirements and produces a datum DA that states whether the requirements are adequate or not.

task name:	requirement assessment
goal:	KNOWN(ADEQUATE(R)) $\vee$ KNOWN(INADEQUATE(R))
input role:	R: set of requirements
output role:	DA: KNOWN(ADEQUATE(R)) $\vee DA$: KNOWN(INADEQUATE(R))

By carefully looking at requirement management in different (re)design systems we distinguish three different *competence refinements* of this requirement-assessment task to more specialized tasks. Each of these refinements requires different teleological assumptions, providing different restrictions on the *goal* of the requirement assessment task: KNOWN(ADEQUATE(R)) $\vee$ KNOWN(INADEQUATE(R)). The input and output roles stay the same: in each refinement the task takes as input a set of requirements R and delivers as output a statement on some aspect of the adequacy or inadequacy of R.

Assess completeness A possible way in which the competence of the assess requirement tasks can be refined is to refine the notion of 'adequacy' to the more specialized notion of 'completeness'. The goal of the thus refined assess completeness task then becomes: KNOWN(COMPLETE(R)) $\vee$ KNOWN(INCOMPLETE(R)). This is an example of the introduction of additional teleological assumptions: the output of the assess requirements task is restricted from a statement about the *adequacy* of the set of requirements to a more specialized statement about the *completeness* of the set of requirements.

The completeness of a set of requirements can only be assessed with respect to a specific problem or a specific set of problems. Problem-solving methods for assessing incompleteness of a requirement management in general use problem-specific knowledge to decide whether a requirement management is complete with respect to the problem(s) posed. This information can e.g. be represented the form of common forms (cliches) for different problems (Reubenstein & Waters, 1991; Pos *et al.*, 1997), in the form of cases (Maher & Balacandran, 1994) or in the form of a predefined list of requirements to be specified (Brazier *et al.*, 1996c). A necessary ontological assumption for each of these PSMs is that knowledge on when a set of requirements is supposed to be complete is present in the domain knowledge.

Other possible refinements of the requirement assessment task are checking whether the set of requirements is internally consistent (assess consistency), or checking whether each requirement is fully operationalized (assess operationality), i.e. can automatically be inferred from the design description by some inference procedure. This does not mean that these refinements are mutually exclusive: an actual redesign system can include one or more refinements.

All three tasks refinements are pure *competence refinements* of the original task goal 'assess requirements': they do not make any claims on how the task goal should be operationalized in terms of subtasks, but only on the way the task goal should be further *specified.* This allows us to view the task 'requirement assessment' as an abstract description for a family of related tasks, each of which tests a different aspect of the 'adequacy' of a set of requirements. Our purpose in constructing the notion of task refinements has been to better distinguish different steps, corresponding to different design decisions, in the construction and classification of problem-solving methods. The next step is to either refine the current task further to an even more specialized task, or to match sufficiently refined task goals to the functional description of (a family of) PSMs which might be able to achieve the goal of the refined task.

3.2 Design modification

The task of design modification modifies part of the current design description to solve one or more discrepancies between the set of requirements and the current design description. Figure 5 presents a partial task decomposition of this task. This task has as input (a part of) the design description and delivers as output a modified version of this part that is hopefully more suitable in the new situation. Three general families of problem-solving methods for this task are ***substitution***, ***transformation*** and ***generation*** (Maher *et al.*, 1995). ***Substitution*** methods substitute a part of the old design description with a new part more suitable for the new situation. *Transformation* methods are used to transform an old solution into one that will (hopefully) work in the new situation. *Generative* methods re-enact (part of) the reasoning trace to modify the design description. FIRST (Daube & Hayes-Roth, 1989), a case-based system for redesign of mechanical systems, and COBRA (Finn *et al.*, 1992), a case-based system for redesign of heat-transfer models, provide examples of the latter approach. Both systems retrieve 'redesign plans' from a case-base and transfer these plans to the new problem at hand.

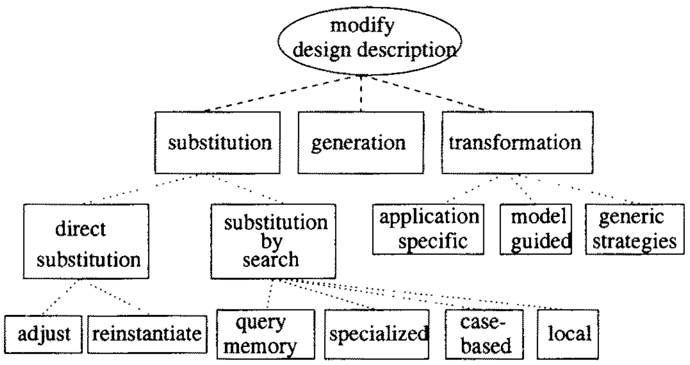

Fig. 5. A partial task decomposition of the modify-design-description task.

Several dimensions of redesign (Table 1) play a role as ontological assumptions in this choice between alternative (families of) PSMs: whether the structure of the design is supposed to be fixed, and whether the reasoning trace or the end product of design is used for redesign. Generative methods make use of a reasoning trace, while substitution and transformation methods require only the end product of design in the form of a design description. Substitution and transformation methods mainly differ in their assumptions on system structure: substitution methods assume that the system structure is fixed, while transformation methods assume that the structure of the design is adaptable.

Substitution methods can be further specialized in direct substitution methods and substitution by search. This distinction is based on the dimension of redesign concerning the *plan based versus search based* nature of adaptation knowledge. This dimension defines ontological assumptions on the type of additional knowledge necessary to make the PSMs work: substitution by search requires some sort of database in which substitution elements can be found, while direct substitution methods require procedural knowledge on how to decide on a new value for a part of the old design description. Search-based substitution methods can be further specialized by considering the *additional knowledge* necessary for searching: query memory (Hinrichs & Kolodner, 1991) requires information on *what* to look for, in the form of a partial description of the item searched for. Local search, which can e.g. be found in PLEXUS (Alterman, 1986), requires instructions on *where* to search for alternatives. Specialized search, which can e.g. be found in the SWALE system (Schank & Leake, 1989), requires instructions on *how* to find an alternative. Case based substitution, which can e.g. be found in the JULIA system (Hinrichs & Kolodner, 1991), requires instructions on how to find a *similar* case which might suggest an appropriate alternative.

Transformation methods can be further specialized by considering the *general or specific* nature of the modification knowledge used for transformation: generic strategies like 'divide-and-conquer', which can e.g. be found in JULIA (Hinrichs & Kolodner, 1991), form one end of this spectrum, while application-specific heuristics, which can e.g. be found in the VT-system for elevator design (Marcus *et al.*, 1987), form the other end of the spectrum. Systems like KRITIK (Goel, 1991) and 007 (Pos *et al.*, 1997) fall somewhere in between: the repair plans used in these systems are general enough to be applied to many different problems, but not as generic as general strategies like 'divide and conquer'.

All the method specializations discussed above are *competence refinements*: they only make claims on the functional specification of (a family of) problem-solving methods, and not on how these PSMs should be operationalized in terms of subtasks and control.Method refinements represent *intermediate* decisions in the classification of PSMs. Our purpose in constructing these intermediate relations has been to better distinguish different design decisions as separate steps in the classification of PSMs, based on the dimension underlying each design decision. They allow us to describe a *family* of related PSMs by means of an abstract functional specification common to all its members. In the original PSM framework, these intermediate, seperate design decisions would either have be compressed into direct relations between the task and the bottom-level PSMs (e.g., one way to satisfy the task 'modify design description' is to directly use the bottom-level PSM 'substitution by local search'), or an intermediate operationalization would have to be constructed for each family of PSMs in the refinement decomposition.

The task decomposition depicted in Figure 5 covers the same set of methods as the classification of adaptation methods and strategies in case-based reasoning presented in (Kolodner, 1993). This lends further support to our claim that redesign methods form a part of several more general design paradigms, of which case-based reasoning is one.

4 Conclusions & Related Work

Redesign is an inherent part of most design processes, but can also be seen as a family of design methods in itself. In the latter point of view, redesign is a family of problem-solving methods for which two questions need to be answered:"what is common to redesign?" and "which dimensions are involved in characterizing redesign?". The family of methods called redesign can be characterized by a number of common principles: redesign starts with an existing design description, and it requires some form of knowledge that allows the adaptation of designs. This knowledge is based on two underlying principles: (1) minimizing the changes being made to the design description and (2) maximizing the exploit of known properties of the design description. These characteristics can be used to differentiate this family of methods from other design methods. In order to compare different (families of) problem-solving methods for the same task we distinguished a number of *dimensions* along which redesign methods can differ: the nature of the *design description*, the nature of the *requirements* and the nature of the *adaptation knowledge* are the three main sources of variation in the family of problem-solving methods called redesign.

We also presented a general model of redesign developed in the REVISE-project. The main tasks in this general model are requirement management, artifact assessment and artifact repair. The task of requirement management has not often been modeled in other (re)design models, but is in our view essential in non-routine (re)design. Requirement management in design corresponds to the task of requirements engineering in software design (Wieringa, 1996). Each of these redesign subtasks can be achieved by one or more (families of) PSMs. In this paper we selected two subtasks, requirement management and artifact modification, and described task-structures for these subtasks in somewhat more detail. For the other redesign subtasks similar task-structures have been developed.

In developing a knowledge based system for a redesign task a large number of choices and decisions are made. In order to express all these decisions as separate reasoning steps in a task-method structure, the current notion of possible relations between tasks and methods in a PSM architecture needs to be extended. Notions of *task refinement* and *method refinement* are introduced to represent intermediate decisions in a task-method structure, in which the competence of (a family of related) tasks or methods is refined without directly paying attention to the operationalization in terms of subtasks and control. Explicit representation of this kind of abstract, intermediate reasoning steps helps to make and represent decisions in a task-method structure in a more piecemeal fashion.

The characterization of tasks and methods for redesign presented in this paper is informal. We think that this type of characterization is useful in the early stages of KBS design, where problem-solving methods are constructed and selected to construct the skeleton design model for a knowledge based system. In this early stage of KBS design, common sense descriptions of problem-solving methods and their underlying knowledge requirements (in terms of teleological and ontological assumptions made by each problem-solving method) will often be used in advance of any formal work. In a later stage, *adapters* as proposed in (Fensel, 1997), could be used to implement and formalize the additional refinement relations proposed here in an informal context.

An alternative approach to specifying and selecting PSMs is presented in (ten Teije *et al.*, 1996). Here, a family of PSMs for diagnosis tasks is represented by a skeletal functional description, with a number of subfunctions. Values for these are chosen from a fixed set to obtain a specific diagnostic function. Parametric design can then be used to construct a specific method functionality. However to construct such a skeleton and appropiate values requires a well-understood formalisation of different methods and their properties, which does not yet exist for redesign methods.

The extended PSM framework described in this paper has been used to describe relevant choices in selecting problem-solving methods for a specific redesign system in the domain of computational engineering models, called 007 (Pos *et al.*, 1997). In the near future, we plan to use this framework to describe other redesign systems, starting with the redesign systems which are at the moment being developed by the other participants in the REVISE-project. These systems focus on redesign of compositional architectures (Brazier *et al.*, 1996b) and redesign of control knowledge in knowledge based systems (Straatman, 1995), respectively.

Acknowledgments

This paper has been significantly influenced by many discussions in the REVISE-project. We would like to thank all other REVISE participants for their input. We would also like to thank Tim Menzies for his extensive comments on this paper.

References

AKKERMANS, H., WIELINGA, B., & SCHREIBER, G. (1994). Steps in constructing problem solving methods. In Gaines, B. R. & Musen, M., editors, *Proceedings of the 8th International Knowledge Acquisition Workshop (KAW'94)*, volume 2, pages 29.1–29.21, Banff, Alberta. University of Calgary, SRDG Publications.

ALTERMAN, R. (1986). An adaptive planner. In Kehler, T., Rosenschein, S., Filman, R., & Patel-Schneider, P. F., editors, *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)*, pages 65–69, Philadelphia, PA. Morgan Kaufman Publishers, Inc.

BENJAMINS, V. R. (1993). *Problem solving methods for diagnosis*. Ph. D. Thesis, University of Amsterdam.

BENJAMINS, V. R., FENSEL, D., & STRAATMAN, R. (1996). Assumptions of problem-solving methods and their role in knowledge engineering. In Wahlster, W., editor, *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, pages 408–412, Budapest, Hungary. John Wiley and Sons.

BERNARAS, A. (1994). Problem-oriented and task-oriented models of design in the COMMONKADS framework. In Gero, J. S. & Sudweeks, F., editors, *Artificial Intelligence in Design '94*. Dordrecht, the Netherlands, Kluwer Academic Publishers.

BRAZIER, F. M. T., TREUR, J., & WIJNGAARDS, N. J. E. (1996a). Interaction with experts: the role of a shared task model. In Wahlster, W., editor, *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, pages 241–245, Budapest, Hungary. Wiley and Sons.

BRAZIER, F. M. T., VAN LANGEN, P. H. G., TREUR, J., & WIJNGAARDS, N. J. E. (1996b). Redesign and reuse in compositional knowledge-based systems. *Knowledge Based Systems, Special Issue on Models and Techniques for Reuse of Designs*, 9(2):105–119.

BRAZIER, F. M. T., VAN LANGEN, P. H. G., TREUR, J., WIJNGAARDS, N. J. E., & WILLEMS, M. (1996c). Modelling an elevator design task in DESIRE: the VT example. *International Journal of Human-Computer Studies*, 46:469–520.

CHANDRASEKARAN, B. (1988). Generic tasks as building blocks for knowledge-based systems: the diagnosis and routine design examples. *The Knowledge Engineering Review*, 3:183–210.

CHANDRASEKARAN, B. (1990). Design problem solving: a task analysis. *AI Magazine*, 11(4):59–71.

DAUBE, F. & HAYES-ROTH, B. (1989). A case-based mechanical redesign system. In Shridharan, N. S., editor, *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, Detroit, Michigan. Morgan Kaufman Publishers, Inc.

ELDONK, S. J. M., ALBERTS, L., BAKKER, R., F.DIKKER, & WOGNUM, P. (1996). Redesign of technical systems. *Knowledge-Based Systems, Special Issue on Models and Techniques for Reuse of Designs*, 9(2):93–104.

FENSEL, D. (1997). The tower-of-adapters method for developing and reusing problem-solving methods. In *Proceedings of the European Knowledge Acquisition Workshop (EKAW-97)*. Springer-Verlag. Lecture Notes in Artificial Intelligence (LNAI).

FINN, D. P., GRIMSON, J. B., & HARTY, N. M. (1992). An intelligent modelling assistant for preliminary analysis in design. In Gero, J., editor, *Artificial intelligence in Design (AID'92)*, pages 579–596. Dordrecht, Kluwer Academic Publishers.

FISCHER, G., LEMKE, A. C., & RATHKE, C. (1987). From design to redesign. In *Proceedings of the 9th International Conference on Software Engineering*, pages 369–376, Washington, D.C. IEEE Computer Society Press.

FUNK, P. J. & ROBERTSON, D. (1994). Case-based support for the design of dynamic system requirements. In *Proceedings of the 2nd European Workshop on Advances in Case-Based Reasoning (EWCBR-94)*, pages 211–225, Chantilly, France.

GENNARI, J., TU, S., ROSENFLUH, T., & MUSEN, M. (1994). Mapping domains to methods in support of reuse. *International Journal of Human-Computer Studies*, 41:399–424.

GOEL, A. K. (1991). A model-based approach to case adaptation. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society (CogSci'91)*, pages 143–148, Chicago, Illinois.

HINRICHS, T. & KOLODNER, J. (1991). The roles of adaptation in case-based design. In Dean, T. & McKeown, K., editors, *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI-91)*, pages 28–33. AAAI Press / The MIT Press.

KOLODNER, J. (1993). *Case-Based Reasoning*. Morgan Kaufman Publishers, Inc.

MAHER, M. & BALACANDRAN, B. (1994). Flexible retrieval strategies for case-based design. In Gero, J. & Sudweeks, F., editors, *Artificial Intelligence in Design '94*. Dordrecht, Kluwer Academic Publishers.

MAHER, M. L., BALACHANDRAN, M. B., & ZHANG, D. M. (1995). *Case-based reasoning in design*. Hove, UK, Lawrence Erlbaum Associates.

MARCUS, S., STOUT, J., & MCDERMOTT, J. (1987). VT: an expert elevator designer that uses knowledge-based backtracking. *AI Magazine*, 8(4):39–58.

MCDERMOTT, J. (1988). Preliminary steps toward a taxonomy of problem-solving methods. In Marcus, S., editor, *Automating Knowledge Acquisition for Expert Systems*, pages 225–255. Boston, Kluwer.

MOSTOW, J. (1989). Design by derivational analogy: Issues in the automated replay of design plans. *Artificial Intelligence*, 40:119–184.

O'HARA, K. & SHADBOLT, N. (1993). Locating generic tasks. *Knowledge Acquisition*, 5:449–481.

POS, A. & AKKERMANS, J. M. (1996). 007: A system for automated model revision. In Javar, A., Lehmann, A., & Molnar, I., editors, *Proceedings of the 10th European Simulation Multiconference (ESM'96)*, pages 50–54, Budapest, Hungary. SCS.

POS, A., AKKERMANS, J. M., & TOP, J. L. (1997). Automated model revision. *IEEE Expert*. In Press.

REUBENSTEIN, H. B. & WATERS, R. C. (1991). The requirements apprentice: Automated assistance for requirements acquisition. *IEEE Transactions on Software Engineering*, 17(3):226–240.

SCHANK, R. C. & LEAKE, D. B. (1989). Creativity and learning in a case-based explainer. *Artificial Intelligence*, 40:353–385.

SMYTH, B. & KEANE, M. T. (1996). Using adaptation knowledge to retrieve and adapt design cases. *Knowledge-based Systems, Special Issue on Models and Techniques for Reuse of Designs*, 9(2):127–136.

STEELS, L. (1990). Components of expertise. *AI Magazine*, 11 (2):28–49.

STRAATMAN, R. (1995). Learning control knowledge in models of expertise. In Fensel, D., editor, *Proceedings of the ECML-95 Workshop on Knowledge Level Modeling and Machnine Learning*, pages I.2.1–I.2.13, Heraklion, Greece.

TEN TEIJE, A., VAN HARMELEN, F., SCHREIBER, A. T., & WIELINGA, B. J. (1996). Construction of problem-solving methods as parametric design. In Gaines, B. R. & Musen, M. A., editors, *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, volume 1, pages 12.1–12.21, Alberta, Canada. SRDG Publications, University of Calgary. track: Shareable and reusable problem-solving methods.

WIELINGA, B. J. & SCHREIBER, A. T. (1997). Configuration-design problem solving. *IEEE Expert (in press)*.

WIELINGA, B. J., VELDE, W. V. D., SCHREIBER, A. T., & AKKERMANS, J. M. (1993). Towards a unification of knowledge modeling approaches. In David, J. M., Krivine, J. P., & Simmons, R., editors, *Second-generation expert systems*, chapter 14, pages 299–335. Berlin, Springer-Verlag.

WIERINGA, R. J. (1996). *Requirements Engineering: Frameworks for Understanding*. Chicester, England, John Wiley and Sons.