Using reflection techniques for flexible problem solving  
(with examples from diagnosis)

Annette ten Teije,  
Department of Social Science Informatics,  
University of Amsterdam,  
Roetersstraat 15, 1018 WB Amsterdam, The Netherlands,  
anette@swi.psy.uva.nl

Frank van Harmelen,  
AI-group, Department of Mathematics and Computer Science.  
Free University Amsterdam,  
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands,  
frankh@cs.vu.nl

Abstract

Flexible problem solving consists of the dynamic selection and configuration of problem solving methods for a particular problem type, depending on the particular problem and the goal of problem solving. In this paper, we propose an architecture that supports such flexible problem solving automatically. For this purpose, problem solving methods are described in a uniform way, by an abstract model of components, which together define the functionality of the methods. Such an abstract model is used for dynamic selection and configuration of the problem solving methods. The proposed architecture for flexible problem solving consists of well known reflection techniques: two object-meta relations, a definable naming mechanism and the axiomhood and theoremhood reflection rules. We have succeeded in using standard meta-architecture techniques to enable flexible problem solving.

1 Introduction

The literature on Knowledge Engineering of the past decade has identified a number of different problem types (originally proposed in [13], later refined by [5] and many others). Examples of such problem types are diagnosis, design, monitoring, assessment, classification. For each of the various problem types, researchers have identified a number of problem solving methods which are methods that can be employed to solve a problem of the particular type. For
example, for design problems, we have generate-and-test or propose-critique-modify [4] as two of the many methods. Similarly, diagnosis problems can be solved by such diverse methods as consistency-based diagnosis, hierarchical diagnosis, abduction (see [6] for a survey).

A central question is then “Which problem solving method (PSM) is optimal for a given problem type?” In section 2, we argue that such a single optimal PSM does not exist for most problem types. In general, the choice of an appropriate PSM will depend on the goal of problem solving, and on characteristics of the specific input (knowledge and data). As a result, PSMs must be selected or configured. In the former case, methods are selected from a predefined set, while in the latter case parts of existing methods or newly defined parts are combined to construct a new method. However, such a selected or configured method might fail in realising the intended goal. In this case, we have to iterate the selection and configuration process, using information about the failure of the previous attempt. For example, we might want to use another method which is more appropriate in the current context. This can result in a new selection or configuration process, that leads to another more preferable method in the current context. We use the term “flexible problem solving” for this process of selection, configuration and strategy adaptation. Ideally, we would like to perform such flexible problem solving automatically.

In this paper we present an architecture that supports such flexible problem solving. This architecture employs meta-reasoning techniques to represent, knowledge about: (i) problem types, (ii) PSMs for a problem type, and (iii) several types of knowledge of a particular domain, that are required for PSMs.

We can draw three general lessons from this paper:

(i) A general way to express the functionality of PSMs schematically, under the assumption that the application domain is described in a logical theory.
(ii) A general way to select and configure functionality of PSMs using the schematic form of point (1) above.
(iii) Simple reflection techniques are sufficient for realising (1) and (2).

These general lessons are based on our study in the diagnosis domain of diagnostic problem solving methods.

This paper is organized as follows. We first give a motivation for flexible problem solving in section 2. Section 3 contains an analysis of the various knowledge types involved in flexible problem solving. Section 4 gives a global overview of the architecture for flexible problem solving. Section 5 discusses the architecture in more detail. The examples for illustrating the ideas are all taken from the diagnostic domain. Section 6 discusses related work, conclusions, and future work.
2 Motivation for Flexible Problem Solving

In this section we motivate (1) why we do not look for the best problem solving method and (2) why we work on automated flexible problem solving.

In the literature [3] many problem solving methods (PSMs) are described that could be used for a particular problem type such as diagnosis, monitoring, or design. When trying to select a method from this set of methods for a particular problem it becomes clear that most methods are only useful in a particular setting. Therefore a best PSM does not exist in general. We illustrate the absence of one overall best PSM by two reasons.

The first reason is that the selection or configuration of a method depends on the goal we want to achieve. For example, goals of problem solving can differ in desired amount of detail of the computed solutions. A concrete example in diagnosis is: when someone is asking for an ambulance [18] we need a diagnostic method that performs an abstract way of diagnosis, namely the method should diagnose whether we have a possible urgent case or not. A more detailed diagnosis is not possible because the available data is only the information that is given by the caller. Later in the hospital, for prescribing a therapy or medicines the doctor needs a more detailed diagnosis. Therefore there is a need for a method that performs diagnosis in much more detail. Such a detail diagnosis is possible because more knowledge is available about the complaints and the patient, such as results of blood tests etc. Another example of the influence of the goal on the appropriateness of a PSM is the desired amount of certainty of the solutions. For example, if someone needs radiation for treating cancer then the diagnosis has to be very safe, because a doctor would not give a patient needless radiation. At other times we need a very safe diagnosis in the sense that every possible cause has to be included in the diagnosis. For example, if there are problems in a nuclear plant, then this can have disastrous consequences. In such a case all the possible causes need to be checked and it does not matter that the checks are possibly superfluous. Therefore there is a need for a method that determines the differential diagnosis in stead of a possible diagnosis. Provan and Poole also emphasized in [19] the importance of the goal of doing diagnosis.

The second reason that the best PSM for a given problem type does not exist in general is that method selection and configuration depends on the particular input (=knowledge and data). An example of depending on knowledge in the diagnosis domain is: if there is only knowledge about correct behaviour of the system, then a pure abductive method is not useful, but a pure consistency based method is. Other examples of depending on characteristics of the input is that a method (e.g. GDE [9]) is applicable only when a structure model and a behaviour model is available. Another example is the method in [10]. This
method requires an acyclic model as input.

Now that we have motivated why there is a need for several PSMs best PSM, we continue with our motivation for why we work on automated flexible problem solving.

A consequence of the necessary existence of several PSMs instead of one is the need for a process of selecting and configuring appropriate methods. When solving a particular problem, questions arise such as: “Which PSM should be used?” and possibly “What should be done if the PSM does not give the desired solution?” The answers to these questions are guided by the goals that need to be realised. Examples of goals in diagnosis are:

- the cause must be part of the diagnosis; in other words the requirement of a differential diagnosis.
- the PSM must give not more than one solution
- the diagnosis must be at the same abstraction level as the most detailed observation.

The goal of flexible problem solving is to compute the most appropriate solution. Such a solution is based on:

- the given input: goal, knowledge, data
- characteristics of the input (often called assumptions [1])
- the available knowledge of the problem type, such as available PSMs
- characteristics of these PSMs.

Such a flexible problem solving system should be able to select or configure the most appropriate PSM for the current problem solving goal and the given input. Dependent on the goal, one would like to tune the method more on the specific domain (knowledge base) or even on the specific case (data).

Such a flexible problem solving system should also be able to reflect on this process and initiate new strategies. For example, when no applicable method is available, the system must be able to adapt the particular problem or the given goal. An example of such adaptation of the problem is if a method cannot solve a particular problem because of missing data or knowledge, then the system should be able to assume some information in such a way that it can solve the problem. In other words the system adapts the under-specified problem, in such a way that it can be solved by the method. An example in diagnosis is that the system assumes some absent observations for computing a diagnosis. This resembles early work of Jansweijer [14]. Also useful are adaptations of a method in case that the current method does not give the desired result. For example, suppose that the system selects the CHECK diagnostic method [7], but it turns out that the method gives too many solutions. Such observation results in an additional goal for the desired diagnostic
method, namely a method that gives less solutions than CHECK. The system could adapt its strategy by choosing CHECK as its basis, but using a stronger minimality criterion. In other words a new method configuration process is initiated and results in a method with a stronger minimality. See [25], for more examples of adaptations of inputs and methods.

We would like to perform flexible problem solving automatically, because this process depends often on the goal of problem solving and the specific input (data and knowledge), and therefore it is not possible to determine this statically.

Summarizing, we argue that a single best PSM for a problem type does not exist. We study theories to solve a particular problem in the most appropriate way. Such theories use existing PSMs flexibly (select, configure, adapt), and exploit the goal of problem solving, the characteristics of the available PSMs and the characteristics of the current input problem.

3 Knowledge Analysis for Flexible Problem Solving

This section contains an analysis of the different knowledge types that are required for flexible problem solving. We distinguish three types of knowledge in section 3.1. We discuss in section 3.2 the relations between these three types of knowledge. In the last section we end up with the requirements for the architecture based on this knowledge analysis.

3.1 Three Knowledge Types

The three types of knowledge that we distinguish for performing flexible problem solving are: knowledge that describes the applications, knowledge that describes the problem solving methods and knowledge that describes how we can use these methods flexibly for problem solving. We describe these knowledge types below.

**Application description (APPL):** The first type of knowledge is the knowledge and data that is used by a problem solving method. We denote this knowledge by APPL. A part of APPL is case independent. We use this knowledge each time we solve a problem. In the diagnosis domain this holds for the models of expected behaviour of the systems such as knowledge of a causal model “if the battery is flat, the lights do not work”. This knowledge can be used in every diagnosis session and does not depend on a specific problem.
APPL also contains case dependent knowledge. An example in the diagnostic
domain is the observed behaviour such as “the high temperature indicator is
red”. The high temperature indicator is not always red, in contrast with the
causal knowledge of the flat battery.

Summarizing, the category of application description APPL contains domain
knowledge and data that is used by a problem solving method. A part of the
knowledge is used for every application of a method, another part depends on
a specific case.

Methods description (METH) The second type of knowledge describes
the problem solving methods, themselves. For example, CHECK [7], GDE [9],
GDE+ [21] are diagnostic PSMs: methods for diagnosing which give a solution
for a diagnostic problem using knowledge of the expected and observed
behaviour. Our intention is to selection and configuration methods and, if re-
quired to adapt the strategy of problem solving by a new select and configure
process or an adaptation of the input.

In this paper, we will consider the functionality of methods, often called the
input/output behaviour or the declarative description, and base our choice of
a method only on this aspect. However, there are many operationalizations of
a particular functionality. Such operationalization aspects play an important
role, because the efficiency is often an criterion that determines the appro-
priateness of the method. Intuitively, one should first make a choice of the
functionality “what to compute” and then a choice “how to compute”. How-
ever, sometimes it is better to give up part of the functionality and to get
a more efficient operationalization. This illustrates the interaction between
the choices of functionality and operationalization. We concentrate only on
the functionality of PSMs, because this is an essential part of the PSM. We
assume the ideal word in which for every functionality there exists an appro-
priate efficient algorithm and try to find out when to use which input/output
relation for a given problem. Such considerations have to play a role in every
form of flexible problem solving. However, we do realise that (1) other crite-
ria play a role in flexible diagnosis (specially the efficiency criterion), and (2)
that these criteria interact with the functionality criterion. A consequence of
our approach is that we consider methods with the same functionality as the
same methods, but these methods can be very different on other aspects (e.g.
algorithm aspects) than functionality.

A central idea of this paper is that the functionality of a PSM can be de-
scribed by several components, and that it is possible to develop a framework
for describing the functionality of PSMs. In particular, we have developed
a general framework for expressing the functionality of diagnostic methods
[22]. The claim in [22] is that the functionality of many if not all diagnostic
PSMs can be described by choosing the appropriate value for each of the six components. In this paper we use a very simple version of such a component-oriented description of diagnosis methods. We distinguish three components for the functionality:

(i) the explanation component: the specification of what it means for observables to be covered: Using the behaviour model and the context the cover relation explains the covered observables. The input of this functional component Cover is the behaviour model, the context and the observables that must be covered. The output is a covering explanation.

(ii) the not-contradicting component: The specification of what it means for observables not to be contradicted. Using the behaviour model, the observables that must not be contradicted and the context this relation returns all non-contradicting explanations. The input of this functional component NotContra is the behaviour model, the context and the non-contradictory observables. The output is a non-contradicting explanation.

(iii) a selection component: the specification of a minimal explanation, this is the criterion that determines which explanation is selected among all possible explanations. The input of this functional component Select is the set of possible explanations. The output is the set of selected explanations.

Summarizing, the category of description of methods METH contains the functionality of methods.

**Knowledge for solving a problem flexibly (FLEXPROB):** We are aiming at selecting, configuring and adapting problem solving methods, therefore we have to reason about methods. A general scheme for representing all methods of a problem type makes it possible to reason about methods. Such a schema expresses which components (e.g. explanation relation in diagnosis) we distinguish in a method and how these components can be put together to form the functionality of a complete method. Such a general schema is used to express different PSMs of a problem type in a uniform way, which is necessary (but not sufficient) for automatically reasoning about PSMs.

In diagnosis we could use a very simple general schema for diagnosis as: A diagnosis is selected by a selection criterion among all possible expressions that satisfy the explanation relation and the not-contradicting relation with the given observations. Such _general scheme_ for the functionality of PSM is the first ingredient of this knowledge type.

For reasoning about methods we also need relations and properties of methods. For example, one method gives a subset of the solutions of another method. Such relations are often based on the relations between parts of the methods.
For example, in diagnosis we use the relation between two distinct minimalities, such as one minimality is stronger than another one, for relating methods, such as one method give less solutions than another method. The second ingredient of this knowledge type are these relations and properties of methods or components of methods.

The last ingredient is the knowledge which captures when we should use a component or method, but also when we should adapt the problem solving strategy, such as a change of the configured method. This is knowledge that uses the general scheme for representing methods and the relations and properties of components and methods for selecting, configuring method and adapting the strategy. We call this the strategy knowledge of flexible problem solving.

In summary, this category contains a general schema for problem solving methods. This schema enables us to compare and relate methods to each other, because they are all expressed in the same way. The other kinds of knowledge in this category are the relations and properties of methods or method parts, and how we use them for solving a problem flexibly.

Summarizing, in this section, we have distinguished three knowledge types:

(i) APPL: knowledge that is related to a domain and is used by a problem solving method.
(ii) METH: descriptions of problem solving methods and their components.
(iii) FLEXPROB: knowledge for problem solving flexibly.

3.2 Relations between Knowledge Types

In this section we describe the relations among APPL, METH and FLEXPROB. There are three important relations between APPL and METH.

The first is that the methods described in METH state relations between the objects of the application description APPL. These relations are of a particular kind, namely they describe functional input/output behaviour of a complete method or of part of a method. In the diagnosis domain the diagnostic method relates the observed and the expected behaviour to possible diagnoses.

The second relation is the abstraction relation. METH abstracts from specific instances of APPL. In other words the description of a method is generic with respect to the input and output. METH uses only the structure of APPL. This structure can be substituted by parts of a particular application description. This means that METH abstracts from specific instances of APPL. For example the APPL in diagnosis refers to a specific symptom, such as “pain in chest”,
but the diagnostic method only knows about the general class of symptoms without referring to specific instances.

The third relation is the role-assignment relation: METH assigns specific roles to each of the objects in APPL. These roles are based on how these objects are used by the problem solving method. For instance, in diagnosis, some knowledge is used in the role of causal knowledge ("empty battery causes no light") and other knowledge is used in the role of structural knowledge ("contact point is part of battery").

These three kinds of relations between APPL and METH also exist between METH and FLEXPROB. FLEXPROB states relations between the objects of METH: FLEXPROB describes properties of the definitions of methods and parts of methods. The component definitions can be used for expressing something about their coherence. They can also be used for saying something about the use of a complete method. Furthermore, FLEXPROB abstracts from specific method descriptions, and uses only the general structure of a method definition. FLEXPROB abstracts from specific instances of METH, since the description of a diagnostic method is generic. Finally, FLEXPROB states which role is played by the different parts of a method (e.g. selection criterion or explanation relation).

3.3 Requirements for the Architecture

The above knowledge analysis results in four requirements for the architecture.

The first requirement is based on the existence of the three knowledge types. If we represent these knowledge types explicitly and separately, it becomes possible to reuse them. We can use often the same METH for several APPL; for example we can use a diagnostic method for diagnosing several systems (e.g. a car, a digital circuit). Similarly we can use several methods for the same application. One method could give a very precise diagnosis, another method a more global one. The same holds between METH and FLEXPROB. METH is the base for several methods for several theories for flexible problem solving. METH describes complete methods which can be used for selecting a method. The possible components of a diagnostic method which can be used in a configuration theory. METH is the base for several theories for flexible problem solving, such as a configuration theory and an approximating theory. In other words we reuse METH in these theories.

The three relations among the knowledge types give us three further requirements for the architecture. First, one knowledge type must be able to express relations among objects of another knowledge type, secondly one knowledge type must be able to abstract from the instances of another knowledge type,
and thirdly, one knowledge type must be able to express the roles that are played by the objects of another knowledge type.

4 Meta-Architecture

In this section we motivate why we use a meta-architecture to realise flexible problem solving, and we describe the main meta-representation techniques that we use.

4.1 Motivation for a Meta-Architecture

The characteristic of an object-meta architecture is that these architectures exploit the difference between knowledge and knowledge about this knowledge (meta-knowledge). As we mentioned in the previous section, METH is knowledge about APPL, and FLEXPROB is knowledge about METH. Therefore, we can see two object-meta relations (see figure 1).

Meta-architectures have been developed using several kinds of formalisms, such as functional languages, procedural languages, object-oriented languages or using a logical language [16] and [23] for surveys. The first order logic based meta-systems used first order logic (FOL) as object-language and as meta-language. The idea of these FOL based systems is that such systems enable us to simulate second order logic by using only first order logic in the object- and meta-theory.

We use such FOL based systems because in the literature the application description for diagnostic systems is often described as FOL. In the meta-layer METH we describe the functionality of PSMs and here also logic is appropriate because we are interested in declarative knowledge and not in procedural knowledge. The meta-knowledge of METH is FLEXPROB (knowledge of configuring a PSM). This can also be expressed in logic, because here we are also interested in what configuring is. We are not interested in procedural aspects such as the sequence of steps that we have to do for getting a configuration.

An object-meta architecture is a good candidate, because it captures our requirement of supporting the relation between the knowledge types. In an object-meta relation, the meta-theory described relations among the elements of the object-theory. Another requirement for our architecture is the support of abstraction. This is supported because in the meta-theory we can quantify over the elements of the object-theory. This allows us to abstract from specific instances. The third relation is role-assignment. This is supported because
<table>
<thead>
<tr>
<th>meta-meta-theory (FLEXPROB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(= flexible problem solving knowledge)</td>
</tr>
<tr>
<td>• select/configure/adapt methods</td>
</tr>
<tr>
<td>• schema of PSMs</td>
</tr>
<tr>
<td>• relations between components/methods</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>meta-theory (METH)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(= problem solving methods)</td>
</tr>
<tr>
<td>• definitions of methods</td>
</tr>
<tr>
<td>• definitions of method components</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>object-theory (APPL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(= application theory)</td>
</tr>
<tr>
<td>• domain knowledge</td>
</tr>
<tr>
<td>• data</td>
</tr>
</tbody>
</table>

Fig. 1. Structure of the architecture: There are object-meta relations between the domain data/knowledge and the description of PSMs, and between the PSM descriptions and knowledge for flexibly problem solving.

we will use a sophisticated naming relation between object- and meta-theory, where the names will encode the roles of the object-expressions they refer to. For example, between METH and FLEXPROB, names are chosen to express the functionality of the methods. See the next section for details. Our final requirement, the reusability requirement, is also satisfied because of the separation of the object- and meta-knowledge. Our three knowledge types are connected by two object-meta relations thus we have three separated parts of knowledge: object (APPL), meta (METH) and meta-meta (FLEXPROB), see fig. 1.

Summarizing, we conclude that the object-meta architecture could be a good tool for developing an architecture for developing theories for flexible problem solving.

4.2 Type of Meta-Architecture

Two import aspects of meta-architectures, namely the naming mechanism and the reflection rules are discussed below.
Naming: In a meta-architecture the object-theory is separated from the meta-theory. Technically this means that the meta-theory needs a representation for the object-theory. This representation is called the object-model and is used in the meta-theory. The object-model is realised by giving names to object-expressions. These names are meta-expressions, which point to object expressions. In first order logic the expressions of the object-theory could be sentences, predicates, and terms. These expressions are mapped to ground terms in the meta-theory. Because we can quantify about terms in first order logic, we simulate quantifying about object predicates and sentences if we quantify about meta-terms, which are names for these object-expressions. This is what is called the simulation of second order logic by using two first order theories and a naming relation between object-expression and meta-terms.

There are many mechanisms for naming in the literature [24]. Quoting is the simplest one. This means that a predicate \( p(a) \) in the object-theory is mapped to a constant in the meta-theory, often written as “\( p(a) \)”\).

A more sophisticated one is a mapping that preserves the structure of the object-expression. For example the predicate \( p(a) \) is mapped to the functional term \( p(a) \) where \( p \) is a function symbol of the meta-theory. A third form is definable naming. The user can chose his owns names of the object-expressions in the meta theory. This allows to encode more than only syntactic information in a name. For example, if \( p(a) \) plays the role of observation in the inference, then appropriate name would be \( \text{observation}(p(a)) \).

We use such a definable naming relation. Definable naming allows us to give more than one name to one object-expression. It is also possible to give some object expression no name at all in the meta-theory. The possibility of no naming or double naming of object-expressions is not possible in the other mechanisms. They demand that each object-expression has exactly one name. Definable names allow to give different names to expressions that are syntactically similar. This is not possible in the other mechanisms. This allows to represent semantic and pragmatic information in the name. The other mechanisms only encode syntactic information.

Definable naming can also be used for realising the role-assignment relation, discussed in section 3.2, since we can encode the role of an object-expression as part of its name in the meta theory. The same knowledge can play sometimes more then one role, and for reusability of the theories we would like to express this explicitly. Here we can exploit that we have more than one names for particular object knowledges. For example in diagnosis there is sometimes a distinctions between findings and observations, but in other cases there is not. In these cases one can use the same expressions for observations and findings. Sometimes we would like to have no names at all. For example one would like to quantify about a part of the object predicates. In diagnosis for example one would like to quantify only about the things that can be explanations. In such
a case we would like to give only these object expressions names, which could be explanations. The idea is that we would only represent the things that we need at the meta level. We can use the definable naming relation for selecting which expressions are useful.

The notation that we use for the naming mechanism in this paper is as follows: $\text{exp}$ in the meta-layer means the ground term that is the name of the expression $exp$ in the object-layer. We will use capital letters for variables. In a meta-theory, we will write $\overline{Exp}$ for a variable that ranges over names of explanations.

**Reflection Rule**  The connection between the object-theory and the meta-theory is realised by reflection rules. A reflection rule is a rule from a formula that holds in the object-theory to a formula that holds in the meta-theory, or vice versa. These two kinds of rules are called reflection up (from object to meta) and reflection down (from meta to object). Two well known rules are axiom-hood and theoremhood [28]:

\[
\begin{align*}
\text{axiom} : & \quad \phi \in T_{\text{object}} \\
& \frac{}{T_{\text{meta}} \vdash \text{axiom}(T_{\text{object}}, \phi)}
\end{align*}
\]

\[
\begin{align*}
\text{theorem} : & \quad T_{\text{object}} \vdash \phi \\
& \frac{}{T_{\text{meta}} \vdash \text{thm}(T_{\text{object}}, \phi)}
\end{align*}
\]

Axiom-hood means that iff $\phi$ is a sentence in the object-theory $T_{\text{object}}$, then $\text{axiom}(T_{\text{object}}, \overline{\phi})$ holds in the meta-theory. Theoremhood means that iff $\phi$ is derivable in the object-theory $T_{\text{object}}$ then $\text{thm}(T_{\text{object}}, \overline{\phi})$ holds in the meta-theory. We use also an abduction reflection rule, but this rule can be expressed in terms of the theoremhood rule, namely $\overline{\phi}$ is an abducible for $\psi$ iff $\text{thm}(T_{\text{object}} \cup \overline{\phi}, \overline{\psi})$.

We can conclude that our architecture for flexible problem solving is a combination of standard meta-architecture techniques. In the next section we discuss this architecture in more detail.

5 **Details of the Meta-Architecture**

The different knowledge types and their relations are described informally in section 3. In section 4 we described the architecture based on this informal description. In this section, we give formal theories of these knowledge types and their relations as the content of the architecture. In other words we focus
on the technical side of the three knowledge types and their relations in the architecture.

5.1 Application Description Layer

The APPL theory represents the domain data and knowledge. This data and knowledge can be further refined in several types. For example a refinement of the domain knowledge in diagnosis is the distinction between the causal behaviour model and the structural model of the system under diagnosis. We assume that APPL can be expressed in FOL theories. This assumption is reasonable for diagnosis, because in the diagnostic literature the application knowledge is mostly described as logic [6].

Connection to method description The METH theory describes how the domain data and knowledge from APPL is used in problem solving methods. For this purpose the METH theory uses the names of the expressions in the APPL theory. We use the definable naming mechanism for the mapping of object-expressions into meta-terms. For example, in the diagnostic literature, it is more or less standard to represent the causality relation as an implication. An appropriate naming is the mapping of formulas in the causal theory of the form

\[ \text{Cause} \rightarrow \text{Effect} \]  

into names

\[ \text{causes(Cause, Effect)}. \]

The function symbol causes indicates the role of the knowledge. Different knowledge from APPL could concern typologies of faults, such as:

\[ \text{burned-fuses} \rightarrow \text{electrical-fault}. \]

This implication would be named as

\[ \text{abstraction(burned-fuses, electrical-fault)}. \]

where a different function symbol is used (abstraction) to indicate the different knowledge role.
Relations among components of APPL can be expressed in METH by naming the object-formulas from the APPL and using these names (terms) as arguments of the predicates in METH. This satisfies the requirement, that METH should be able to express properties of APPL. Because we use these names as arguments of predicates we are able to quantify about these names. This enables us to give generic definitions of a PSM. This satisfies the abstraction requirement. We realise the role assign requirement, because the naming is based on the role of the APPL expression in the inference of the PSM. If we use another APPL theory then we need only to define names for this new theory in such a way that the object-expressions are mapped into names in correspondence with the role that they play in the inference of the PSM. This satisfies the reusability requirement of METH. The reusability of APPL is realised by using several naming possibilities, when each naming belongs to a specific method. This allows the reuse of the same APPL knowledge for different PSMs.

5.2 Method Description Layer

This layer (METH) describes the functionality of PSMs and their parts. The functionality of a PSM is a relation between a problem and a solution. This relation is described by a predicate that relates the names of the formulas of APPL. These expressions are the input and output of a PSM. Often there are several kinds of input and output of a PSM. For example in diagnosis, the input consists of three types, namely the observations (e.g. pain in chest) which need to be explained, the context which does not need an explanation (e.g. age is 40) and the expected behaviour model (e.g. a causal model). In general the method description predicate has the form of $\text{psm}_i(in(T_1,..T_n), out(O_1,..,O_m))$, where $T_1,..,T_n$ are input names and $O_1,..,O_m$ are output names. The predicate $\text{psm}_i$ has to be defined in METH, that means we have to express how the PSM determines the $\text{out}(O_1,..,O_m)$ using the $\text{in}(T_1,..,T_n)$.

The functionality of a PSM is specified by several components which are combined in such away that it captures the functionality of the PSM. These components are defined as predicates and each of these predicates uses a part of the input/output or intermediate results as arguments. Thus a component predicate is a relation between names of formulas of the APPL or intermediate results.
A general example of a $psm_i$ definition is:

$$
\forall T_1, T_2, T_3, O_4, X, Y :

psm_i(in(T_1, T_2, T_3), out(O_4)) \leftrightarrow

\begin{align*}
&c_1(T_1, X) \land \\
&c_2(T_2, X, Y) \land \\
&c_3(T_3, Y, O_4)
\end{align*}

(2)

The component predicates $c_i$ have to be specified in this layer as well. The definition of these components predicates often contains a reflection step to APPL, using the predicates $thm$ and $axiom$. In general, the conjunction of the $c_i$ can be any formula in which the $c_i$ occur.

Notice that both PSM definitions and component definitions are predicates. Notice also the use of variables (names of APPL expressions) as arguments of these predicates. Therefore, these definitions do not hold for a particular instance, but are defined for arbitrary substitutions for the quantified variables.

A very simple functionality description of a diagnostic PSM is the relation among the observations $\overline{Obs}$, the expected behaviour $\overline{Bm}$ and a minimal explanation $\overline{E}$. We distinguish three components for the functionality: (1) the explanation component: an explanation has to cover a subset of observations (Obs), (2) the not-contradicting component: an explanation be consistent with the observations and (3) a selection component: a criterion that selects a minimal explanation among all possible explanations. See for [22] for a more realistic framework for functionality description of a diagnostic PSMs.

This diagnostic method $diagnostic-psm_1$ can be defined as follows using the predicates $expl-by-cover, notcontra-by-cover,$ and $select-by-subset$ for the components:

$$
\forall \overline{Obs}, \overline{Bm}, \overline{E}, \overline{E}_s :

diagnostic-psm_1(in(\overline{Obs}, \overline{Bm}), out(\overline{E}))

\leftrightarrow

\overline{E}_s = \{ \overline{E}' | expl-by-cover(\overline{Obs}, \overline{Bm}, \overline{E}') \land \\
notcontra-by-cover(\overline{Obs}, \overline{Bm}, \overline{E}') \}

\land select-by-subset(\overline{E}_s, \overline{E})

(3)

$\overline{E}_s$ is the set of all possible explanations determined by the $expl-by-cover$ and
notcontra-by-cover components, while a minimal explanation is selected by the select-by-subset component. The expl-by-cover, notcontra-by-cover and select-by-subset predicates are the components of the functional description of this diagnostic PSM diagnostic-psm₁. Notice that this formula is an instance of the general example above formula (2).

These component predicates have to be defined in METH. As we mentioned before, there is often a reflection step to APPL in these component definitions, for example in the expl-by-cover definition:

\[
\forall\overline{Ob}, \overline{Bm}, \overline{E} : \\
\text{expl-by-cover}(\overline{Ob}, \overline{Bm}, \overline{E}) \leftrightarrow \text{thm}(\overline{Bm} \cup \overline{E}, \overline{Ob}) \tag{4}
\]

Notice that this is the abductive use of the theoremhood rule (see section 4.2). In our example using the simple functionality description above, the method description layer contains definitions of components like formulae (4) and definitions of complete diagnostic methods like (3).

**Connection to FLEXPROB** The third layer (FLEXPROB) describes how the PSMs from METH can be used to solve a problem flexibly. The requirements of being able to state properties, to abstract from instances, and to assign roles, and the reusability are realised in the same way as between the APPL and METH.

We emphasize the names of the METH expressions. These express how the component and PSM definitions, are used for solving a problem flexibly. We give two examples of an appropriate name relation, that we use in our examples. We map the component predicates of the form

\[
C_i(T_1, \ldots, T_n);
\]

to terms of the form

\[
\text{comp}(\text{type}_i, \overline{C_i}, \overline{T_1}, \ldots, \overline{T_n})
\]

In FLEXPROB, \text{type}_i is the pragmatic meaning of the component, indicating the use of a component. For example the component

\[
\text{expl-by-cover}(\overline{Ob}, \overline{Bm}, \overline{E})
\]
would be named as
\[
\text{comp}(\text{explain}, \text{exp-by-cover}, \text{Obs}, \text{Bm}, \text{E'},
\]
indicating that this is an explain component.

In FLEXPROB, \( \overline{C_i} \) is a name of a component predicate symbol in METH. This naming relation allows us in FLEXPROB to quantify over predicate symbols of METH, and also to quantify about particular types of components. The notation \( \overline{T_i} \) in FLEXPROB is a name of a METH expression \( \overline{T_i} \), that is itself a name of a APPL expression \( T_i \). The naming relation that we use for \( \text{in}(\overline{T_1}, \ldots, \overline{T_n}) \) is \( data-in(\overline{T_1}, \ldots, \overline{T_n}) \), and for \( \text{out}(\overline{O_1}, \ldots, \overline{O_n}) \) is \( data-out(\overline{O_1}, \ldots, \overline{O_n}) \). Also here we see how the expressions of METH get names that indicate how they are used in FLEXPROB, namely as data.

5.3 Solving a Problem Flexibly

The subjects in this layer are (i) a general schema of PSMs, (ii) relations and properties of component predicates defined in METH and PSMs, and (iii) strategy knowledge for solving a problem flexibly. (iii) means how do we use (i) and (ii). We will discuss each of these in turn.

**General schema of PSMs** A general schema of PSMs expresses which components can be distinguished in the functionality of a PSM, and how these components have to be combined for expressing the functionality. Such a general schema of PSMs is a relation between the used PSM components, the problem and the solution. The used components definitions are represented as names of predicate symbols of components definitions in METH. The problem and the solution are names of the input/output expressions in METH which are themselves names of expressions in APPL. Such general PSM schema can therefore be described as a relation of the form:

\[
\text{general-psm-schema}( \text{meth}(\overline{C_1}, \ldots, \overline{C_n}), \\
\text{problem}(\text{data-in}(\overline{T_1}, \ldots, \overline{T_m})), \\
\text{solution}(\text{data-out}(\overline{O_1}, \ldots, \overline{O_n})))
\]

The components \( \overline{C_1}, \ldots \overline{C_n} \) together express the functionality. All the PSMs based on this general schema use a definition for each of these components.
The $C_i$ are predicate symbols in \textsc{meth}. For example, the general schema predicate for the diagnostic \textsc{psm}s predicate of the previous section has an explain component, a notcontra component and a selection component:

$$\text{meth}(\overline{\text{Explain}}, \overline{\text{NotContra}}, \overline{\text{Select}})$$

A particular instance is the components used in method \textit{diagnostic-\textsc{psm}$_1$}:

$$\text{meth}(\overline{\text{expl-by-cover}}, \overline{\text{notcontra-by-cover}}, \overline{\text{select-by-subset}})$$

The problem is given by the observations $\overline{\text{Obs}}$ and the behaviour model $\overline{\text{Bm}}$. The general input form

$$\text{problem}(\text{data-in}(I_1, \ldots, I_n)),$$

becomes for diagnosis:

$$\text{problem}(\text{data-in}(\overline{\text{Obs}}, \overline{\text{Bm}})).$$

The solution is the explanation $\overline{E}$, thus the general form of the third argument of the general \textsc{psm} schema becomes $\text{solution}(\text{data-out}(\overline{E}))$. The instance of this schema that expresses the definition of \textit{diagnostic-\textsc{psm}$_1$} from \textsc{meth} given above is:

$$\text{diagnosis-schema}(
\text{meth}(\overline{\text{expl-by-cover}}, \overline{\text{notcontra-by-cover}}, \overline{\text{select-by-subset}}),
\text{problem}(\text{data-in}(\overline{\text{Obs}}, \overline{\text{Bm}})),
\text{solution}(\text{data-out}(\overline{E})))$$

In words: given a problem

$$\text{problem}(\text{data-in}(\overline{\text{Obs}}, \overline{\text{Bm}}))$$

and the method defined by the components \textit{expl-by-cover}, \textit{notcontra-by-cover} and \textit{select-by-subset} gives the solution

$$\text{solution}(\text{data-out}(\overline{E})).$$

It remains to be defined how the different components (e.g. $C_1$) of the method $(\text{meth}(C_1, \ldots, C_n))$ are related together. Notice that this is exactly what we did
in $\text{METH}$ for each specific $psm_i$ definition (equation (3)). In other words, here we give the general scheme for $psm_i$ definitions. Because this schema needs to use the definitions of the $C_i$ components, we have to reflect on $\text{METH}$ for using these definitions: $\text{tm}(\text{METH}, \text{comp}(\text{type}_i, C_i, \text{Args}))$, which corresponds to the provability of $C_i(\text{Args})$ in $\text{METH}$. We give the definition of the general diagnosis-schema for illustrating this. Remember the simple schema of diagnosis: a diagnostic PSM consists of three components $\text{Explain}$, $\text{NotContra}$, and $\text{Select}$: the $\text{Explain}$ component and the $\text{NotContra}$ component generate possible explanations $Es$, and the $\text{Select}$ component selects a minimal explanation $E$. The definition is:

$$
\forall Bm, \overline{Obs}, \overline{E}, \text{Explain}, \text{NotContra}, \text{Select}, \overline{Es}:

\text{diagnosis-schema}(\text{meth}(\text{Explain}, \text{NotContra}, \text{Select}),

\begin{align*}
\text{problem}(\text{data-in}(\overline{Bm}, \overline{Obs})).
\text{solution}(\text{data-out}(\overline{E}));
\end{align*}
\implies

\overline{Es} = \{ \overline{E} | \text{tm}(\text{METH}, \text{comp}(\text{explain}, \text{Explain}, \overline{Bm}, \overline{Obs}, \overline{E}')) \}
\wedge

\text{tm}(\text{METH}, \text{comp}(\text{notcontra}, \text{NotContra}, \overline{Bm}, \overline{Obs}, \overline{E}'))}
\wedge

\text{tm}(\text{METH}, \text{comp}(\text{select}, \text{Select}, \overline{Es}, \overline{E}))
\tag{5}
$$

This $\text{FLEXPROB}$ predicate $\text{diagnosis-schema}$ expresses the schema of the $\text{diagnosis-psm}_1$ definition in $\text{METH}$ (equation (3)), but now we can configure this general schema with different components predicates, since the component predicates are represented by variables ($\text{Explain}, \text{NotContra}, \text{Select}$). This could not be done in $\text{METH}$. This is the main advantage of writing:

$$
\text{tm}(\text{METH}, \text{comp}(\text{explain}, \overline{\text{Explain}}, \overline{BM}, \overline{Obs}, \overline{E}'))
$$

in $\text{FLEXPROB}$ instead of the equivalent

$$
\text{Explain}(\overline{Obs}, \overline{Bm}, \overline{E}')
$$

in $\text{METH}$: formula (3) the $\text{METH}$-predicate $\text{Explain}$ has becomes a variable $\overline{\text{Explain}}$ in $\text{FLEXPROB}$ and can be quantified and substituted. Notice that the naming relation enforces that we only use components of the appropriate types. For example, we use only select components for the $\text{Select}$ definition,
because only these predicates have as their name \( \text{comp}(\text{select}, \overline{\text{Select}}, \overline{E}, \overline{E}) \)). This is an example of role assignment of components from \text{METH} used in \text{FLEXPROB}.

Several schema definitions are possible, for example a very detailed one with many components or a more global one with less components. The measure of varying the functionality of PSMs depends on the number of components and the available definitions of these components. More components increase the possible PSM functionalities that can be described. More available definitions increase also the PSM functionalities that can be described. The idea is to use the schema that distinguishes useful components for guiding the functionality of PSMs for flexible problem solving. In any case the components which are used in the schema definition \textit{need} to be specified in \text{METH}. A desired property of a schema is that it captures most of the PSMs of the problem type.

\textbf{Relations and properties of components definition(s) and PSMs definition(s).} In this paragraph we illustrate some relations between components and how these can be used for expressing relations and properties of methods. An example in diagnosis of a relation between components is if one selection component is stronger then another one. That means that every diagnosis selected by one component is also always selected by the other selection component. Such a relation between two selection components can be expressed simply as a fact:

\[
\text{stronger-selection}(\text{select-by-number}, \text{select-by-subset})
\]

where \text{select-by-number} and \text{select-by-subset} are defined predicates in \text{METH}. But we are also able to give generic definitions of relations between components, because we can quantify over the components. For example if one component \text{Select} always implies the solutions of another component \text{Select'} then we call \text{Select} stronger then \text{Select'}:

\[
\forall \text{Select}, \text{Select}' : \\
\text{stronger-selection}(\text{Select}, \text{Select}')
\]

\[
\forall \overline{E}, \overline{E} : \\
\text{thm}(\text{METH}, \Rightarrow ( \text{comp}(\text{select}, \overline{\text{Select}}, \overline{E}, \overline{E}), \\
\text{comp}(\text{select}, \overline{\text{Select'}}, \overline{E}, \overline{E})))
\]

21
However, the formula (7)\(^1\) is in general not computable, so for pragmatic reason we may prefer the extensional definition (6) over the intensional definition (7).

These relations between components will be used for relations or properties of entire methods. One has to describe the effect of the relations between components on the methods. In our diagnosis example: using a stronger selection criterion results in less solutions of a PSM. We describe this as follows:

\[
\forall \overline{\text{Explain}}, \overline{\text{NotContra}}, \overline{\text{Select}}, \overline{\text{Select'}} : \\
\text{stronger-selection}(\overline{\text{Select}}, \overline{\text{Select'}}) \rightarrow \\
\text{less-solutions}(\text{meth}(\overline{\text{Explain}}, \overline{\text{NotContra}}, \overline{\text{Select}}), \text{meth}(\overline{\text{Explain}}, \overline{\text{NotContra}}, \overline{\text{Select'}}))
\]

(8)

In other words, for relating methods (e.g. less-solutions) we use relations between components (stronger-selection) as part of the condition. Again, we use quantification over predicates of MATH (such as Explain), therefore it is essential that `FLEXPROB` is a meta theory of MATH.

Although we have a general scheme for a method, often there are restrictions among distinct types of components which can be used in one method. Therefore beside using relations and properties of components for expressing effects on method, we use them for expressing constraints on methods, and thus for getting a valid method. In other words we restrict the possible combinations of components in a method by expressing which constrains have to hold. A valid method is a possible method for which the constraints hold. A possible method is built up by reflecting on the MATH theory for finding which components are available in the MATH theory.

In general a possible method is any combination of available components:

\[
\forall C_1, \ldots, C_n, \text{Args}^1, \ldots, \text{Args}^n, C_{1_{\text{def}}}, \ldots, C_{n_{\text{def}}} : \\
\text{possible-meth}(\text{meth}(C_1, \ldots, C_n)) \leftrightarrow \\
\text{axiom}(\text{MATH}, \overline{\overline{\text{comp}}}(C_1, C_{1_{\text{def}}})) \land \\
\ldots \land \\
\text{axiom}(\text{MATH}, \overline{\overline{\text{comp}}}(C_n, C_{n_{\text{def}}}))
\]

(9)

\(^1\overline{\text{comp}}(\text{select}, \overline{\text{Select}}, \overline{\text{Es}}, \overline{E}), \overline{\text{comp}}(\text{select}, \overline{\text{Select'}}, \overline{\text{FEs}}, \overline{E}))\) is the name of the MATH expression \(\text{Select}(\overline{\text{Es}}, \overline{E}) \rightarrow \text{Select'}(\overline{\text{Fes}}, \overline{E})\).
In general a valid method is:

\[
\forall C_1, \ldots, C_n : \\
\text{valid-method}(\text{meth}(C_1, \ldots, C_n)) \leftrightarrow \text{possible-meth}(\text{meth}(C_1, \ldots, C_n)) \land \\
\text{constraints-hold}(\text{meth}(C_1, \ldots, C_n))
\] (10)

Notice that possible-method and valid-method are properties of a method.

In our diagnostic example it is reasonable to demand that “explaining an observation” implies that “not contradicting the negation of the observation” holds too. This restricts the possible combinations of \text{Explain} and \text{NotContra} components in a diagnostic method. In this example we use the restriction between \text{Explain} and \text{NotContra} in the definition of constraints-hold.

\[
\forall \overline{\text{Explain}}, \overline{\text{NotContra}}, \overline{\text{Select}} : \\
\text{constraints-hold}(\text{meth}(\overline{\text{Explain}}, \overline{\text{NotContra}}, \overline{\text{Select}})) \leftrightarrow \\
\text{implies-expr}(\overline{\text{Explain}}, \overline{\text{NotContra}})
\] (11)

Using formulas we can state the relevant properties of components in the form of formula (6):

\[
\text{implies-expr} (\overline{\text{expl-by-cover}}, \overline{\text{notcontra-by-cover}})
\]

A method

\[
\text{meth} (\overline{\text{notcontra-by-cover}}, \overline{\text{expl-by-cover}}, \overline{\text{Select}})
\]

is an invalid method, but

\[
\text{meth} (\overline{\text{expl-by-cover}}, \overline{\text{notcontra-by-cover}}, \overline{\text{Select}})
\]

is valid.

Again such formulas as 9, 10 and 11 cannot be written in \text{METH}, because in \text{METH} we cannot quantify about the components (e.g. \text{Select}), which are predicate symbols in \text{METH}.
**Strategy knowledge for flexible problem solving.** The last kind of knowledge of FLEXPROB is how we use the earlier discussed predicates such as the general scheme of PSMs (see formula (5)) and relations between methods (such as formula (8)). Our goal is to compute the most appropriate solution, given the problem and the goal of problem solving. We define this as a relation between the goal, the input problem, the used method, and the solution. We give an example of the definition of a flexible problem type predicate in diagnosis, *flex-diagnosis*.

In our example, the given goal is to find the smallest number of solutions for a given diagnostic problem. We use the *diagnosis-schema* from the example above (formula (5)). We could achieve our goal (the smallest number of solutions) if we configure a method, that contains the strongest explanation component (expressed by using *stronger-explain*), the weakest not-contra component (expressed by using *weaker-not-contra*) and the strongest component for selection (expressed by *stronger-selection*). These predicates *stronger-explain*, *weaker-not-contra*, and *stronger-selection* have to be defined in FLEXPROB. For example, *stronger-selection* was defined earlier in formula (6). Such a configured method is the desired one for our goal and using this method means that the smallest number of solutions is returned. The definition of *flex-diagnosis* is: given a problem (problem(data-in(\(\overline{\text{Obs}}, \overline{\text{Bm}}\))) and given a goal (smallest-nr) the appropriate solution is any solution (solution(data-out(\(\overline{E}\)))) that satisfies the following:

\[
\forall \text{Explain}, \text{NotContra}, \text{Select}, \overline{\text{Bm}}, \overline{\text{Obs}}, \overline{E}:
\]

\[
\text{flex-diagnosis} \left( \text{problem(data-in(\overline{\text{Obs}}, \overline{\text{Bm}})))},
\right.
\]

\[
\text{smallest-nr},
\]

\[
\text{meth} (\text{Explain}, \text{NotContra}, \text{Select}),
\]

\[
\text{solution(data-out(\overline{\text{E}})))
\]

\[
\implies \text{configure(smallest-nr, meth(Explain, NotContra, Select)) \land}
\]

\[
\text{diagnosis-schema(meth(Explain, NotContra, Select),}
\]

\[
\text{problem(data-in(\overline{\text{Obs}}, \overline{\text{Bm}}))),
\]

\[
\text{solution(data-out(\overline{\text{E}})))}.
\]

24
An example of the configure relation is:

\[ \forall \text{Explain}, \text{NotContra}, \text{Select} : \]
\[ \text{configure}(\text{smallest-\text{-}nr}, \text{meth}(\text{Explain}, \text{NotContra}, \text{Select})) \leftrightarrow \]
\[ \text{valid\text{-}method}(\text{meth}(\overline{\text{Explain}}, \overline{\text{NotContra}}, \overline{\text{Select}})) \land \]
\[ \neg (\exists \text{Explain}', \text{Select}', \text{NotContra}') : \]
\[ \text{stronger\text{-}explain}(\overline{\text{Explain}'}, \overline{\text{Expl}}) \land \]
\[ \text{weaker\text{-}not\text{-}contra}(\overline{\text{NotContra}'}, \overline{\text{NotContra}}) \land \]
\[ \text{stronger\text{-}selection}(\overline{\text{Select}'}, \overline{\text{Select}}) \]  \hspace{1cm} (13)

The case of failing of diagnosis will be another part of the flex-diagnosis definition. As mentioned earlier if the diagnosis method fails then we will adapt the problem solving strategy by adapting data or by initiating a new configuration process.

5.4 Overview

The APPL layer expresses domain data and knowledge as formulas. Expressions of APPL are used in the METH layer by the use of their names as arguments in predicates. These predicates in METH are definitions of a PSM or of its components. The FLEXPROB layer uses these definitions by using the names of these expressions as arguments in its predicates. The predicates in FLEXPROB express a general schema for PSMs in terms of their components. Such a general schema enables us to configure more PSMs than were explicitly defined in METH. Such configuration can be done on the basis of properties of components that are also expressible in FLEXPROB.

Fig. 2 is a picture of the complete architecture with examples of formulas in every layer, which were discussed in this section.

6 Related Work, Conclusions and Future Work

**Related work** In the literature, problem solving methods such as diagnostic methods are often studied as a decomposition of a task in subtasks [1]. We have another view of problem solving methods. We see the problem solving method as a scheme consisting of components, and instantiations of this scheme corresponds to specific methods. We do not expand a task (problem)
<table>
<thead>
<tr>
<th>Knowledge type</th>
<th>Formula schema</th>
<th>Eq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>flex. prob. solv.</td>
<td>(\text{flex-diagnosis}(\overline{\text{Obs}}, \text{smallest-nr}, \overline{\text{Select}}, \overline{E}) \leftrightarrow (12))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{configure}(\text{smallest-nr}, \overline{\text{Select}}) \land )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{diagnosis-schema}(\overline{\text{Obs}}, \overline{\text{Select}}, \overline{E}) )</td>
<td></td>
</tr>
<tr>
<td>configure PSM</td>
<td>(\text{configure}(\text{smallest-nr}, \overline{\text{Select}}) \leftrightarrow (13))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{valid-method}(\overline{\text{Select}}) \land )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\overline{\text{stronger-selection}}(\overline{\text{Select}}, ..) )</td>
<td></td>
</tr>
<tr>
<td>schema of PSM</td>
<td>(\text{diagnosis-schema}(\overline{\text{Obs}}, \overline{\text{Select}}, \overline{E}) \leftrightarrow (5))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{thm}(\text{METH, comp}(\text{select}, \overline{\text{Select}}, \overline{E})).. )</td>
<td></td>
</tr>
<tr>
<td>property of method</td>
<td>(\text{possible-meth}(\overline{\text{Explain}}, \overline{\text{Select}}) \leftrightarrow (9))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{axiom}(\text{METH, \equiv(comp(select, Select, .., ..))}.. )</td>
<td></td>
</tr>
<tr>
<td>relation of methods</td>
<td>(\text{stronger-selection}(\overline{\text{Select}}, ..) \rightarrow (8))</td>
<td></td>
</tr>
<tr>
<td>relation of</td>
<td>(\text{less-sols}(\overline{\text{Select}}, ..) )</td>
<td></td>
</tr>
<tr>
<td>components</td>
<td>(\text{stronger-selection}(\overline{\text{select-by-number}}, ..) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\downarrow \quad \uparrow )</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>problem solving methods (METH)</strong></td>
<td></td>
</tr>
<tr>
<td>Knowledge type</td>
<td>Formula schema</td>
<td>Eq.</td>
</tr>
<tr>
<td>def. of method</td>
<td>(\text{diagnostic-psm}_1(\overline{\text{Obs}}, \overline{E}) \leftrightarrow (3))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{expl-by-cover}(\overline{\text{Obs}}, ..) )</td>
<td></td>
</tr>
<tr>
<td>def. of component</td>
<td>(\text{expl-by-cover}(\overline{\text{Obs}}, ..) \leftrightarrow (4))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{thm}(Bm \cup \overline{E}, \overline{Obs}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\downarrow \quad \uparrow )</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>application (APPL)</strong></td>
<td></td>
</tr>
<tr>
<td>Knowledge type</td>
<td>Formula schema</td>
<td>Eq.</td>
</tr>
<tr>
<td>domain knowledge</td>
<td>(bm\text{ conjunction of } Cause \rightarrow Effect ) (1)</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>( obs, e )</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 2. Overview of the architecture for solving a problem type flexibly. Notice that the \( obs \) in the APPL layer becomes the variable \( \overline{Obs} \) in the METH layer, and that the term \( expl-by-cover \) in METH becomes the variable \( Explain \) in FLEXPROB.
in subtasks (subproblems) but we reason about possible instantiations of the components of the scheme.

The work of Böttcher and Dressler [2] is also about flexible diagnosis, but they do not make a difference between knowledge of the diagnostic method and its meta-knowledge, as we do in our METH and FLEXPROB layer. They also code application knowledge and diagnostic method knowledge in a single language (our APPL and METH). Their work is more focussed on describing search strategies. They work on controlling the diagnosis process in such a way that the functionality does not change.

In [17] the work of [2] is formalized. They define the concept of diagnosis strategies using a modal logic language that makes strategic knowledge explicit.

Also the work [8] can be compared with our work. They work on model-based diagnosis preferences and strategies representations like [17] but using logic meta-programming. They distinguish an object-level and a meta-level. Their object-level contains the system description, the observations and the working hypothesis. In our case the system description and the observations are part of APPL, and the working hypothesis are part of the functionality of the method, therefore part of METH. Their meta-level contains the strategy formulas. In our case such formulas are examples of what we could describe at the FLEXPROB layer.

Washio’s work [27] is also in the diagnosis domain. However, they use only names of the methods, and do not define the functionality of a method.

Our architecture is a knowledge level reflection system [26]. That means the object-model represents the competence of the object-theory and is independent of implementation details. In other words we abstract from the implementation code. Another kind of reflection systems are the symbol level reflection systems [15]. In this kind of systems the reflection is on the code, so this would result in another kind of reflective theory, for example theories of efficiency.

Our work is much inspired by the REFLECT project [26]. Also the work of [20] is in this direction. The REFLECT project also used a knowledge level architecture. The difference with our architecture is that we are able to reason about several systems and relations between these systems, and not only about a stand alone system as in REFLECT. This is because we have used general schemas of PSMs for a problem type which express the functionality in a generic way.

We can also compare our architecture with GETFOL [11,12]. The main difference here is the distance between the code and the abstract model of this code. We choose for an object-model that expresses the functionality and we
therefore have a big distance from code to model. In GETFOL the distance between code and model is much smaller. This gives them the real possibility to automate the connection between code and model. We are not able to do this. However, the price they pay for this is that their models are much more implementation dependent, whereas our models are strictly functional.

In comparison with other work in meta-architectures [16], we have chosen for a logical representation. Furthermore, we use only upwards reflection rules. This is in contrast with other systems which often use other representations (functional, object-oriented, etc), and which also often use downward reflection. Also, we do not only reflect on theoreomhood, but also on axiomhood. Finally, we use a non-standard naming relation, which enables us to represent non-syntactic information in names of formulas.

Conclusions The proposed architecture for flexible problem solving consists of well known reflection techniques: two object-meta relations, definable naming mechanism and the axiomhood and theoremhood reflection rules. We have used standard meta-architecture techniques for solving a problem flexibly.

An open question concerns the generality of our architecture. An important assumption of the architecture is that it is possible to give PSM schemas for functionality. Another assumption is that APPL can be described in FOL. We have shown [22] that these assumptions hold for diagnosis, but whether they hold for other problem types remains to be investigated.

The general lessons we have learned:

(i) A general way to express the functionality of PSMs schematically, under the assumption that the application domain is described in a logical theory.
(ii) A general way to select and configure functionality of PSMs using the schematic form of point (1) above.
(iii) Simple reflection techniques are sufficient for realising (1) and (2).

These general lessons are based on our study in the diagnosis domain of diagnostic problem solving methods.

Future work After having defined the architecture in this paper, a main point for further work is to concentrate on particular axiomatizations of the second and third layer: different diagnostic methods and components, their relations and how to use them for performing flexible diagnosis.

Another direction of future work is the connection of our architecture with realistic systems that provide efficient implementations of the methods we
have specified. When the application description and the diagnostic method
description are available as logical specifications, they have to be connected
to the code of a system that can actually compute the solution.

Acknowledgement

We are grateful to Guus Schreiber, Bob Wielinga and Dieter Fensel for their
comments on an earlier version of this paper. Our work has benefited from
discussions with them.

References

University of Amsterdam, Amsterdam, The Netherlands, June 1993.

diagnostic trackhound in leash. In Proceedings IJCAI–93, Chambery, August
1993.


preferences and strategies reprentation with logic meta-programming. In K.R. Apt and F. Turini, editors, Metalogics and Logic Programming, chapter 11,


International Joint Conference on Artificial Intelligence.


