MML, a Modelling Language with Dynamic Selection of Methods

Vicente Guerrero-Rojo*
vicenter@rsuna.crn.cogs.susx.ac.uk

CSRP 44
ISSN 1 50- 162

COGS, University of Sussex, at Brighton, BN1 9QH

October 1 , 1994

Abstract

Second generation knowledge based systems (KBS) often incorporate multiple problem solving methods. However the decision about which method to use is very much an open problem. The control involved in the dynamic selection of methods is considered a complex activity that requires the acquisition of specific knowledge and strategies. There is a need for modelling languages capable of handling, invoking, evaluating and choosing multiple methods at run-time. There are several modelling languages with such capabilities. With them it is possible to develop robust, more flexible and less brittle systems. Unfortunately, those languages are not flexible enough to cope with the behaviour of the systems when more methods are incorporated. In this paper we propose a new modelling language which overcomes these shortcomings. In doing this a framework is provided for reviewing the flexibility of current modelling languages.

1 Introduction

Second generation knowledge based systems (KBS) often incorporate multiple problem solving methods. In some systems the methods are specialized for a particular subtask [Bylander et al. 93]. For example, the GTD system (Generate, Test and Debug) [Simmons 93] incorporates a different method for each of its main tasks. In this kind of system the decision about which method to use is taken by the knowledge engineer at the design stage. A number of systems provide facilities in choosing which methods to use. For example, the COPILOTE system [Delouis 93], and a medical diagnosis system in TIPS [Punch, Chandrasekaran 93] incorporate, as part of their problem solving, the selection of the most appropriate method for a given task. The decision about which method to use is taken by the system itself at run-time. Finally, other systems allow multiple methods to work on the same problem simultaneously. For example the Guardian system [HayesRoth et al. 89].

The advantages of having multiple methods in a system have become apparent: robustness [Simmons 93], flexibility [Vanwekenhuysen, Rademakers 90], broader kind of reasoning [Delouis 93], less brittleness [Punch, Chandrasekaran 93], reusability [Punch, Chandrasekaran 93].

The decision about which method to use is very much an open problem [Davis, Krivine 93]. The control involved in the dynamic selection of methods is considered a complex activity that requires the acquisition of specific knowledge and strategies [Reinders et al. 91, Delouis 93]. Nowadays there is a need for modelling languages capable of handling, invoking, evaluating and choosing multiple methods at run-time [Chandrasekaran, Johnson 93].

*sponsored by CONACyT, MEXICO.

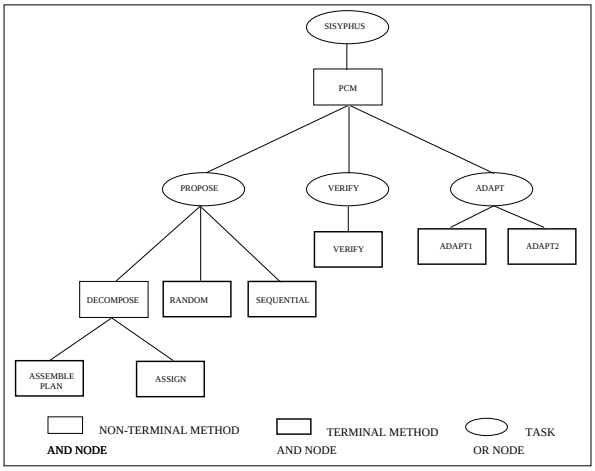


Figure 1: A control structure

participate in the satisfaction of the node. An OR node indicates that just one of its children is sufficient. Such a structure may be a tangled hierarchy since a task or method can appear at several places in the tree. In terms of the model of expertise in KADS methodology [Wielinga et al. 92] the control structure refers to the inference layer and the task layer.

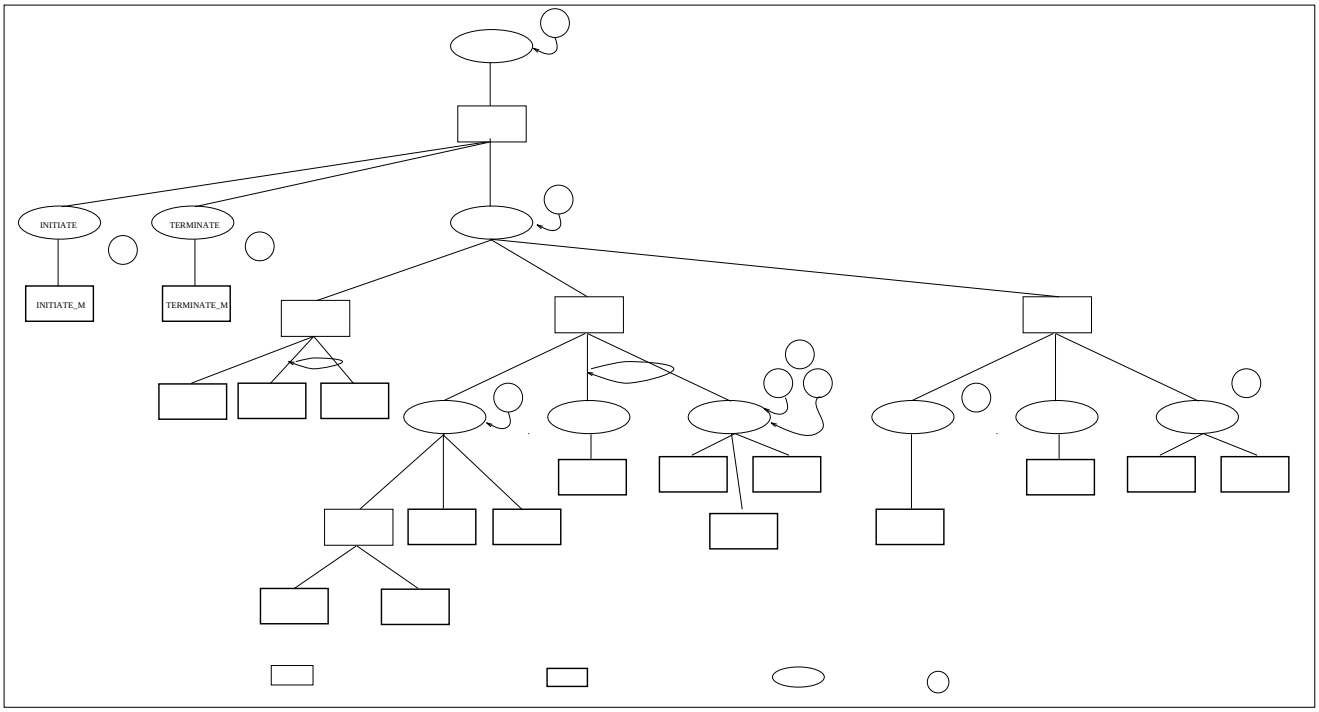
2.1 A case study, Sisyphus-92

Throughout the text the ideas are clarified by examples from the problem known as Sisyphus-92 (from now on the Sisyphus problem) [Linster 91, Linster 92]. Sisyphus was a project that has aimed at comparing different approaches of knowledge modelling. Modelling of knowledge and the influence of those models on the knowledge acquisition activity were the main objectives. A sample problem concerned with office assignment (resource allocation) in a research environment was provided.

The problem consisted on the allocation of some members of the research group YQT to rooms on a new floor. This problem introduced the issue of brittleness. The main interest was to see how a model reacts to unusual situations. In this case, an over-specified problem. This problem was selected because it presents given some interesting characteristics:

- It is a well known problem in the knowledge modelling area.
- Different methods have been applied to the solution of this problem (single method approaches): a general backtracking one (KARL [Angele et al. 92] and MODEL-K [Drouven et al. 92]), a decomposition one (KADS-I [Schreiber 92]), a case-based (MODEL-K), and constraint-based method (MODEL-K and CARMEN [Tong 92]).
- Those approaches embody some assumptions which makes them brittle when they are not satisfied. For example, the KARL approach assumes there is time and space enough to search over the complete search space of possible allocations. The KADS approach assumes a separable problem which can be separated into disjoint subsets whose respective solutions can be joined together later on. The CARMEN approach assumes knowledge that may not be available (i.e.. constraints can be ordered by importance and have associated fix knowledge -what to do in case a constraint cannot be satisfied) [2].
- This problem presents a subset of the problems found in a most general areas such as scheduling which can be solved with different techniques such as constraint satisfaction, simulated annealing, genetic algorithms, tabu search, repair heuristics, and so on [Prosser, Buchanan 94].

[2] It is interesting to note that even though all these methods were part of a single system the brittleness problem is still not solved. It might happen that, although the problem has a solution, no method could be activated .


INITIATE
TERMINATE
INITIATE_M
TERMINATE_M

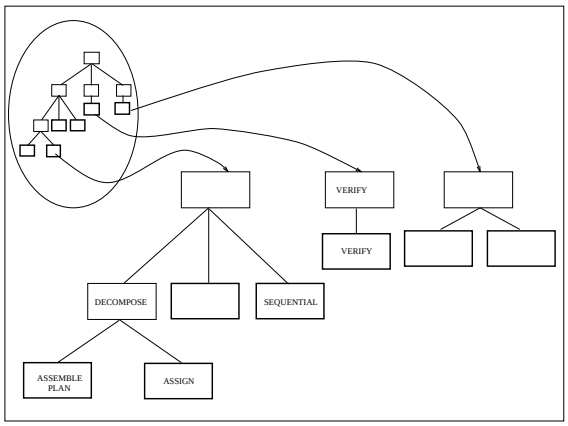
components of a modelling language are:

- An object-level with its control structure and associated control knowledge.
- Meta-level objects and their properties.
- Abstract structures.
- Meta-level activities.
- A meta-level.

In the following paragraphs these components will be described although not in the same order as they appear above. The instances are the specific entities in each component (i.e. methods and tasks in meta-level objects, different criteria in abstract structures).

- Control structure. There is a trend to develop modelling languages which adhere to leading methodologies. Following an established methodology determines the kind of entities (i.e. knowledge sources, roles, tasks, goals, methods, basic inferences) and control structure that the systems will have. Thus, identifying the underlying methodology of a language determines the control structure and entities. For example, the AND/OR tree is one of the most common control structure [Delouis 93]. It is based on hierarchical decomposition of entities (e.g.

etc. Therefore, a number of entities need to be defined as meta-level objects incorporating meta-knowledge in their descriptions. Thus, it is important to answer questions such as: what kind of objects are defined at that level? Are they implicit or explicit? Where and how are they described and manipulated? Do they have a static or dynamic description?


VERIFY
VERIFY
DECOMPOSE
SEQUENTIAL
ASSEMBLE
PLAN
ASSIGN

how a task can be solved. Two types of methods are distinguished: decomposition methods (they decompose a task into subtasks), and action methods (they provide direct solutions for a task).

Only two meta-level activities (method related) are found in this framework: the selection and activation of methods. In this framework the selection of methods depends on characteristics of the task such as: knowledge availability, runtime constraints, cost of observation and so on. Thus, the dynamic selection of methods is carried out by the satisfaction of the applicability criteria of the methods. This activity is implicit in the language.

Its control strategy is simple. It consists of the selection of the best method and its activation. This control strategy is implicit. All the tasks in the framework share the same control strategy.

Tasks, methods and domain knowledge are explicitly represented as objects. This makes them available for inspection, modification, or adaptation. A task is characterized by its: inputs, outputs, domain knowledge, and the practical problems that occur in the domain (incompleteness of information, uncertainty, etc.). A method is characterized by: its decomposition, control knowledge over the decomposition, and conditions indicating when a method is applicable to a task (applicability criterion). It is not clear if such descriptions can be extended or modified.

It is not clear as well what kind of control statements and abstract structures the language provides.

In summary, this language provides an implicit and simple control strategy which can be applied to every task in the control structure. Its objective is just to choose a method. It is based on a simple cycle applying an applicability criterion: the first method that satisfies the criterion is activated.

The disadvantages that can be appreciated in this language are:

- Meta-level activities. This language does not provide facilities for any additional meta-level activity. Therefore other activities such as method ordering and method monitoring cannot be represented.
- Control strategy. Since it has an implicit fixed control strategy it is neither possible to modify it nor to define new or specific strategies. For example, a strategy that iterates over the methods in case of method failure.
- Abstract structures. TroTelC has a single abstract structure, namely the applicability criterion. Therefore, its criterion must contain other embedded criteria (i.e. necessity, appropriateness) which make it complex and difficult to explain the reasons why a method cannot be selected.

3.4 TIPS

TIPS [Punch, Chandrasekaran 93] is a task-specific language for diagnosis that allows the development of systems involving the integration of multiple methods and their dynamic selection. This language is a response to the dynamic selection of methods problem in the GT approach. The TIPS approach is to provide only enough mechanism to allow monitoring of tasks (goals in TIPS terms) and a mapping of methods that can achieve a task. The design of those methods is outside the TIPS approach.

In TIPS the control structure consists of a task-subtask-method tree. In this tree a node can be a task or a method. An initial task is decomposed into subtasks and so on. Only the last subtasks are carried out by methods. Most of the tasks are defined as AND nodes while a few of them (named control choice points) are defined as OR nodes. The control knowledge associated with the AND nodes is heuristic. The control knowledge associated with the OR node is called a Sponsor-Selector structure. This structure is a combination of abstract structures (applicability, appropriateness and tie-breaker criteria) and meta-level activities (e.g. verification of task satisfaction, identification of special cases, ordering methods, and method selection).

The basis for the representation of its method selection activity is the Sponsor-Selector structure. It is a hierarchy of three elements: a selector, a set of sponsors and a method for each sponsor. Each task with multiple methods (control choice point), has such an associated structure. At any control choice point, the

sponsors are activated to rate their associated methods, then the selector chooses one of those methods based on the sponsor values and other data.

Each sponsor is independently coded without taking into account other sponsors (local knowledge). They provide a measure of the appropriateness of their associated methods (appropriateness criteria) to a given context. The measure provided by the sponsors is a result of a pattern matching process (a table that contains patterns associated with appropriateness measures) about two kinds of information: dynamic method information - this describes which methods have run so far and when they ran; dynamic task information - Its concern is task achievement. For example, has the finding been explained? This information is provided by the knowledge engineer.

The technique of rating methods based on an appropriateness measure is a good one since it concentrates just on the events (set of conditions) that makes a method appropriate. It provides an ordering of the methods for every recognized context.

The objective of a selector is then to decide what method to activate next. The criteria involved in this decision are:

- Appropriateness measures. The highest measure wins. If no clear candidate is available and no other criterion is available, then a random choice from the best candidates is selected. The former is an appropriateness criterion and the latter is a tie-breaker one.
- Priority list. This is a list of methods which specify which method should be preferred in the case of ties. It is a tie-breaker criterion.
- Pattern matching. This structure is similar to the one used by sponsors, but in this case instead of an appropriateness measure, the name of the method is returned. It is used in special situations to override the normal choice mechanism (categorical criterion). According to [Punch, Chandrasekaran 93] an example of this situation is when a method has been applied but not yet completed, then it should be the next selected method. In cases where no matching occurs, the priority list is used.

There are two distinguished sponsors:

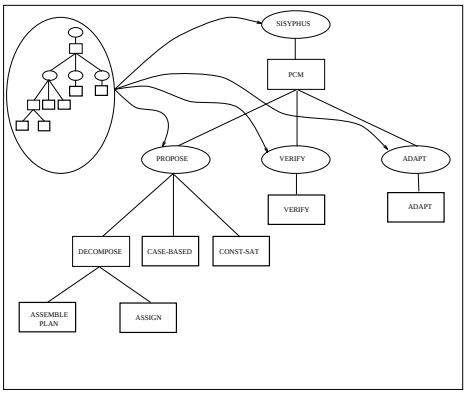


Figure 4: Control Strategy and Control Structure in LISA

3.5 LISA

LISA [Delouis 93] is a general purpose knowledge modelling language. Its design was influenced by the KADS and COMMET [Steels 90] methodologies. This language has three underlying principles: the modelling of expertise using three perspectives - methods, problems and domain knowledge; the dynamic selection of methods and the explicit description of activities and control knowledge intervening in the process of selection.

In LISA the control structure is a task-method tree (a task in LISA is named a goal). A node can be a task or a method. A task is decomposed into methods and a method into tasks or terminal methods. Tasks are OR nodes, whereas methods are AND nodes.

LISA is a reflective language which contains and explicit meta-level. It contains a double control structure, one for the object-level and the other for the meta-level. In the meta-level the activities are represented explicitly. LISA supports several meta-level activities and new ones might be defined. They are method related activities. The activities identified in this language are: collect methods - this collects the methods that might achieve a task; select a method - this is in charge of selecting a method from the collected methods; activate a method - this applies a method; results evaluation - this verifies whether the task has been achieved.

These activities are represented in the same terms as tasks, so, it is possible to define different ways to carry them out. These ways are represented in LISA as methods. Therefore, tasks meta-tasks, methods and meta-methods can be identified.

In LISA just a single control strategy can be defined. It might be user-defined or system-provided. This strategy is applied to every task in its two control structures. A control strategy in LISA is defined explicitly in terms of tasks and methods (see figure 4). The tasks define the activities whilst the methods the ways to carry them out. This ingenious representation allows definition of different ways to carry out those activities. New tasks and methods can be added, eliminated or modified. Therefore, a control strategy in LISA is as

Since the construct meta-task is the most important in LISA it will be described in detail. This meta-task is carried out by a single meta-method which has the following meta-tasks:

1. Identify possible methods. This meta-task generates a list of methods that might carry out a task. It has three associated meta-methods:
   - Collect associated methods. This activity just obtains those methods assigned on an *a priori* basis to a task.
   - Dynamically collect possible methods. This activity collects those methods in the system whose results satisfy the task (called pertinent methods in LISA) and whose inputs and requirements (applicability criteria) are available in the context (applicable methods in LISA).
   - Collect non-applicable pertinent methods. This activity (known elsewhere as subgoaling) collects non-applicable methods and generates a new activity whose goal is to activate any of those non-applicable methods. It is used when the pertinent methods cannot be activated and so the task cannot be satisfied.
2. Select one method. This activity is in charge of selecting one method from the methods collected in the previous activity. It has the following associated meta-methods:
   - Select one. Used when there is only one method available.
   - Select by favorable contexts or preferences. This activity eliminates from the associated methods those whose context is not appropriate. It applies the favorable contexts criterion. This meta-method has three sub-meta-methods to choose just one method: ask the user, use preferences (preference lists) and at random.
3. Activate a method. This is in charge of recognizing the type of method (terminal, non-terminal or rule based) and calling the specific interpreter to activate the method.
4. Evaluate the success of a method. This activity verifies whether the task has been achieved and whether the results of the method are of the expected quality.

Tasks and methods are the only entities considered meta-level objects. They have an explicit description which consists of predefined sets of properties. Most of them are abstract structures. The abstract structures identified in this language are: criterion of success - this is a satisfaction criterion; favorable contexts - this is an appropriateness criterion; associated methods - a list of methods that are known to satisfy a task; preferences - this is tie-breaker criterion.

In summary, it can be said that LISA is a language that allows the explicit description of control strategies which can be adapted for each application (just the explicit one). Each activity in the control strategy might be carried out in different ways. New activities can be incorporated into the control strategy. The control strategy in LISA basically consists of four method related activities: method collection, method selection, method activation, and evaluation of methods' results. These activities are controlled using a task-method control structure.

The disadvantages that can be appreciated in this language are:

- Control strategy. In LISA there is only a single control strategy associated with tasks in both levels. The language has been designed to apply the same control strategy to every task in the system. Thus, a specific task cannot have its own control strategy. The control strategy needs to be too general to satisfy any possible task requirement or to have dedicated activities for specific tasks.
- Meta-level objects. The meta-level objects in LISA have a fixed description. This means that it is not possible to add new properties or abstract structures to them. Thus, when an extra property is needed in an activity it has to be included implicitly by means of symbol-level constructs (Lisp predicates). This represents a shortcoming in those cases in which the methods involved can only be differentiated by those additional properties. For example, in those cases in which there is a trade-off between time and space.

- Control knowledge. In LISA there is a lack in the language with respect of control in the method decomposition. Lisp code is used instead which implies that such representations are not fully at the knowledge level.

3.6 Discussion

After reviewing a number of modelling languages some disadvantages can be summarized:

- Some of those languages have concentrated almost exclusively on the dynamic selection of methods ignoring basic method related activities such as diagnosis and repair, and competence assessment.
- In most languages, the meta-level objects are characterized by a fixed number of descriptors, namely inputs, outputs and requirements. This represents a shortcoming since the addition of new methods might require the use of new descriptors to differentiate those methods. For example, important properties such as how many solutions can a method provide, or completeness [7]. Sometimes that

In summary, it could be said that current languages provide environments for defining single control strategies with a few meta-level activities and a few abstract structures in which no assumptions are considered and the separation of methods is required (addition of methods only at upper levels in the control structure).

4 MML - Multiple Method Language

The MML is a task-independent modelling language for the explicit representation of systems with flexible control strategies which allow dynamic selection of methods. MML is being designed as an initiative to overcome the above mentioned shortcomings, as well as to easy the representation and acquisition of the knowledge, activities and control strategies involved in such systems.

This approach has been influenced by the languages LISA [Delouis 93] and TIPS [Punch, Chandrasekaran 93], and the methodologies KADS [Wielinga et al. 92] and Generic Tasks [Chandrasekaran 90]. In fact it can be said that MML is a generalization of LISA and TIPS. The underlying ideas of this approach are:

- A modelling language should be a reflective language in which instances of the components mentioned in Section 3.1 can be defined.
- At the same time, the underlying architecture should be open ended in order to allow the addition of new instances or instances with extended or modified descriptions. This facility represents a generalization from current approaches.
- The objects in the system should be described not just in terms of their features (properties and abstract structures) but also in terms of how those properties are used (activities and control strategies).
- A general modelling language should be meta-task specific. It means that it should provide some primitives meta-level activities for specific groups (i.e. method selection, explanation, monitoring). For example it might have method related meta-level activities primitives.
- Brittleness might be reduced not just by providing multiple methods and the capability for their dynamic selection but also, incorporating a number of other method related activities [8]. For example, monitoring the development of the execution of the method, fault diagnosis and repair, analysis of results (e.g., quality, quantity), etc. Thus, a language for the specification of second generation expert systems should be capable of representing and controlling those activities, and the related abstract structures and properties involved in them.

```
[define                 propose
properties:
   [type
```

associated (on *a priori* basis) methods that are known to satisfy its goal. Among the properties that a task might have, the following are the most common: goal, input, output.

```
[define             decomposition
properties:
   [type            method]
   [goal            'To allocate components into resources using ...']
   [input           components resources]
   [output          allocations]
   [m-type          non-terminal]
   [ck-type         procedural]
   [backtracks      false]
   [structure       assemble-plan assign-resources]
abstract structures:
   [applicability-criteria
                    components-value       exist and
                    resources-value        exist ]
   [appropriateness-criteria
                    a-plan                 exist and
                    is-a-separable-problem exist ]
   [code            debug("m", 'Executing decomposition');
                    getdomain( "components", "value") -> Components;
                    getdomain( "resources", "value") -> Resources;
                    for Res in Resources do
                        putdomain("store", "allocations", Res, []);
                    endfor;
                    ...]
activities:
   [applicable      eval-boolean]
   [apply-method    call-method]
control strategy:
%   [decomposition-strategy  apply-method with code]
].
```

Figure 6: Decomposition method in the Sisyphus problem

```
[define                     single-method
properties:
   [goal                    'call an object-level method']
   [type                    method]
   [ck-type                 procedural]
   [m-type                  meta-method]
   [structure  collect-methods applicable
               apply-method satisfaction]
abstract structures:
   [code    take-method with associated-methods
            if applicable then
                apply-method
                satisfaction with satisfaction-criteria
            endif]
activities:
   [apply-meta-method       call-method]
control strategy:
].
```

Figure 7: Single-method meta-method in the Sisyphus problem

conceptual, operational, dynamic and other. The conceptual properties describe the knowledge the method use. The operational ones describe how the method carries out a goal. It includes basically the methods' decomposition into subtasks or sub-methods. The dynamic properties describe the status of a method at run-time. They are mainly modified by the interpreter of the system. For example, the number of times that it has been activated. Finally, other properties describe

5 Conclusions

This paper has proposed a framework for analyzing modelling languages, with particular emphasis on use of multiple methods. This framework identifies the following as important components of second generation expert systems: an object- and a meta-level, meta-level objects, abstract structures, and meta-level activities.

We reviewed the languages: MODEL-K [Karbach, Voβ 92], TroTelC [Vanwekenhuysen, Rademakers 90], TIPS [Punch, Chandrasekaran 93], and LISA [Delouis 93]. The following features were noted:

- Current modelling languages provide a range of components' instances that goes from a few (MODEL-K) to many (LISA).
- Although according to their authors, more robust, more flexible, or less brittle systems might be developed, those languages are not flexible enough. Their components are fixed, namely: a fixed set of properties and abstract structures; a fixed set of meta-level activities; or, a single control strategy for handling those activities.
- Some of those languages have concentrated almost exclusively on the dynamic selection of methods ignoring meta-level activities such as diagnosis and repair, and competence assessment, which are very important related activities.
- They have concentrated on the use of general methods. They do not provide any support for repre-

References

[Angele et al. 92] Angele J., Fensel D., Landes D., An executable model at the knowledge level for the office-assignment task, in: Linster Marc, Sisyphus'92 Models of Problem Solving, Gesellschaft Fur Mathematik, Und Datenverarbitung MBH, 1992.

[Bartsch-Spörl, Bredeweg 91] Bartsch-Spörl B., Bredeweg B., Studies and Experiments with Refelctive Problem Solvers, ESPRIT Basic Research Project P3178 REFLECT PROJECT, Doc. RFL/BSR-UvA/II/2/1, September 1991.

[Bartsch-Spörl, Reinders 90] Bartsch-Spörl B., Reinders M., A Tentative Framework for Knowledge-level Reflection, ESPRIT Basic Research Project P3178 REFLECT, Doc. RFL/BSR-ECN/I.3/1, May 1990.

[Bylander et al. 93] Bylander T., Wientraub W., Simon S.R., QUAWDS: Diagnosis Using Different Models for Different Subtasks, in: Davis J.M., Krivine J.P. (editors), Second Generation Expert Systems, Spring Verlag, 1993.

[Chandrasekaran, Johnson 93] GenericSecond

[Maes 87] Maes Pattie, Computational Reflection, Technical Report 87-2, Artificial Intelligence Laboratory, University of Brussels, 1987.

[Minton et al. 92] Minton S., Johnston M., Philips A., Laird P., Minimizing conflicts: a heuristic repair method for contraint satisfaction and scheduling problems, Artificial Intelligence, 58 1992, pp 161-205, Elsevier.

[Prosser, Buchanan 94] Prosser P., Buchanan I., Intelligent Scheduling: past, present, and future, Intelligent Systems Engineering, Summer 1994.

[Punch, Chandrasekaran 93] Punch W.F., Chandrasekaran B. An Investigation of the Roles of Problem-Solving Methods in Diagnosis, in: Davis J.M., Krivine J.P. (editors), Second Generation Expert Systems, Spring Verlag, 1993.

[Reinders et al. 91] Reinders M., Vinkhuyzen E., Voβ A., Akkermans H., Balder J., Bartsch-Spörl, B., Bredeweg B., Drouven U., van Harmelen F., Karbach W, Karsen Z, Scheiber G., Wielinga B., A Conceptual Modelling Framework for Knowledge-Level Reflection, AI Communications, Vol. 4, No 2/3, Jun/Sep 1991, pp 74-87.

[Schreiber 92] Schreiber Gus, Applying KADS-I to the Sysiphus Domain, ESPRIT Project P5248 KADS-II, Doc. KADS-II/WP6/TR/UvA/19/2.0, July 1992.

[Simmons 93] Simmons Reid, Generate, Test and Debug: A Paradigm for Combining Associational and Causal Reasoning, in: Davis J.M., Krivine J.P. (editors), Second Generation Expert Systems, Spring Verlag, 1993.

[Steels 90] Steels Luc, Components of Expertise, AI Magazine, Summer 1990.

[Tong 92] Tong Huejun, Solving the Office Assignment Problem wiht CARMEN, in: Linster Marc, Sisyphus'92 Models of Problem Solving, Gesellschaft Fur Mathematik, Und Datenverarbitung MBH, 1992.

[Vanwekenhuysen, Rademakers 90] Vanwekenhuysen J., Rademakers P. Mapping a Knowledge Level Analysis onto a Computational Framework, in: Aiello Loigia C. (Editor), Proceedings of the 9th European Conference on AI, 1990.

[Werner et al. 89] Werner K., Linster M., Voβ A., GMD, Proceedings of the 5th Knoledge Acquisition for Knowledge-Based Systems Workshop, Alberta, Canada, November 1990.

[Wielinga et al. 92] Wielinga B., Schreiber A. Th., Breuker J.A., KADS: A Modelling Approach to Knowledge Engineering, in: The KADS Approach to Knowledge Engineering Special Issue, Knowledge Acquisition, Vol. 4, No. 1, March 1992. Academic Press, London.