MML, a Modelling Language with Dynamic Selection of Methods

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

CSRP 344
ISSN 1350-3162

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

October 13, 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.

There are several modelling languages with such capabilities. With them it is possible to develop robust, more flexible and less brittle systems. These capabilities are mainly derived from the addition into the systems of more than one method and specialized activities such as selection, ordering and evaluation of those methods. The success of those languages so far is due to the use of general methods. In other words, no interaction at all between methods exist. However, problems may appear when: the methods involved are sub-methods (part of other methods), in particular when some of those combinations are invalid, effective for specific problems or inefficient; when the user of the systems have preferences for specific combinations rather than for single methods; when no method can be chosen because of a lack of information about the context of the problem; when some of the methods require a particular handling that differs from the others. Thus, it can be said that those modelling languages are not flexible enough to cope with the behaviour of the systems when more methods are incorporated.

As an initiative to overcome those shortcomings a new modelling language is proposed. Thus, the objectives of this work are:

- To determine what the basic characteristics a flexible modelling language should provide,
- To see how flexible current modelling languages are, and
- To define a modelling language that incorporates such characteristics and be capable of overcoming the above mentioned problems.

With that in mind, the next section provides the basic terminology and an application problem (the Sisyphus-92 problem). The third section contains a framework defined to describe modelling languages and an analysis of the following languages: MODEL-K [Karbach, Voβ 92], TroTelC [Vanwekenhuysen, Rademakers 90], TIPS [Punch, Chandrasekaran 93], and LISA [Delouis 93]. The fourth section provides the description of MML (Multiple Method Language) a flexible task-independent modelling language for the explicit description of systems that allow the dynamic selection of method. Finally, in the last section the conclusion are given.

2 Terminology

In order to avoid confusion in the terminology used in this work some of the terms will be defined.

The term task has been used elsewhere to denote an instance of a problem, a problem class, and both a problem class and an abstract description of a method [Chandrasekaran, Johnson 93]. We use the term task to refer to a type of problem (a set of problems with something in common). Thus, resource allocation [1] is a task. A task has a goal which is a specification of what needs to be achieved. A task by itself does not include, as part of its description, any specification of how it will be accomplished. A task may be decomposed into subtasks.

The same situation appears with the term method. It has been defined in different ways for example, as a procedure that defines, selects, instantiates, and executes actions [Werner et al. 89]. We use the term method to define a way in which the goal of a task can be achieved. Accordingly, it is possible to talk about the application of the Heuristic Classification [Clancey 85] method to the fault diagnosis task in the car engine domain. Methods might be of many kinds, for example, a precompiled algorithm, a search in a state space, a connectionist network and so on [Chandrasekaran, Johnson 93]. A Method can be described in terms of subtasks, the sub-methods it employs, plus any control knowledge about how to organize subtask or sub-method application to satisfy a task goal. There are some distinguished methods named terminal methods and basic inferences.

The structure that represents the relationships between tasks and methods is called the control structure (see figure 1). This structure is an AND/OR tree in which an AND node indicates that all its children

[1] A resource allocation problem is characterized by two sets of objects where each element of one set is assigned to one element from the other satisfying a set of requirements or constraints. Sometimes these problems have an objective function to evaluate the quality of the assignment.

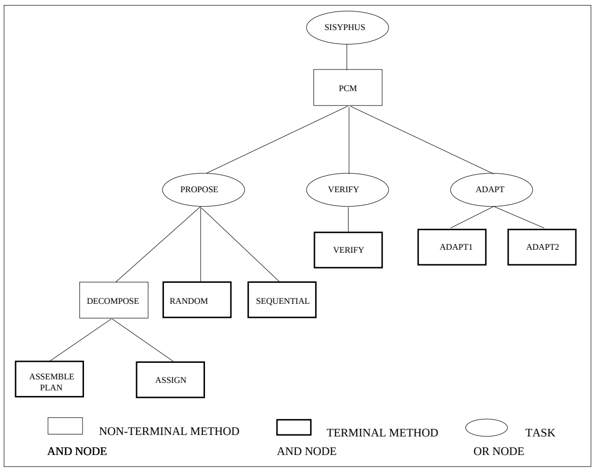

Figure 1: A control structure

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 .

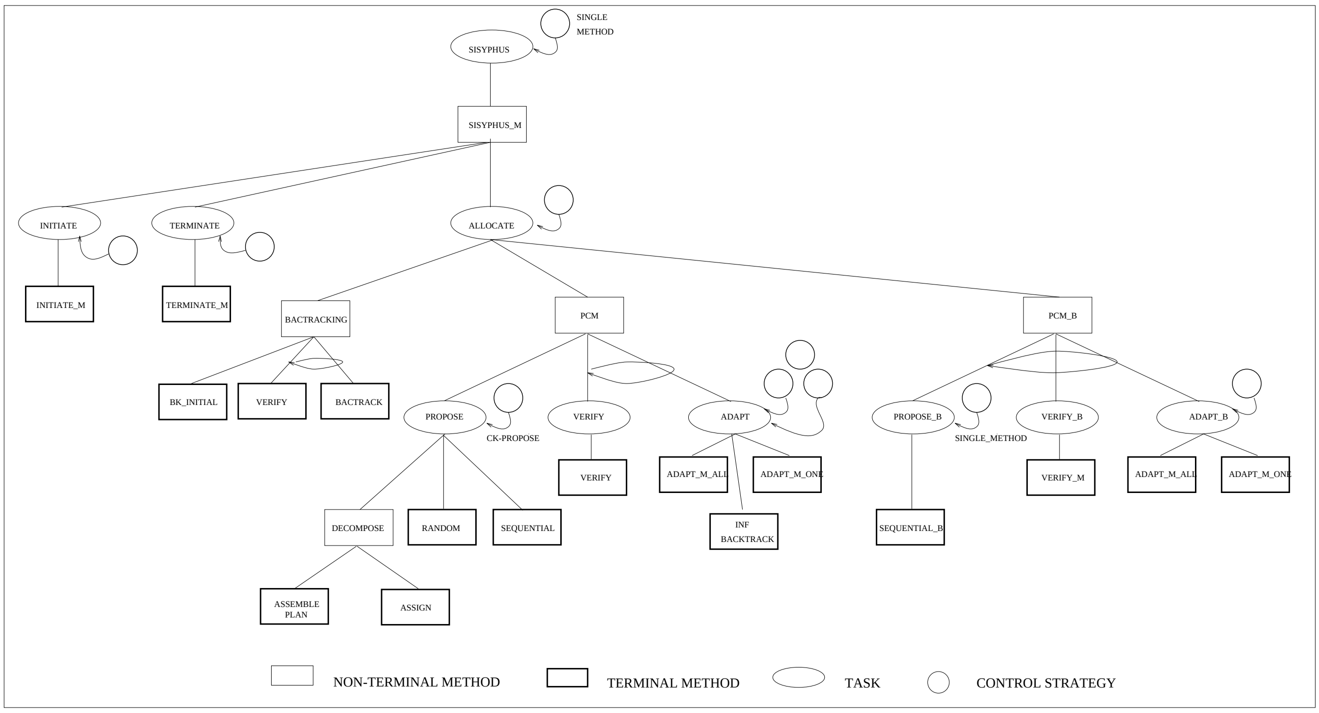

Figure 2: Control structure of the Sisyphus problem

In order to test our ideas about dynamic selection of multiple methods, we envisage a system that solves resource allocation problems, and in particular the Sisyphus problem, with the following characteristics:

- It incorporates several methods which are capable of solving allocation problems.
- Those methods are selected dynamically.
- The system contains strategies that overcome the already mentioned disadvantages of current modelling languages and in particular those of the single method approaches.
- The system incorporates methods belonging to the PCM family of methods. Those methods are: constraint-based [Tong 92], simple backtracking, and backtracking with the min-conflicts heuristic [Minton et al. 92] (see figure 2). They consist of three tasks: Propose, Critique and Verify. The first task proposes a solution to the problem. The second one verifies whether the solution satisfies all the constraints. The last one makes the required modifications (e.g., ordering of input data, relaxing constraints) in order to get a solution, or a partial one in over-specified situations.

In the next section we review a number of current modelling languages that allow the development of systems which incorporate dynamic selection of methods.

3 Modelling Languages with Dynamic Selection of Methods

Nowadays, there are several modelling languages that provide dynamic selection of methods. Unfortunately, the terminology used is not homogeneous. Every language defines its own terms and graphical representation. Thus, in this section a framework for describing modelling languages is proposed and a number of them are briefly analyzed and criticized based on this framework.

3.1 The Framework

In order to have a common way to refer to other modelling languages it is necessary to establish a common terminology. The proposed framework consist of a set of basic components and their instances. The basic 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. tasks and methods). Methodologies such as Components of Expertise [Steels 90] or GT [Chandrasekaran 90] propose a clear separation between the problems (tasks) and the description of the methods available to solve them. Such a control structure does not entail any kind of control knowledge. It just describes the relationships between the tasks and methods.
- Meta-level activities. An underlying idea in this work is that there are two types of control knowledge which can be identified and clearly separated: knowledge for controlling the decomposition (sequencing) of subtasks or sub-methods, and knowledge for controlling meta-level activities [3]. Thus, while talking about modelling languages it is necessary to address issues such as: is there an explicit meta-level? Does the language support meta-level activities? If so, what are they? How are they represented and activated? Are they implicit or explicit? Is it possible to define new meta-level activities? Are those activities method related or system related? Are they global (applicable to every node in a control structure) or local (just for a few of those nodes)?
- Control strategies. Since the main interest in this work is about modelling languages with multiple methods, special attention is reserved for method related meta-level activities contained in control strategies [4]. Thus, it is necessary to ask questions such as: what kind of control strategies can be defined for controlling methods? Are they explicit? What are the basic activities in those strategies? How are those activities are controlled? Can new control strategies be defined? How flexible are those control strategies? For example, Can a strategy be defined that reselects a method once it has been previously selected (e.g., a backtracking method)? Is it possible to define strategies that incorporate tie-breaker mechanisms in case of method draws? Can a new method be added to a control structure? Are the properties involved in the selection of methods a fixed set? What kind of control statements does it provide?
- Meta-level objects. A number of those activities require a certain amount of knowledge about system entities (e.g., methods, domain models). This knowledge can be explicit or implicit. In general this knowledge is represented implicitly through the use of domain entities as in GTD [Simmons 93] or in the Sisyphus problem (e.g., the existence of an allocation implies that the propose task has succeeded). However, some of those activities treat methods as entities themselves which can be ordered, selected,

[3] A meta-level activity is an activity that occurs in a meta-level system. A meta-level system is a system that has as domain another computational system called its object system. A meta-level system reasons about and acts upon another system [Maes 87]. Meta-level activities can be grouped into groups: method related activities (e.g. method selection, method monitoring, diagnosis and repair) [Chandrasekaran, Johnson 93], and system related activities (e.g. sensible explanation, competence assessment, knowledge base maintenance, performance validation) [Harmelen 92].

[4] A control strategy is the set of meta-level activities that carried out under an specific control allow the satisfaction of a task. It incorporates the control knowledge specified by the methods that achieve the task as well as the control knowledge for controlling the meta-level activities related with those methods.

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? Can such descriptions be modified?

- Control Knowledge. Sequencing knowledge, commonly known as control knowledge, has been dealt with in different ways. Languages following KADS methodology propose control primitives such as loops and conditionals (procedural representation) [Fensel, Harmelen 93] while languages for the GT methodology propose heuristic approaches based on operators and preconditions in a search-space (heuristic representation). Thus, it is necessary to describe what is the nature of the sequencing knowledge in the language.

- Abstract structures. Finally, it is important to highlight the roles that certain abstract structures [5] play in control strategies as well as the vocabulary involved in their description. For example, in method related activities a number of these structures might be found: satisfaction and failure criteria - to identify when a task has been achieved; categorical criterion - to avoid excessive deliberation; applicability criterion - to verify whether a method can be activated or not; appropriateness criterion - to identify the appropriateness of a method to a current situation; tie-breaker criterion - to choose just one method in case of draws, preference lists - ordering of methods by preference.

Having described the framework, the modelling languages are briefly analyzed and criticized.

3.2 MODEL-K

MODEL-K [Karbach, Voβ 92] is a language for operationalizing KADS models of expertise. It introduces KADS-specific modelling primitives. Thus, an operational model can then be built by attaching code to those primitives. It is considered as a language for the implementation of a final system [Fensel, Harmelen 93]. Systems are described in three layers: domain, inference and control layers. In the domain layer the concepts of the domain are represented. In the inference layer, the knowledge is represented in terms of knowledge sources (basic operators) and meta-classes (stores of knowledge). In the task layer, the methods [6], which determine the control that has to be applied to knowledge sources, are represented by 'task structures' (a control structure). A method is described in terms of its preconditions, goal, sub-methods and its body (sequencing knowledge). A sub-method can be another method or a knowledge source (basic inference).

Although MODEL-K does not provide explicit facilities for dynamic selection of methods, it presents interesting features and it is one of the fewest languages that implements KADS models with a strategic layer. This layer essentially consist of a meta-level system, and some additional constructs for accessing and modifying the object system. This layer is used to model reflective problem-solving. According to [Fensel, Harmelen 93] a reflective system consist of: an object system - modelled in terms of the three lower levels; one or more reflective modules (meta-level activities) - modelled in the three levels as well (in this case the domain layer contains the model of the object system); a scheduler - it defines the control flow (control strategy) between the reflective modules in order to decide what reflective module or object task comes next and what meta-level activities are applied before, during, or after the methods.

MODEL-K is an open ended language. It supports any number of meta-level activities. They can be added or modified. The activities can be method or system related activities though these concepts do not exist in MODEL-K. Their representation is explicit. Almost every object-level method can have attached a meta-level activity which controls its execution.

The scheduler provides a procedural language consisting of sequences, branches, and loops. Thus, flexible control strategies can be made (see figure 3).

[5] An abstract structure is the representation of the knowledge about objects in both levels and their relationships. It is important to note that sometimes an abstract structure might be considered a property of a meta-level object. In fact, a meta-level object property is an evident fact by itself whilst an abstract structure requires certain processing to determine its value.

[6] A task in MODEL-K represents a method in this work since the concept of task does not exist in MODEL-K.

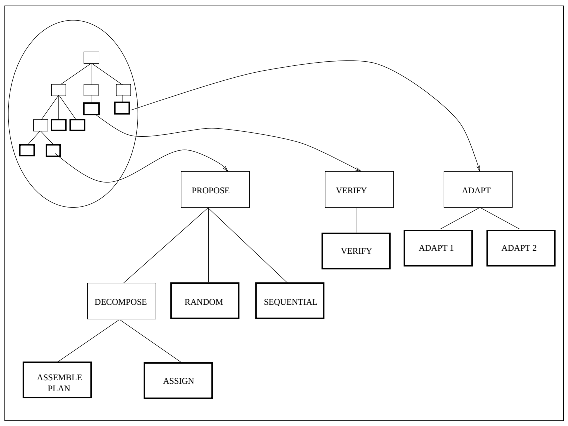

Figure 3: Control strategy and control structure in MODEL-K

In this language the objects at the meta-level (meta-objects) can be described as having as many properties as needed. There is no a predefined set of properties other than those in the structures in each layer.

In summary, MODEL-K is one of the few general purpose modelling language that provides a meta-level in which any number of objects and activities might be defined explicitly.

The disadvantages that can be appreciated in this language are:

- Control structure. This language does not have the concept of task. It just provides the concept of method. Therefore, its control structure consist uniquely of AND nodes.
- Meta-level activities. MODEL-K does not provide any primitive meta-level activity. The knowledge engineer has to develop the required activities from scratch.
- Control strategy. Although this language represents a flexible language, it does not provide any activity for the dynamic selection of methods. Therefore, the knowledge engineer has to design the control strategies from scratch.

  In particular, the scheduler defined in this language is a rigid scheduler (it prescribes a fixed method sequence) [Bartsch-Spörl, Bredeweg 91] as opposed to a flexible one which is necessary for this type of activity (OR nodes with attached activities). The flexibility of the scheduler has effect only on the top most methods in the control structure. The tasks below are not directly affected by the scheduler. Thus, an intermediate method cannot have an associated meta-level method. In other words, it cannot be an OR node. In the Sisyphus problem it would represent a shortcoming; since the propose task is an intermediate task it cannot have associated meta-level activities (see figure 3). A task might have an associated meta-level activity whenever the control knowledge that controls it is taken over by such meta-level activity. In other words the control knowledge is passed from the object level to the meta-level.
- Abstract structures. Since this language does not provide any facility for these type of structures. The knowledge engineer has to develop the required abstract structures from scratch.

3.3 TroTelC

TroTelC [Vanwekenhuysen, Rademakers 90] is a language with a computational framework which bridges the gap between a knowledge-level model and the actual implementation. It consists of an abstract language for describing knowledge and control. Its goal is to explicitly represent the tasks, methods and domain models of a knowledge-level model into a computational framework. Its underlying methodology is Components of Expertise [Steels 90]. Its control structure is a named task hierarchy. This hierarchy does not imply any control. In it a problem is decomposed into subproblems. Problems are known as tasks. A method specifies 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: return and fail. They indicate when a selector-sponsor has finished, that is whether the task associated with the selector has been achieved or not. These are the satisfaction and failure criteria respectively.

In summary, TIPS has a fixed and implicit control strategy based on a cycle of four meta-level activities, each using a criterion (abstract structure): verify satisfaction using distinguished sponsors, identify special cases using pattern matching, order the methods using appropriateness criterion, and choose only one using a priority list or a random process. This control strategy is the same for all the tasks.

The disadvantages that can be appreciated in this language are:

- Meta-level activities. Although TIPS has a number of method related activities it does not have an explicit meta-level. Thus, it does not allow additional method or system related activities. Other activities such as method monitoring, method assessment (can the method be activated?) cannot be represented explicitly.
- Control strategy. Although, in TIPS each task has its own control strategy, the sponsor-selector structure, it is a fixed control strategy which can be neither extended (e.g., adding new activities) nor modified (e.g., adding new abstract structures). Thus, for example, it is not possible to define strategies that iterate over the available methods in case of method failure, or that activate a task in case of the most appropriate method cannot be activated.
- Meta-level objects. TIPS has fixed meta-object descriptions. While methods are described using basically dynamic properties (run-time properties), tasks are only represented implicitly in the selector-sponsor structure [Punch, Chandrasekaran 93].
- Abstract structures. TIPS does not have applicability criteria. These criteria are sometimes embedded in the appropriateness criteria. Thus, sometimes it might be not clear why a method is not selected. It might be because it is not applicable or it is not the most appropriate.

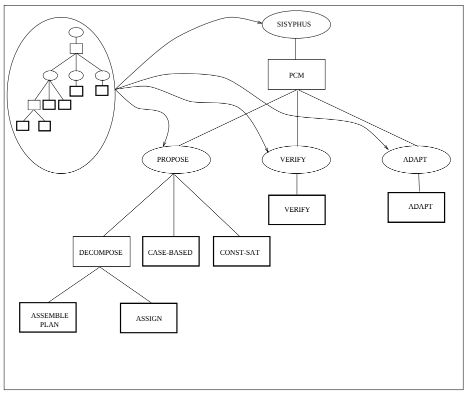

Figure 4: Control Strategy and Control Structure in LISA

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 flexible as any other object level strategy. LISA has a meta-control strategy to interpret its control strategy. This meta-control strategy is a control strategy as well but, in this case, it is fixed and implicit. In fact both are the same, one is the implicit representation of the other (this avoids an infinite tower of meta-levels). The system-provided control strategy basically consists of a single task with two associated methods: step-by-step (control strategy) and procedural (meta-control strategy). In order to avoid confusion the tasks and methods in the control strategy will be preceded by the term 'meta-'. The meta-tasks in the step-by-step meta-method are the following:

- Verify context. This meta-task verifies whether the current context is suitable for a given task.
- Construct a solution. This meta-task is the heart of LISA's control strategy. It is in charge of collecting, selecting and applying a method.
- Validate results. This verifies if the task has been achieved.

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 information is not described or is implicit [Prosser, Buchanan 94]).
- Most of the languages do not include all the method related activities or all the criteria already mentioned, namely: satisfaction, categorical, applicability, appropriateness, and tie-breaker. Applying one or two activities limits the flexibility of the language. Using only one or two criteria implies that the criteria used are more complex than they should. They embody different types of knowledge that depend on the activity and situation at hand.
- Current environments provide a single control strategy. Every task in the system uses the same strategy. A single control strategy not only implies an overhead in some systems, but also represents a shortcoming in others. This strategy represents an overhead in those systems in which not all tasks require the same control. For example, in the Sisyphus problem most of its tasks are carried out by a single method (see figure 2).

  It represents a shortcoming in those applications which require a specific control strategy in a number of its tasks. For example, in the propose task a strategy is needed that avoids selecting methods when the decomposition method (a backtracking one) has been activated previously, whereas in the PCM method such a strategy does not make sense.

  Note that in a number of those languages it is not clear whether a method will be activated once or several times, or be interleaved with other methods. It is not clear whether the selection activity has to be carried out every time till task satisfaction or just when a method has exhausted its results (backtracking method).
- All the languages with the exception of TIPS define their abstract structures in terms of local knowledge. It means that that knowledge does not involve any reference to other methods or tasks. Therefore, those languages do not provide any support for representing the relations and constraints between methods associated with different tasks. Associating a new method to a task might generate both a number of novel or invalid combinations. These relations and constraints should be represented in order to be known and to avoid generating strategies that include them during problem solving. The lack of these facilities does not encourage the addition of new methods at the lower levels in the control structure.
- Last but not least, all the languages described assume that the necessary features that allow choosing one method instead of another are always provided as part of the problem at hand. In the real world, most of the time this assumption is not satisfied and so it might generate strategies in which no method is applicable. In order to tackle this situation, a system needs to accept assumptions from the user or to use predefined strategies with their own assumptions For example, given just the employees and the rooms in the Sisyphus problem, a strategy might have one or several of the following assumptions: the problem is separable (a decomposition method is suitable), and simple (not over specified and so no relaxation methods are needed); a solution is enough (a non optimizer method is suitable); or there is limitation of time (use the fastest method).

[7] The lack of results in a method does does not necessarily imply that the problem is insoluble [Prosser, Buchanan 94].

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. At the same time it should provide a minimum set of those elements for easy of system development. This idea represents an improvement in current approaches.
- Flexibility might be increased providing facilities for defining not just one general control strategy but rather, providing the facilities for defining specific control strategies for each node in the control structure, in particular for tasks. Control strategies can be shared or assigned by default to several nodes in the control structure. This facility represents a generalization from current approaches.
- The problems, the methods defined for solving those problems, the activities and the control strategies that are applied to those methods are modelled explicitly at the knowledge-level and clearly separated from each other. The language is represented in a task-method control structure in which two types of control knowledge can be identified and clearly separated: knowledge for controlling the decomposition (sequencing) of subtasks or sub-methods, and knowledge for controlling meta-level activities. Both might be represented in procedural or heuristic terms depending on the specific application.

In the proposed approach there are two levels: meta and object. Both levels are controlled by an interpreter. At the meta-level there are basically two meta-level objects: tasks and methods. The underlying architecture resembles the so-called subtask-management [Harmelen 91]. The system follows an object level plan (control structure) and hands over control to the meta-level in specific situations such as task activation.

[8] According to [Bartsch-Spörl, Reinders 90] there are twelve groups of meta-level activities among which the dynamic selection of methods is just one of them.

```
[define                 propose
properties:
   [type                task]
   [goal 'The goal  of this  task is the  allocation (solution)  of the
         employees (components) of the YQT company into  the offices
         (resources) in the 3rd  floor. Only one solution is required']
   [input               components resources]
   [output              allocations]
abstract structures:
   [associated-methods  decomposition random-init sequential]
   [satisfaction-criteria
                        get-domain("allocations", "value") not = empty ]
   [appropriateness-criteria  external]
   [preferences         [random-init all] [sequential decomposition]]
activities:
   [collect-methods  take-assoc-methods]
   [select-a-method  appropriateness]
   [tie-breaker      take-first-one]
   [applicable       external]
   [apply-method     external]
   [satisfaction     eval-boolean]
control strategy:
   [satisfy-task-goal-strategy
        collect-methods with associated-methods
        while more methods do
            select-a-method with selection-criteria
            tie-breaker with preferences
            if applicable then
                apply-method
                if satisfaction with appropriateness-criteria then
                    return
                endif
            endif
        endwhile
   ]
```

Figure 5: The propose task in the Sisyphus problem

The environment envisioned tries to combine the search-space and the procedural-space in an effort to bring together certain aspects of KADS and GT methodologies. It means that the control structure can be described in procedural or heuristic terms

The following subsections provide an outline of MML.

4.1 Meta-level objects

MML has basically two meta-level objects: tasks and methods. Tasks represent the problems to be solved while methods the different ways in which problems can be solved. Both tasks and methods are related by means of a control structure. In order to describe those objects, MML relies in the distinction of four components: properties, abstract structures, meta-level activities, and control strategies. Every component consists of an attribute and a value. The value might be implicit or explicit. Implicit values refer to basic inferences, methods, or tasks, that once activated, can generate the expected (explicit) value or the desired behavior.

4.2 Tasks

Tasks characterize the set of problems to be solved. A task has a goal which is a specification of what needs to be achieved. A task by itself does not include, as part of its description, any specification of how it will be accomplished. At most it just describes the strategies that apply to the methods that are known to satisfy its goal. A task may be decomposed into subtasks. In this framework a tasks may or may not have 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.

For example, the propose task in the Sisyphus problem can be described as in figure 5. This description can be interpreted as: the task will succeed if there is an non empty allocation (components into resources). The methods are collected and then applied until task satisfaction or no more methods remain. They are selected by a primitive activity called appropriateness which makes decisions based on external criteria. In the case of a method draw it uses a tie-breaker activity. The activities applicable and apply-method are defined as external activities since they are activities that depend on the method selected.

4.3 Methods

Methods characterize the set of mechanisms: algorithms, plans of actions, sets of heuristics, that are available for the satisfaction of tasks. Methods, as tasks, are represented by the four components mentioned above. Methods do not have also, a fixed set of features. When a method have especial requirements such as: preparation of inputs, interpretation of output, its control strategy is the place to store the respective activities. This situation is very common when third party software is used. For example, statistical routines or simulation packages. For example, the decomposition method in the Sisyphus problem can be described as in figure 6. This description can be interpreted as: the decomposition method is a non-terminal non-backtracking method with two sub-methods (i.e. assemble-plan, assign-resources). Its control knowledge is procedural. It has two activities, one determines if the method is applicable and the other how to apply the method.

A method can be decomposed into subtasks or sub-methods. Both the decomposition and the sequencing knowledge needed to control such decomposition are stated explicitly. Among the properties that a method might have, the following are the most common: goal, input, output, type (terminal, non-terminal, basic inference), structure (method decomposition), ck-type (formalism involved in the description of its sequencing knowledge: procedural, heuristic), backtracks (capability of backtracking).

4.3.1 Types of methods

MML recognizes three different types of methods:

- Non-terminal. This method proposes the sequence of subtasks or sub-methods that carry out a task. The sequencing knowledge in these methods is either deterministic (procedural control) or non-deterministic (heuristic control). For example, the method PCM has three subtasks (propose, verify and adapt) which might be controlled heuristically. This approach does not worry about the control knowledge involved in the terminal methods and the basic inferences.
- Terminal. This method is a method whose internal structure is not known or is not worth worrying about (named black box). For example, statistical routines. These methods can only be applied rather than monitored for example. There is a distinguished set of terminal methods, namely basic inferences. These are methods which cannot be split into subtasks any more although they do not represent black boxes. For example, the method assign-resources; the user of the system may also participate as a terminal method.
- Meta-method. These methods are used to store some of the control strategies and activities. They are methods at the meta-level which have access to data not available at the object-level. Besides this fact, meta-methods are similar to methods. For example, the meta-method single-method (see figure 7) that can be used as a default strategy for the Sisyphus task. This description says that the method single-method consists of four activities: collect the method, verify its applicability, apply it and verify whether the task has been satisfied or not.

4.4 Properties

Properties are a collection of basic features of objects. An object might have any number of properties. Properties might be further classified. For example, in the method case, they are classified in four sets:

```
[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 different aspects (meta-knowledge) of the method to be used in activities. For example, the number of solution that the method provides.

4.5 Abstract structures

They are structures that categorize and represent the knowledge about the relations and constraints between the objects (at both levels) in the system. The different categories in this knowledge helps in its acquisition.

Several abstract structures have been identified. They might be included as primitives in a modelling language. For example:

- Satisfaction Criterion. This is a criterion based on domain data used to recognize whether the task has been carried out successfully.
- Failure Criterion. This is a criterion based on domain data used to recognize whether the task has not been satisfied.
- Associated methods. This is a list with the name of the methods that can carry out the task.
- Preferences. This is a partial ordering of the associated methods.
- Code. This is the explicit description of the sequencing knowledge for controlling its subtasks or sub-methods.
- Categorical criterion - This criterion avoids excessive deliberation.
- Applicability criteria - This criterion verifies if a method can be activated.
- Appropriateness criterion - This criterion identifies the appropriateness of a method to a given situation.
- Tie-breaker criterion - This criterion chooses just one method in case of draws.

4.6 Meta-level Activities

These activities are processes that run at a meta-level. They are the components that interpret and manipulate abstract structures. For example, activate a task, select a method, evaluate the results of a method, select a domain model. In other words, they are the semantics attached to abstract structures. Activities in MML might be specified as basic inferences, meta-methods or tasks, some of which might be primitive. For example, the activities might be simple as basic inference that evaluates a structure to true or false (e.g. satisfaction in figure 5), or so complex that a method (as in TIPS) or a task (as in LISA) is required (e.g. apply-method in figure 6). The decision of how to specify activities represents a departure from the current languages since MML has great flexibility in this respect. So, depending on the complexity of the activity, its representation varies.

A number method related control activities have been identified in the dynamic selection of methods (e.g., collection, ordering, selection, and evaluation of methods) and might be included as primitive activities in a modelling language (e.g. appropriateness in figure 5).

4.7 Control strategies

They are a collection of (control) statements that prescribe the order in which meta-level activities are applied. They not only incorporates the control knowledge associated with the activities but also the control knowledge specified by the methods to which those activities are related to.

For controlling meta-level activities MML uses high level control statements and built-in functions (i.e. conditional, loops). This represent another difference with current environments. The control in some of them is full of symbol-level constructs. In MML the need for using symbol-level constructs is minimized since some constructs and built-in functions are predefined and a library of them is included in the language.

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 representing the relations and constraints between methods associated with different tasks.
- Last but not least, all the languages described assume that the necessary features that allow choosing one method instead another are always provided as part of the problem at hand. In other words, their control strategies do not contemplate user points of view or problem types.

Having identified these shortcomings, we proposed a new modelling language, MML, which addresses them. The language incorporates the following features:

- MML is an open ended reflective language.
- MML describes objects in terms of both their properties and how those properties are used.
- The language facilitates dynamic selection of multiple methods along with other method-related activities such as diagnosis and repair.
- Control strategies are represented at each node in the control structure.
- The language distinguishes between control knowledge for decomposition of methods and control knowledge for method-related activities.

MML has been designed to capture some of the more desirable features required for solving problems with multiple methods. Initial experiments with the Sisyphus problem indicates that MML is not only a more flexible system but also it has led to an improvement in the syntax and semantics of the language and the specification of the knowledge about methods.

Further work will involve extending the range of methods defined in MML and evaluating its modelling capabilities on a variety of selected problems.

6 Acknowledgments

The author expresses his appreciation to Sharon Wood and Steve Easterbrook for their support and contributions. This work has been supported by a scholarship of the Consejo Nacional de Ciencia y Tecnología, México.

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] Generic Tasks and Task Structures: History, Critique and New Directions, in: Davis J.M., Krivine J.P. (editors), Second Generation Expert Systems, Spring Verlag, 1993.

[Chandrasekaran 90] Chandrasekaran B., Design Problem Solving: A Task Analysis, AI Magazine, Winter 90, pp 59-71.

[Clancey 85] Clancey W.J., Heuristic Classification, AI 27(3), 1985, pp 280-350.

[Delouis 93] Delouis Isabelle, LISA: Un Language Reflexif Pour La Modelisation Du Controle Dans Les Systemses a Base de Connaissances, DPhil Thesis, June 1993.

[Davis, Krivine 93] Davis J.M., Krivine J.P. (editors), Second Generation Expert Systems, Spring Verlag, 1993.

[Drouven et al. 92] Drouven U., Karbach W., Lorek D., Vo*β* A., Solving the office allocation task in reflective OFFICE-PLAN, in: Linster Marc, Sisyphus'92 Models of Problem Solving, Gesellschaft Fur Mathematik, Und Datenverarbitung MBH, 1992.

[Fensel, Harmelen 93] Fensel, D., Harmelen van F., A Comparition of Languages which Operationalize and Formalize KADS Model of Expertise, Research Report, no. 280, University of Karlsuhe, September 1993.

[Guerrero 94] MML, a Modelling Language with Dynamic Selection of MEthods, Guerrero Rojo Vicente, Research Paper CSRP 344, School of Cognitive & Computing Sciences, University of Sussex at Brighton, UK, 1994.

[Harmelen 92] Harmelen van F. (editor), Knowledge-level Reflection: Specifications and Architectures, ESPRIT Basic Research Project P3178 REFLECT, Doc. RFL/UvA/III/2, 1992.

[Harmelen 91] van Harmelen Frank, Meta-Level Inference Systems, Pitman, London, 1991.

[HayesRoth et al. 89] Hayes-Roth B., Washington R., Seiver A., Intelligent Monitoring and Control, IJCAI 89, 11th, Detroit Michigan, 1989, pp243-249.

[Karbach, Vo*β* 92] Karbach W., Vo*β* A. Reflecting about expert systems in MODEL-K in: Chapter 8, Harmelen van F. (editor), Knowledge-level Reflection: Specifications and Architectures ESPRIT Project P3178 KADS-II Doc. RFL/UvA/III/2, 1992.

[Linster 92] Linster Marc, Sisyphus'92 Models of Problem Solving, Arbeitspapiere Der GMD 630, Gesellschaft Fur Mathematik, Und Datenverarbitung MBH, March 1992.

[Linster 91] Linster Marc, Sisyphus'91 Models of Problem Solving, Arbeitspapiere Der GMD 663, Gesellschaft Fur Mathematik, Und Datenverarbitung MBH, July 1992.

[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.